Confiabilidad en Sistemas Desktop Cloud

Tesis de Doctorado
por
Carlos Eduardo Gómez Montoya

Enviado a la Facultad de Ingeniería de la
Universidad de los Andes
en cumplimiento parcial de los requisitos para el grado de
Doctor en Ingeniería

Asesor:	Prof. Dr. Harold E. Castro
Co-Asesor:	Prof. Dr. Carlos A. Varela
Jurados:	Prof. Dra. Sandra J. Rueda
	Prof. Dr. Andrei Tchernykh

Julio 2019

Abstract

Desktop cloud is an opportunistic platform that provides cloud computing services on desktop computers, typically located on a university or business campus. The desktop cloud systems take advantage of the idle resources of computers when their users perform routine activities, or when computers are fully available. A desktop cloud manages these resources to run virtual machines, with their operating systems and applications, without affecting the performance perceived by the users of the computers. Virtual machines allow users of a desktop cloud, typically researchers, to execute their academic or scientific applications at the same time as the processes of the users of desktop computers. Since the infrastructure of a desktop cloud is based on non-dedicated computers, these systems are more susceptible to fail, compared to traditional cloud computing providers. These platforms must face the interruptions and interference caused by the users of desktop computers and their applications. For example, the users of the computers can turn them off or restart them, disconnect them from the network, or execute demanding applications in computational capacity affecting the normal execution of virtual machines.

Desktop clouds have been used successfully in the execution of bag-of-tasks type applications, where a problem is solved by dividing it into independent tasks that run in parallel. In this type of applications, since the applications that run are independent, any of them can be executed on another physical machine, if a node on the platform fails. Other applications, with processes that communicate with each other, for example applications based on *Message Passing Interface (MPI)*, are more fragile when facing failures, since a failure in one node can affect the entire system.

There are several implementations of desktop clouds or similar systems, such as *CernVM*, *cuCloud*, *ad hoc cloud computing*, *GBAC* and *UnaCloud*, our case study. These systems take advantage of idle resources of the participating computers for the execution of virtual machines.

Due to the opportunistic access to resources, in general, desktop cloud systems are platforms that offer a *best effort* delivery service, in which there are no guarantees about the successful execution of the applications that are executed in the virtual machines. However, these platforms can move towards reliable service delivery despite the volatility of the computational resources on which it is supported. In this way, new applications can be executed and some guarantee in the service can be offered to its users.

In this doctoral thesis we have made and refined a fault analysis, considering different desktop cloud and similar systems, emphasizing *UnaCloud*. As a result of this analysis, we found that desktop cloud systems present failures mainly in two moments: during the provisioning and during the execution of virtual machines. We have proposed an extended chain of threats and we have used it in the identification of the main faults that occur in these two phases to determine their causes (anomalies, interruptions and errors) and their consequences. In addition, we suggest some mitigation strategies to counteract anomalies and interruptions.

We have compiled an initial set of strategies that allow us to mitigate the effects of the identified failures, we developed a functional prototype to respond to those failures using those strategies that have the greatest impact in our context and we evaluate their behavior.

Thanks to this analysis, we found that the provisioning of virtual machines has significant limitations in scalability. Virtual images are large files whose transmission through the network is delayed and prone to failures. In addition, reliability problems begin when the number of virtual machines that are going to run on the system begins to grow, particularly because of the space they occupy on disk, the use of the network and the time it takes for provisioning. On the other hand, the actions carried out by the users of the computers, or the applications that they launch, can interrupt the execution of the virtual machines running in the desktop cloud and ruin the work done up to that moment.

We have proposed a solution to mitigate the effects of failures in both the provisioning and the execution of virtual machines. First, implementation of a new virtual machine provisioning model for a desktop cloud based on a catalog of preconfigured virtual images, stored in multitasking writing disks and previously stored in the computers where the virtual machines run. Second, we have developed a global snapshot solution to store the state of a distributed system that runs in the virtual machines of a desktop cloud. Our implementation allows us to obtain multiple global snapshots and a mechanism to resume execution consistently from any of them, without missing or duplicated messages. We have validated the software developed through functional and performance tests and verified that the proposed solutions can be used to improve the reliability of a desktop cloud system.

Keywords: Desktop clouds, Reliability, Fault tolerance, Checkpointing, Global snapshot, Virtual Machines provisioning.

Resumen

Desktop cloud es una plataforma oportunista que proporciona servicios de computación en la nube sobre computadores de escritorio, los cuales típicamente se encuentran en un campus universitario o empresarial. Los sistemas desktop cloud aprovechan los recursos ociosos de los computadores cuando sus usuarios realizan actividades de rutina, o cuando los computadores están completamente disponibles. Un desktop cloud administra estos recursos para ejecutar máquinas virtuales, con sus sistemas operativos y aplicaciones, sin afectar el rendimiento percibido por los usuarios de los computadores. Las máquinas virtuales les permite a los usuarios de un desktop cloud, típicamente investigadores, ejecutar sus aplicaciones académicas o científicas al mismo tiempo que los procesos de los usuarios de los computadores de escritorio. Dado que la infraestructura de un desktop cloud está basada en equipos no dedicados, estos sistemas son más susceptibles a presentar fallas en el servicio, en comparación con proveedores tradicionales de computación en la nube. Estas plataformas deben enfrentar las interrupciones e interferencia causadas por los usuarios de los computadores de escritorio y sus aplicaciones. Por ejemplo, los usuarios de los computadores pueden apagarlos o reiniciarlos, desconectarlos de la red, o ejecutar aplicaciones exigentes en capacidad computacional afectando la ejecución normal de las máquinas virtuales.

Los desktop clouds han sido utilizados exitosamente en la ejecución de aplicaciones tipo bolsa de tareas, donde un problema se resuelve al dividirlo en tareas independientes que se ejecutan en paralelo. En este tipo de aplicaciones, dado que las aplicaciones que se ejecutan son independientes, cualquiera de ellas puede ser ejecutada en otra máquina física, si un nodo en la plataforma llega a fallar. Otras aplicaciones, con procesos que se comunican entre ellos, por ejemplo aplicaciones basadas en *MPI*, son más frágiles frente a las fallas, ya que una falla en un nodo puede afectar todo el sistema.

Existe una variedad de implementaciones de desktop clouds o sistemas similares, tales como *CernVM*, *cuCloud*, *ad hoc cloud computing*, *GBAC* y *UnaCloud*, nuestro caso de estudio. Estos sistemas aprovechan los recursos ociosos de los computadores participantes para la ejecución de máquinas virtuales.

Debido al acceso oportunista a los recursos, por lo general, los sistemas desktop cloud son plataformas que ofrecen un servicio de entrega de *mejor esfuerzo*, en la cual no hay garantías acerca de la ejecución exitosa de las aplicaciones que se ejecutan en las máquinas virtuales. Sin embargo, estas plataformas pue-

den avanzar hacia la entrega confiable de servicios a pesar de la volatilidad de los recursos computacionales en los que se soporta. De este modo, nuevas aplicaciones pueden ser ejecutadas y alguna garantía en el servicio puede ser ofrecida a sus usuarios.

En esta tesis doctoral hemos realizado y refinado un análisis de fallas, considerando diferentes sistemas desktop cloud y similares, haciendo énfasis en *UnaCloud*. Producto de ese análisis encontramos que los sistemas desktop cloud presentan fallas principalmente en dos momentos: durante el aprovisionamiento y durante la ejecución de máquinas virtuales. Hemos propuesto una cadena extendida de amenazas y la hemos utilizado en la identificación de las principales fallas que ocurren en estas dos etapas para determinar sus causas (anomalías, interrupciones y errores) y sus consecuencias. Además, sugerimos algunas estrategias de mitigación para contrarrestar las anomalías e interrupciones.

Hemos recopilado un conjunto inicial de estrategias que permiten mitigar los efectos de las fallas identificadas, desarrollamos un prototipo funcional para dar respuesta a esas fallas usando aquellas estrategias que tienen mayor impacto en nuestro contexto y evaluamos su comportamiento.

Gracias a este análisis, encontramos que el aprovisionamiento de máquinas virtuales tiene limitaciones significativas en la escalabilidad. Las imágenes virtuales son archivos grandes cuya transmisión a través de la red es demorada y propensa a fallas. Además, los problemas de confiabilidad empiezan cuando el número de máquinas virtuales que se van a ejecutar en el sistema empieza a crecer, particularmente por el espacio que ocupan en disco, el uso de la red y el tiempo destinado al aprovisionamiento. Por otra parte, las acciones realizadas por los usuarios de los computadores, o las aplicaciones que lanzan, pueden interrumpir la ejecución de las máquinas virtuales corriendo en el *desktop cloud* y arruinar el trabajo realizado hasta ese momento.

Hemos propuesto una solución para mitigar los efectos de las fallas tanto en el aprovisionamiento como en la ejecución de máquinas virtuales. En primer lugar, implementación de un nuevo modelo de aprovisionamiento de máquinas virtuales para un *desktop cloud* basado en un catálogo de imágenes virtuales preconfiguradas, almacenadas en discos en modo de escritura *multiattach* y previamente almacenadas en los computadores donde corren las máquinas virtuales. En segundo lugar, hemos desarrollado una solución de *snapshot* global para almacenar el estado de un sistema distribuido que corre en las máquinas virtuales de un desktop cloud. Nuestra implementación permite obtener múltiples *snapshots* globales y un mecanismo para reanudar la ejecución de manera consistente a partir de cualquiera de ellos, sin mensajes perdidos o duplicados. Hemos validado el software desarrollado mediante pruebas funcionales y de desempeño y hemos verificado que las soluciones propuestas pueden ser utilizadas en el mejoramiento de la confiabilidad de un sistema desktop cloud.

Palabras clave: Desktop clouds, Confiabilidad, Tolerancia a fallas, Checkpointing, Global snapshot, Aprovisionamiento de máquinas virtuales.

Agradecimientos

Quiero expresar mis agradecimientos a todas las personas que me apoyaron durante la realización de mis estudios de doctorado. En particular:

A Harold Castro, mi asesor de doctorado, por su confianza, su acompañamiento permanente, sus consejos, su paciencia y su calidad humana.

A Carlos Varela, mi co-asesor internacional por su hospitalidad y por el tiempo que me dedicó especialmente durante mi pasantía en Rensselaer Polytechnic Institute en Troy, New York, Estados Unidos.

A Sandra Rueda y Andrei Tchernykh, quienes participaron como jurados en la evaluación de esta tesis doctoral.

A Jaime Chavarriaga, por por escucharme y ayudarme a encontrar soluciones en las muchas ocasiones en las que le consulté.

A David Camilo Bonilla, por todo el apoyo que me brindó durante los últimos semestres, especialmente en la realización de cientos de pruebas de los componentes de software desarrollados en esta tesis.

A las instituciones que financiaron mis estudios de doctorado: La Universidad del Quindío, Colciencias y el Departamento de Ingeniería de Sistemas y Computación de la Universidad de los Andes.

Al Departamento de Ingeniería de Sistemas y Computación de la Universidad de los Andes a cargo de su director Yezid Donoso, a su personal administrativo: Doris Bolívar, Carlos Rozo y Yomara Rincón, por su colaboración en todos los asuntos académicos y administrativos durante estos años; y al personal de la Administración de los Laboratorios: Jonatan Legro, Shurys Zambrano y Juan David Moreno, por su colaboración en la realización de pruebas en las salas de cómputo.

A mis compañeros y amigos del equipo de trabajo de *UnaCloud*: Jesse Padilla, César Forero, Eduardo Rosales y César Díaz.

A mis compañeros y amigos de la Universidad de los Andes: Olga Vega, Adriana Gómez, Iván Alfonso, Diego Castiblanco y Carlos Lozano, por su compañía el el apoyo recibido durante mi estadía en la universidad.

A Shigeru Imai, por su amabilidad y apoyo durante el tiempo de mi pasantía.

A mis compañeros y amigos de la Universidad del Quindío: Julián Esteban Gutiérrez, Christian Candela y Luis Eduardo Sepúlveda.

A mi padre, Gilberto Gómez, a mis hermanas Margarita María, María Constanza y Adriana María, por su incansable apoyo durante todo este tiempo.

A mis hijos Pablo y Martín, quienes tuvieron que vivir mi ausencia durante estos años.

A Claudia, quien con su amor y compañía me ayudó a tener la fuerza necesaria para terminar esta tesis doctoral.

Contenido

Abstract I

Resumen III

Agradecimientos V

Publicaciones XVII
Artículos publicados en revista indexada . . . XVII
Artículos presentados en conferencia . . . XVII
Artículos presentados en workshop . . . XVII
Artículos aceptados para presentar en conferencia . . . XVIII
Artículos aceptados para presentar en workshop . . . XVIII
Foro doctoral . . . XVIII
Software desarrollado . . . XVIII

1 Introducción 1
1.1. Generalidades . . . 1
1.2. Motivación . . . 4
1.3. Objetivos . . . 5
1.3.1. Objetivo general . . . 5
1.3.2. Objetivos específicos . . . 5
1.4. Planteamiento del problema . . . 6
1.5. Metodología . . . 6
1.6. Contribuciones . . . 6
1.7. Organización del documento . . . 8

2 Conceptos generales 11
2.1. Sistemas de computación voluntaria y sistemas desktop grid . . 11
2.2. Computación en la nube y sistemas desktop cloud . . . 14
2.2.1. Computación en la nube . . . 14
2.2.2. Sistemas desktop cloud . . . 14
2.2.2.1. Funcionamiento de un sistema desktop cloud . 16
2.3. Fiabilidad en sistemas de cómputo . . . 17
2.3.1. Concepto de fiabilidad . . . 17
2.3.2. Atributos de la fiabilidad . . . 18

2.3.3. Amenazas contra la fiabilidad 19
2.3.4. Medios para alcanzar la fiabilidad 20
2.4. *UnaCloud* 20
2.4.1. Descripción general 20
2.4.2. Modelo actual del sistema 21
2.4.3. Usuarios 22
2.4.4. Conceptos fundamentales 23
2.4.5. Arquitectura de *UnaCloud* 24
2.4.6. Despliegue de máquinas virtuales 27
2.4.7. Infraestructura disponible 29
2.5. Resumen 29

3 Análisis de fallas en desktop clouds 31
3.1. Implementaciones de desktop clouds y sistemas similares y su forma de enfrentar fallas 32
3.2. Análisis de fallas 36
3.2.1. El desktop cloud no puede aprovisionar la máquina virtual 39
3.2.2. La ejecución de la aplicación es interrumpida 41
3.3. Discusión 43

4 Nuevo modelo de aprovisionamiento de máquinas virtuales 45
4.1. Ofrecer un catálogo de imágenes virtuales preconfiguradas . . . 46
4.2. Establecer un mecanismo eficiente de gestión de espacio en disco 47
4.3. Establecer un mecanismo para reducir la transmisión de archivos 49
4.4. Validación 50
4.4.1. Aprovisionamiento de grupos grandes de máquinas virtuales 50
4.4.2. Errores en el aprovisionamiento de MVs 51
4.5. Discusión 52

5 *Snapshot* global como solución a las fallas en ejecución 55
5.1. Conceptos generales 55
5.1.1. *Checkpointing* 55
5.1.2. *Checkpointing* a nivel de virtualización 57
5.1.3. *Snapshots* en *Oracle VirtualBox* 58
5.1.4. Algoritmos para *Snapshots* globales 59
5.1.5. *Snapshots* en ambientes de red virtualizados 60
5.1.6. Procesamiento de paquetes de red y *coloring* 62
5.2. Desafíos 62
5.3. Global Snapshot 64
5.3.1. Elementos empleados en la solución 64
5.3.1.1. Mecanismos de confiabilidad de TCP 64
5.3.1.2. Colores 64
5.3.1.3. Filtrado de paquetes de red 65
5.3.1.4. Protocolo simple de coordinación 65
5.3.2. Implementación 69

5.3.3. Arquitectura del sistema 71
5.4. Análisis de correctitud 73
5.4.1. Aplicaciones con TCP como protocolo de capa de transporte 74
5.4.2. Aplicaciones con UDP como protocolo de capa de transporte 76
5.5. Evaluación y resultados 77
5.5.1. Configuración de Experimentos 77
5.5.2. Sistema distribuido 78
5.5.2.1. *TestManager* 78
5.5.2.2. *RingTokenPassing* 79
5.5.3. Experimentos y Resultados 82
5.6. Reanudar la ejecución de una máquina virtual 85
5.6.1. Reanudar en la misma máquina física 86
5.6.2. Reanudar en otra máquina física 86
5.6.3. Dónde reanudar la ejecución de una máquina virtual a partir de un *snapshot* 88

6 Conclusiones y trabajo futuro **93**
6.1. Lecciones aprendidas 96
6.2. Acerca de *UnaCloud* como caso de estudio 96
6.3. Acerca de la solución 97
6.4. Impacto del trabajo realizado 97
6.5. Trabajo futuro 98

Bibliografía **101**

Lista de figuras

1.1. Consumo promedio de CPU a lo largo del día. (a) 2011, (b) 2018 . 1

2.1. Visión general de un sistema *desktop cloud* 15
2.2. Funcionamiento de un sistema *desktop cloud* 16
2.3. Taxonomía de la fiabilidad 18
2.4. Cadena de amenazas 19
2.5. Arquitectura de *UnaCloud* 25
2.6. Máquina de estados de un despliegue en *UnaCloud* 27
2.7. Despliegue en *UnaCloud* – Aprovisionamiento 28
2.8. Despliegue en *UnaCloud* – Ejecución 28

3.1. Ejemplo de una cadena de amenazas 37
3.2. Ejemplo de una cadena extendida de amenazas 37
3.3. Cadena extendida de amenazas 38
3.4. Análisis de la falla F_1: *El desktop cloud no puede aprovisionar la máquina virtual* 39
3.5. Análisis de la falla F_1: *El desktop cloud no puede aprovisionar la MV* — Anomalías y Estrategias de mitigación 40
3.6. Análisis de la falla F_2: *La ejecución de la aplicación es interrumpida* 41
3.7. Análisis de la falla F_2: *La ejecución de la aplicación es interrumpida* — Anomalías y Estrategias de mitigación 42

4.1. Ejemplo del uso de discos *multiattach* 48

5.1. Ejemplo de un árbol de *snapshots* 57
5.2. Información de un disco en *Oracle VirtualBox* 58
5.3. Operación de VIOLIN 61
5.4. Corte de eventos 61
5.5. Colores en los nodos 65
5.6. Filtrado de paquetes 65
5.7. Protocolo de *snapshot* global — Fase 1: Verificación previa 66
5.8. Protocolo de *snapshot* global — Fase 2: Obtener el snapshot global 66
5.9. Protocolo de reanudación — Fase 1: Sondeo 68
5.10. Protocolo de reanudación — Fase 2: Asignación de punto de restauración 68
5.11. Protocolo de reanudación — Fase 3: Reinicio de ejecución 69

5.12. Vista general del sistema 70
5.13. Ejemplo de configuración de reglas de *iptables* en máquinas virtuales 71
5.14. Interacción de componentes — (a) Fase de registro (b) Fase de ejecución 72
5.15. Diagrama de secuencia – Fase de registro 73
5.16. Diagrama de secuencia – Fase de ejecución 73
5.17. Categorías de los mensajes 74
5.18. Mensajes y acuses de recibo 76
5.19. RCPBT. Fase de registro. 80
5.20. RCPBT. Fase de operación. 81
5.21. RCPBT. Fase de finalización 81
5.22. Tiempo de *snapshot* global vs Tiempo de *snapshot* local: 1 MV por cada MF 82
5.23. Tiempo de *snapshot* global vs Tiempo de *snapshot* local: 4 VMs . 83
5.24. Tiempo de *snapshot* global vs Tiempo de *snapshot* local: 8 VMs . 84
5.25. Tiempo de *snapshot* global vs Tiempo de *snapshot* local: 16 VMs . 84
5.26. Tiempo de *snapshot* global vs Proporción entre máquinas virtuales y máquinas físicas 85
5.27. Proceso para reanudar la ejecución de una MV en la misma MF. . 86
5.28. Proceso para reanudar la ejecución de una MV en otra MF. 87

Lista de tablas

2.1. Comparación entre DG Voluntario y DG Empresarial 13
2.2. Condiciones requeridas para ejecutar máquinas virtuales en *UnaCloud* 23
2.3. Perfiles de hardware en *UnaCloud* 24
2.4. Algoritmos de ubicación 25
2.5. Especificaciones de los computadores de la sala Waira 29
2.6. Especificaciones de los computadores de la sala Alan Turing 29

3.1. Estrategias de protección frente a fallas para ofrecer un servicio confiable en desktop clouds 35
3.2. Notación para identificar los elementos de la cadena extendida de amenazas 38

4.1. Tiempo de aprovisionamiento de máquinas virtuales en *UnaCloud* 51

5.1. Formato del archivo de resultados en el sistema desarrollado para pruebas 79
5.2. Archivos de una máquina virtual luego de 4 *snapshots* 89
5.3. Archivos requeridos para reanudar la ejecución de una máquina virtual con 4 *snapshots* en otra máquina física — Modelo actual de aprovisionamiento de máquinas virtuales 89
5.4. Archivos requeridos para reanudar la ejecución de una máquina virtual con 4 *snapshots* en otra máquina física — Nuevo modelo de aprovisionamiento de máquinas virtuales 90

Acrónimos

BoT bolsa de tareas. 3

CV Computación Voluntaria. 2

DC Desktop Cloud. 2

DG Desktop Grid. 2

DGE Desktop Grid Empresariales. 12

DGV Desktop Grid Voluntarios. 12

GS Global Snapshot. 69

HPC computación de alto rendimiento. 3

IaaS Infrastructure as a Service. 14

IV Imagen Virtual. 5

MV Máquina Virtual. 3

PaaS Platform as a Service. 14

SaaS Software as a Service. 14

SLA Service Level Agreements. 14

SSH Secure SHell. 23

TI Tecnología Informática. 13

VCaaS Volunteer Computing as a Service. 33

Publicaciones

Artículos publicados en revista indexada

1. **Harvesting idle CPU resources for desktop grid computing while limiting the slowdown generated to end-users**.

   E. Rosales, G. Sotelo, A. de la Vega, C. Díaz, C. E. Gómez, and H. Castro. Cluster Computing, 18(4), 1331-1350.

Artículos presentados en conferencia

1. **Global Snapshot File Tracker**.

   C. E. Gómez, J. Chavarriaga, D. C. Bonilla, and H. E. Castro. In H. Flórez, C. Díaz, and J. Chavarriaga, editors, Applied Informatics, pages 90–104, Cham, 2018. Springer International Publishing. Bogotá, Colombia.

2. **Fault Characterization and Mitigation Strategies in Desktop Cloud Systems**.

   C. E. Gómez, J. Chavarriaga, and H. E. Castro. In Latin American High Performance Computing Conference — CARLA2018, pages 322–335, Cham, 2019. Springer International Publishing. Piedecuesta, Colombia.

3. **Global snapshot of a distributed system running on virtual machines**.

   C. E. Gómez, H. E. Castro, and C. A. Varela. In 2017 29th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD), pages 169–176. IEEE, 2017. Campinas, Brazil.

Artículos presentados en workshop

1. **Federated Campus Cloud Colombian Initiative**.

   C. Díaz, C. E. Gómez, H. Castro, C. Barrios and H. Bolívar. In 2016 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid). Cartagena, Colombia.

Artículos aceptados para presentar en conferencia

1. **UnaCloud: Ejecutando Aplicaciones HPC en las Salas de Computadores de la Universidad**.

   C. E. Gómez, J. Chavarriaga, end H. E. Castro. In Novena Conferencia de Directores de Tecnología de Información y Comunicación en Instituciones de Educación Superior, TICAL2019. Cancún, México, Septiembre 2 - 4, 2019.

Artículos aceptados para presentar en workshop

1. **Improving Reliability for Provisioning of Virtual Machines in Desktop Clouds**.

   C. E. Gómez, J. Chavarriaga, H. E. Castro, and A. Tchernykh. In 12th Workshop on Resiliency in High Performance Computing (Resilience) in Clusters, Clouds, and Grids. Göttingen, Germany, August 26 - 30, 2019

Foro doctoral

1. **Desktop Cloud Systems: Offering a Dependable Service**.

   C. E. Gómez, H. E. Castro, and C. A. Varela. In 2018 IEEE 32nd International Parallel and Distributed Processing Symposium (IPDPS) PhD Forum. *IEEE Technical Committee on Parallel Processing (TCPP) Award.* Vancouver, Canada.

2. **VM Reliability in Opportunistic Desktop Cloud Environments**.

   C. E. Gómez, H. E. Castro, y C. A. Varela. In Latin American High Performance Computing Conference — CARLA2018. *Best Postgraduate Symposium Proposal Award.* Piedecuesta, Colombia.

Software desarrollado

1. **Global Snapshot**.

   C. E. Gómez, J. Chavarriaga, y H. E. Castro. 2018.

   El sistema Global Snapshot permite obtener un *snapshot* global consistente en el contexto de un desktop cloud manteniendo la semántica del sistema sin modificar las aplicaciones que se ejecutan en MVs o hipervisores.

2. **Ring-based Cloud Performance Benchmark Tool**.

   C. E. Gómez, J. Chavarriaga, y H. E. Castro. 2018.

   Es un sistema de análisis de desempeño, basado en anillo, creado para su ejecución en ambientes de computación en la nube. El sistema ha sido

desarrollado para su ejecución automática en máquinas virtuales formando sistemas distribuidos de múltiples participantes. El sistema permite determinar la duración de la ejecución del sistema a partir del intercambio de un número específico de mensajes que son enviados por un participante a su sucesor.

3. **Global Snapshot File Tracker**.

   C. E. Gómez, J. Chavarriaga, D. Bonilla y H. E. Castro. 2018.

   Esta herramienta analiza y procesa el archivo de configuración de una máquina virtual (el archivo .vbox). Como resultado se producen dos archivos en formato de valores separados por coma (*csv — comma separated value*). En el primero, se reúnen en una única estructura de datos (fácil de entender) los nombres de todos los archivos que comprenden el estado de una máquina virtual, es decir, los archivos de disco virtual (.vdi) y los archivos con el estado de la memoria (.sav) en el tiempo de cada *snapshot*. Esto permite saber en cada caso, cuáles archivos pertenecen a cuál *snapshot*. En el segundo, se genera una lista con los archivos que deben ser replicados para que se pueda continuar la ejecución de la máquina virtual en otro computador.

4. **Hypervisor Remote Control**.

   C. E. Gómez, J. Chavarriaga, y H. E. Castro. 2018.

   Es una herramienta que facilita la administración remota de máquinas virtuales (MVs) y permite controlar el hipervisor y realizar experimentos de investigación. Este programa no solo admite operaciones para crear, modificar y controlar máquinas virtuales, sino que también admite secuencias complejas de comandos de operaciones para automatizar las funciones principales del hipervisor.

5. **Multiattach Manager**.

   C. E. Gómez, D. Bonilla, J. Chavarriaga, y H. E. Castro. 2019.

   Es un conjunto de *scripts* que permite la realización de las siguientes tareas: precargar un disco virtual, configurado en modo de escritura *multiattach*, en el administrador de medios virtuales del hipervisor; crear una máquina virtual y conectarla a un disco *multiatach* precargado; crear una máquina virtual a partir de un disco diferencial; clonar una máquina virtual a partir de un disco diferencial para su ejecución en la misma máquina física; crear una máquina virtual a partir de un disco diferencial para su ejecución en otra máquina física; migrar una máquina virtual de una máquina física a otra; eliminar una máquina virtual; y eliminar un disco *multiattach* del adaptador de medios virtuales del hipervisor.

Capítulo 1

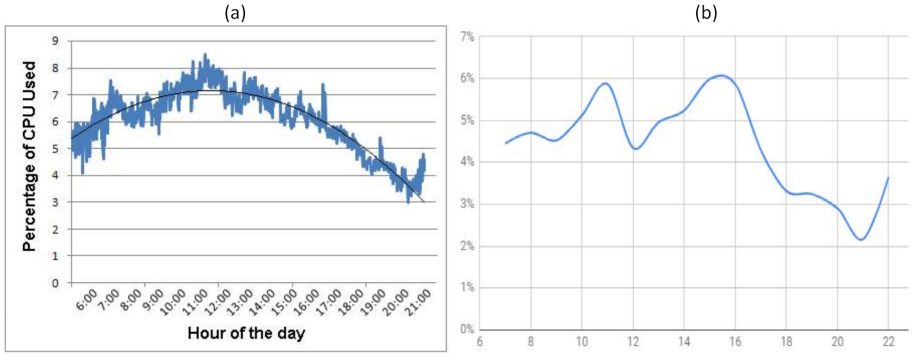
Introducción

1.1. Generalidades

Algunos estudios han mostrado que los usuarios de los computadores utilizan los recursos computacionales muy por debajo de su capacidad. Esto ocurre especialmente en sesiones interactivas [1] [2] donde el usuario está utilizando los dispositivos de entrada y salida del computador, tales como el teclado, el ratón y la pantalla. Otras publicaciones indican que el uso del procesador en salas de computadores de instituciones educativas es cercano al 5 % [3] [4].

En la Universidad de los Andes hemos analizado el consumo promedio de CPU en diferentes momentos del día, en dos oportunidades, en 2011 [5] y en 2018 [6].

Figura 1.1: Consumo promedio de CPU a lo largo del día. (a) 2011, (b) 2018



La Figura 1.1 muestra el consumo promedio de CPU entre las 7:00 am y las 10:00 pm en los equipos de las salas de cómputo del Departamento de Ingeniería de Sistemas y Computación. En la Figura 1.1 (a), el consumo promedio en horas del día varía entre 5 % y 8 %, bajando a cerca del 3 % al 5 % en horas de la noche. Por su parte, en la Figura 1.1 (b), se puede notar que el consumo

permanece entre 4 y 6 % casi todo el día, llegando al 2 % en horas de la noche, cerca del horario de cierre de las salas.

A pesar de la diferencia en tiempo y que las especificaciones técnicas de los computadores de ese entonces son diferentes a las actuales, el comportamiento en el uso de los recursos computacionales por parte de los usuarios de los equipos es similar. De acuerdo con la capacidad ociosa actual, sigue existiendo una gran oportunidad para su aprovechamiento.

Muchos sistemas han sido desarrollados para aprovechar los recursos de los computadores total o parcialmente ociosos para la ejecución de tareas (típicamente de tipo científico o académico) sin afectar al usuario. Entre ellos se pueden destacar las plataformas de *Computación Voluntaria (CV)* o *Volunteer Computing* [7], los sistemas *Desktop Grid (DG)* [8] y los sistemas *Desktop Cloud (DC)* [9] [10]. Estos sistemas sirven de intermediarios para ejecutar proyectos académicos o científicos [11]. Las características de estos sistemas de computación voluntaria y desktop grid son similares y no son muy claras las diferencias entre ellos, a tal punto que autores como McGilvary y Andrew [12] señalan que los sistemas computación voluntaria en ocasiones son descritos como desktop grid.

Por otra parte, de acuerdo con Alwabel, et al. [9], Cunsolo, et al.[13] y Fedak [14], entre otros, el concepto de desktop cloud surge de la combinación de la tecnología de virtualización con los sistemas de computación voluntaria y desktop grid.

DC se refiere a una plataforma oportunista que proporciona servicios de computación en la nube sobre computadores de escritorio de uso común (*desktops*). Estos sistemas aprovechan los recursos computacionales que resultan ociosos en salas de cómputo cuando sus usuarios realizan actividades regulares [9].

Un DC administra estos recursos para ejecutar máquinas virtuales, con sus sistemas operativos y aplicaciones, sin que los usuarios de los computadores perciban un deterioro en el desempeño del computador o vean comprometida su seguridad.

Típicamente, estos DCs ofrecen un subconjunto de servicios de infraestructura (IaaS) proporcionados por plataformas públicas y privadas basados en infraestructura dedicada, tales como *OpenStack* [15], *Amazon Web Services* [16] y *Microsoft Azure* [17].

Los sistemas DC, tales como *UnaCloud* [10] [18] y *CernVM* [19] [20] ejecutan máquinas virtuales en computadores de escritorio ubicados en campus universitarios o empresariales, donde la capacidad agregada de los recursos ociosos es significativa [4].

Usando estas plataformas DC, los investigadores pueden ejecutar trabajos científicos y académicos donde el procesamiento es ejecutado en máquinas virtuales que corren al mismo tiempo que los programas lanzados por los usuarios de los computadores [21]. Sin embargo, dado que su infraestructura no está basada en centros de datos especializados con equipos dedicados, los DC son más susceptibles a presentar fallas en el servicio que los proveedores tradicionales de computación en la nube.

Un desktop cloud es una plataforma agreste en la que los recursos computacionales, tales como la capacidad de procesamiento, la capacidad de almacenamiento y la red, son compartidos con el usuario de la máquina física. Desafortunadamente, la presencia de los usuarios de los computadores al mismo tiempo que se están ejecutando las máquinas virtuales, es un factor crítico para el funcionamiento de un DC. Lo anterior, a pesar de que estas máquinas virtuales generalmente corren en segundo plano (*background*) y con baja prioridad, muchas veces sin que los usuarios lo perciban.

Los DC, al igual que los sistemas de computación voluntaria y los sistemas desktop grid, han sido utilizados exitosamente en la ejecución de aplicaciones tipo *BoT* (*Bag of Tasks*) o *bolsa de tareas*, un patrón muy popular en la comunidad *HPC* (*High Performance Computing*) o *computación de alto rendimiento* [22], en el que un problema que se desea resolver es dividido en tareas independientes ejecutables en paralelo [23].

En este tipo de aplicaciones, las fallas en unos nodos no impiden que otros puedan continuar con su operación. Por lo tanto, cualquier tarea puede asignarse a otro computador en caso de una falla en los nodos de la plataforma, ocultando algunas limitaciones en su confiabilidad, uno de sus principales desafíos de las plataformas oportunistas, y uno de los más importantes aspectos por mejorar en los DCs.

Por otra parte, existen aplicaciones cooperativas en las cuales los procesos se comunican entre ellos para cumplir con su objetivo. Su ejecución puede verse afectada por la ocurrencia de fallas en un solo nodo, interrumpiendo o arruinando la ejecución de todo el sistema.

Estas plataformas desktop cloud administran la ejecución de máquinas virtuales y controlan su ejecución. Además, le permite al administrador del sistema conocer la evolución de cada *Máquina Virtual (MV)*, desde que es solicitada hasta que finaliza su ejecución, bien sea porque termina el trabajo, porque el usuario la apaga o porque ha ocurrido una falla en el servicio.

Por ejemplo, *UnaCloud* [10] es nuestra implementación de un DC. Este sistema soporta las necesidades computacionales para el desarrollo de trabajos científicos y de investigación en un ambiente académico como el que se encuentra en una universidad. Aunque la plataforma monitorea permanentemente el estado de la ejecución de las MVs, esta información no permite identificar las fallas presentadas ni implementar correctivos apropiados. Además, información histórica acerca de fallas en el servicio y problemas no es usada en la toma de decisiones de nuevos despliegues.

El usuario de *UnaCloud* espera una plataforma funcional, que se cuide a sí misma, que no le quite al investigador los recursos asignados, y que reaccione frente a eventos, que al ocurrir, puedan afectar la continuidad de las aplicaciones. Sin embargo, *UnaCloud* presta un servicio de *mejor esfuerzo* en el cual le deja al usuario una serie de responsabilidades que no siempre son fáciles de cumplir. Por ejemplo, el usuario debe verificar si las tareas fueron ejecutadas satisfactoriamente, y si es necesario, iniciar de nuevo la ejecución.

UnaCloud no tiene mecanismos de protección y es frágil, por lo tanto no ofrece garantía de que una MV va a finalizar su ejecución. Sin embargo, noso-

tros consideramos que *UnaCloud* puede proporcionar mecanismos de protección para ofrecer mejores garantías de éxito en la finalización de la ejecución de MVs.

Por su parte, la confiablidad es la capacidad de un sistema de cómputo para mantener la continuidad en la entrega del servicio durante un período de tiempo específico[24].

En esta tesis doctoral, utilizaremos *UnaCloud* como caso de estudio, lo que nos permite extraer características relevantes acerca la confiabilidad de los sistemas DC.

Tener un servicio confiable es atractivo para cualquier proveedor de servicios de computación en la nube, como lo plantea Kangarlou [25]. Un servicio confiable sería útil para extender los servicios de un DC en general y *UnaCloud* en particular. De este modo, se incrementa el tipo de aplicaciones que podrían ser ejecutadas. Por ejemplo, sería posible ejecutar un sistema distribuido formado por aplicaciones con procesos que se comunican entre ellos, y finalizar normalmente aún si ocurren anomalías en la infraestructura.

1.2. Motivación

Aunque en principio está definido lo que un sistema desktop cloud hace por sus usuarios, hay asuntos sin resolver como la confiabilidad y la tolerancia a fallas, tal como lo señalan algunos autores.

Cunsolo, et al. [13] señala que la confiabilidad es un desafío en los sistemas de computación voluntaria y desktop cloud, mientras que para Alwabel, et al. [9], entre los principales desafíos de investigación en sistemas desktop cloud están la confiabilidad y el mantenimiento del estado de las aplicaciones en caso de falla. Por su parte, Bohórquez, et al. [26] sugiere que a los sistemas desktop grid y de computación voluntaria les hace falta contar con mecanismos de tolerancia a fallas; y Marosi [27] indica que la tolerancia a fallas va a ser un desafío clave en los sistemas de computación voluntaria. Adicionalmente, Desell, et al. [28] hace énfasis en que no hay garantía para la finalización exitosa en los trabajos que corren sobre los sistemas de CV.

En los DC, dada la condición oportunista en el acceso a los recursos computacionales, la presencia del usuario del computador de escritorio al mismo tiempo que se están ejecutando las MVs es un factor crítico para su funcionamiento. Los usuarios de los computadores o máquinas físicas pueden afectar su desempeño o ejecución por varias razones. Por ejemplo, un usuario puede apagar o reiniciar el computador o ejecutar aplicaciones exigentes en capacidad computacional. Como resultado, la interrupción en la ejecución de las MVs puede hacer que se retrase o se pierda total o parcialmente el trabajo realizado por los investigadores. Sin embargo, aunque está detectado que los usuarios de los computadores pueden afectar la ejecución de las MVs, no hay suficiente información acerca de las múltiples fallas que pueden aparecer en un DC, las cuales no ocurren en otras plataformas.

Uno de los objetivos de usar *UnaCloud* como caso de estudio particular de un sistema DC es encontrar características relevantes que se pueden aplicar

a los sistemas DC en general. A continuación presentamos los desafíos que resultan del análisis del modelo actual de *UnaCloud*.

Los usuarios de *UnaCloud* son responsables de crear imágenes o plantillas a partir de las cuales serán creadas sus MVs y subirlas al servidor de *UnaCloud*. Por lo general, cada MV utiliza su propio archivo de disco duro virtual en el que almacena su sistema operativo, las aplicaciones y los datos. Como consecuencia, los archivos correspondientes a las imágenes virtuales son muy voluminosos. Copiar las plantillas por la misma red a través de la cual los usuarios de los computadores de escritorio acceden a Internet es un desafío para la escalabilidad en el aprovisionamiento de las MVs en el sistema. La transferencia de archivos grandes genera congestión en la red, produce errores en la transmisión y retrasa el funcionamiento del sistema.

Por otra parte, el tamaño que ocupan las imágenes virtuales en el disco de cada computador de escritorio es significativo, lo que da lugar a otro desafío para los DC: el manejo eficiente del espacio en disco.

Finalmente, es requerido que las imágenes virtuales respeten ciertas condiciones para que puedan ser usadas en el despliegue de MVs en el DC. Por ejemplo, el sistema operativo seleccionado, la configuración del adaptador de red, las extensiones del hipervisor y la configuración de la cuenta de superusuario, entre otras. Dejar en manos de los usuarios de un DC la responsabilidad de crear las imágenes virtuales se convierte en un desafío para el funcionamiento del sistema porque si una *Imagen Virtual (IV)* no cumple con las condiciones, por una parte el usuario del DC pierde el tiempo empleado en su creación y afinamiento, junto con el tiempo empleado en subirla al servidor, mientras que por otra, se desperdician recursos en el sistema.

1.3. Objetivos

1.3.1. Objetivo general

Extender las capacidades de servicio que prestan los sistemas *desktop cloud* mediante la incorporación de mecanismos de confiabilidad y tolerancia a fallos de manera que sistemas distribuidos formados por aplicaciones con procesos que se comunican entre ellos puedan ejecutarse con alguna garantía de terminación.

1.3.2. Objetivos específicos

1. Entender y caracterizar las fallas que pueden ocurrir en el funcionamiento de un sistema *desktop cloud*.

2. Ofrecer alternativas de solución para enfrentar las fallas que ocurren en sistemas *desktop cloud*.

3. Implementar y evaluar un conjunto inicial de alternativas de solución que permitan mejorar la confiabilidad de los sistemas *desktop cloud*.

1.4. Planteamiento del problema

Un sistema *desktop cloud* es una plataforma oportunista que aprovecha recursos computacionales ociosos en computadores donde los usuarios realizan sesiones interactivas de trabajo para ejecutar máquinas virtuales que apoyan la realización de proyectos académicos y científicos. Estos sistemas pueden fallar debido a su volatilidad inherente y por la presencia del usuario. Cuando fallan los computadores, la ejecución de las máquinas virtuales se interrumpe y el trabajo realizado puede perderse parcial o completamente. Los usuarios de esta clase de sistemas esperan algún compromiso de la plataforma para ofrecer ciertas garantías sobre la finalización exitosa o, de lo contrario, contar con información precisa que les permita tomar decisiones para recuperar o reiniciar el trabajo. Es decir, una plataforma que vaya más allá del simple "*mejor esfuerzo*" para ejecutar de manera desatendida aplicaciones de tipo bolsa de tareas, o la ejecución exitosa de aplicaciones paralelas que utilizan librerías como MPI. Lamentablemente, el desarrollo de estos sistemas *desktop cloud* confiables requieren el conocimiento específico de fallos y estrategias de mitigación que no necesariamente aplican en contextos similares como clústeres HPC y sistemas tradicionales de computación en la nube.

1.5. Metodología

La metodología empleada en el desarrollo de esta tesis de doctorado consiste en las siguientes cuatro etapas:

1. Realizar un análisis de fallas en sistemas desktop cloud en general, con énfasis en *UnaCloud*, para identificar las fallas que ocurren, establecer su origen (en términos de anomalías en el acondicionamiento e interrupciones en la operación), y sus consecuencias.

2. Determinar un conjunto inicial de estrategias que permitan mitigar los efectos de las anomalías y las interrupciones identificadas.

3. Implementar las estrategias de mitigación más relevantes y evaluar su comportamiento.

1.6. Contribuciones

1. **Análisis de fallas en sistemas desktop cloud.** Hemos presentado un contexto general acerca de las fallas que se presentan en sistemas desktop cloud. Analizamos algunas implementaciones de sistemas DC y similares para determinar cómo son las fallas que se presentan y su forma de enfrentarlas, incluyendo el funcionamiento del modelo actual de *UnaCloud*, nuestro caso de estudio. Identificamos y caracterizamos las fallas en el servicio y su origen con el fin de determinar algunas estrategias para enfrentarlas. Utilizamos la cadena de amenazas disponible en la literatura:

anomalías e interrupciones, errores y fallas. En una primera aproximación analizamos anomalías en la fase de configuración e interrupciones en la fase de operación de *UnaCloud*, los errores que estas anomalías e interrupciones pueden causar y las fallas en el servicio derivadas de estos errores. Posteriormente, extendimos esta cadena de amenazas con el fin de analizar los usuarios que generan las anomalías y las interrupciones y las consecuencias que pueden tener las fallas en el servicio. Finalizamos nuestro análisis determinando algunas estrategias que permiten mitigar sus efectos. Hemos encontrado las siguientes dos fallas. En la fase de configuración: El desktop cloud no puede aprovisionar la máquina virtual. En la fase de operación: La ejecución de aplicación es interrumpida. Como resultado del análisis de fallas, de la revisión de la literatura relacionada y de la experiencia de casi diez años desarrollando, utilizando, evaluando y analizando *UnaCloud*, sugerimos un conjunto de ocho estrategias de mitigación que pueden ser utilizadas para mitigar los efectos de las fallas identificadas y mejorar la confiabilidad, tanto en el aprovisionamiento como en la ejecución de máquinas virtuales. Para validar el trabajo realizado, implementamos las estrategias más relevantes en el contexto de *UnaCloud*. Para la fase de configuración, proponemos un nuevo modelo de aprovisionamiento de máquinas virtuales basado en las siguientes tres estrategias: (1) Ofrecer un catálogo de imágenes virtuales preconfiguradas creadas por expertos para que estén a disposición de los usuarios de *UnaCloud* para crear sus máquinas virtuales a partir de ellas. (2) Utilizar discos virtuales en modo de escritura multiconexión (*multiattach*) para la gestión eficiente del espacio en disco en los computadores donde se alojan las máquinas virtuales. (3) Precargar las imágenes virtuales en las máquinas físicas para evitar la transferencia de archivos por la red. Por otra parte, analizamos, diseñamos y desarrollamos un prototipo funcional de una solución de *snapshot* global para un sistema distribuido que corre en las máquinas virtuales de un DC. Esta solución permite enfrentar las interrupciones en la ejecución de máquinas virtuales al guardar el estado de todo el sistema de manera consistente mediante la toma de *snapshots* globales. En caso de ser necesario, se puede volver a un estado anterior utilizando las mismas MFs u otras.

2. **Protocolo de *snapshot* global.** Hemos adaptado un protocolo de *snapshot* global para el diseño de nuestra solución. Implementamos un prototipo funcional de nuestro protocolo consistente de *snapshot* global como estrategia de mitigación para las fallas en el servicio derivadas de la interrupción en la ejecución de máquinas virtuales. El protocolo permite coordinar los hipervisores que corren en las máquinas físicas empleadas en la ejecución de las máquinas virtuales administradas por *UnaCloud*. Esta solución mantiene la semántica del sistema que está corriendo en las máquinas virtuales sin modificar las aplicaciones o el hipervisor. Este sistema soporta la ejecución de sistemas distribuidos formados por aplicaciones con procesos que se comunican entre ellos corriendo en las

máquinas virtuales de un desktop cloud como *UnaCloud*. Nosotros usamos colores en los nodos y filtrado de paquetes de red para garantizar la consistencia. Cuando el sistema se recupera del *snapshot* global, puede continuar su ejecución sin mensajes perdidos o duplicados. En el trabajo incluimos la prueba de correctitud del protocolo de *snapshot* global, basada en la aplicabilidad del algoritmo presentada por Kangarlou, et al. [29]. Nosotros analizamos por separado las aplicaciones que usan TCP y UDP como protocolos de la capa de transporte y considerando los conceptos fundamentales de transferencia de datos a través de redes TCP/IP. El prototipo desarrollado es completamente funcional en *UnaCloud* con el hipervisor *Oracle VirtualBox*. Sin embargo, es posible extenderlo a otros hipervisores con mínimas modificaciones en el código fuente de la aplicación.

3. **Migración de máquinas virtuales basada en *snapshots*.** Hemos diseñado e implementado un mecanismo de migración de MVs de una MF a otra a partir de un *snapshot* especificado. Esta solución incluye el desarrollo de una herramienta de análisis de los archivos que forman una MV cuando se han obtenido múltiples *snapshots* para determinar los archivos requeridos. La migración de MVs es esencial dado que permite dar continuidad a la ejecución de una MV en caso de interrupción.

4. **Herramientas de administración de máquinas virtuales.** El diseño e implementación de un conjunto de herramientas para automatizar la administración remota de MVs. Estas herramientas son bastante útiles para la administración efectiva de MVs, ya que la automatización de tareas ahorra tiempo significativo, reduce la probabilidad de errores y permite reproducir los ambientes necesarios en pruebas que se desean repetir.

5. **Nuevo modelo de aprovisionamiento de máquinas virtuales en entornos oportunistas.** Como respuesta a las fallas identificadas en el acondicionamiento de un sistema DC, hemos desarrollado una solución eficiente y escalable para el aprovisionamiento de MVs basada en tres estrategias: catálogo de imágenes virtuales preconfiguradas, uso de discos virtuales en modo de escritura multiconexión y precarga de las imágenes virtuales en las MFs.

1.7. Organización del documento

Este documento final de la tesis doctoral tiene seis capítulos, los cuales se describen a continuación. En el capítulo 1 encontramos la introducción, la cual incluye un contexto general del tema, la motivación, los objetivos, el problema, la metodología, las contribuciones y la organización del documento. En el capítulo 2, exponemos los conceptos generales de DC incluyendo la terminología principal y las tecnologías que dieron su origen. Además, presentamos los

fundamentos de fiabilidad en sistemas de cómputo y describimos *UnaCloud* la implementación de DC desarrollada en la Universidad de los Andes, nuestro caso de estudio. En el capítulo 3 hablamos del análisis de fallas en *desktop clouds*. Este análisis de fallas finaliza con una estrategia integral de mitigación de las principales anomalías e interrupciones que pueden afectar el servicio prestado por un DC. En el capítulo 4 presentamos la propuesta para el nuevo modelo de aprovisionamiento de máquinas virtuales. En el capítulo 5 describimos el análisis, diseño e implementación de una solución de *snapshot* global para guardar el estado de la ejecución de un sistema que corre en máquinas virtuales. Este capítulo incluye los conceptos generales, el trabajo desarrollado, la evaluación de rendimiento y una discusión acerca de la reanudación de la ejecución de las MVs en los mismos computadores donde se tomaron los *snapshots* o en otros, si es necesario. El documento termina con el capítulo 6 en el cual presentamos las conclusiones y el trabajo que ha quedado para que en el futuro se pueda dar continuidad a esta tesis de doctorado.

Capítulo 2

Conceptos generales

Esta sección presenta una descripción acerca de los conceptos que dan contexto al trabajo de investigación. En particular, vamos a hablar de: (1) sistemas de computación voluntaria y sistemas *desktop grid*, (2) computación en la nube y sistemas *desktop cloud*, (3) fiabilidad en sistemas de cómputo, y (4) *UnaCloud*, nuestro caso de estudio.

2.1. Sistemas de computación voluntaria y sistemas desktop grid

Los sistemas de computación voluntaria y los sistemas *desktop grid* están estrechamente relacionados, dado que son sistemas distribuidos que ofrecen una plataforma de gran escala que aprovechan equipos con bajo nivel de uso que están total o parcialmente ociosos para ejecutar proyectos académicos o científicos [11].

Los sistemas de computación voluntaria y los sistemas *desktop grid* son un tema atractivo sobre el cual muchos autores han realizado investigación desde 1996, cuando se desarrolló el que se considera el primer proyecto de computación voluntaria: el proyecto GIMPS – *Great Internet Mersenne Prime Search* [30]. GIMPS permite buscar números primos de Mersenne, los cuales tienen aplicación en la industria informática [11]. Por otra parte, desde comienzos de los años ochenta ha habido una preocupación por extender las capacidades de los computadores, aprovechando los recursos ociosos de otras máquinas [31], así como para maximizar la utilización de los equipos de cómputo con la menor interferencia posible a las actividades de los usuarios [32] [33].

Los autores que han investigado sobre los sistemas de computación voluntaria y los sistemas *desktop grid* han presentado distintas caracterizaciones de este tipo de sistemas. Para algunos, los participantes están ubicados en cualquier lugar del mundo [34] [28], mientras otros autores los diferencian de acuerdo a su ubicación [35] [14] [7]. Estos pueden estar distribuidos en Internet o hacer parte de una red local en una organización. Como resultado, han surgido nombres diferentes para sistemas inspirados en principios similares. Por ejemplo,

Anderson [34] hace referencia a computación sobre recursos públicos, mientras que Desell [28] utiliza el término *grids* de computación voluntaria.

Los sistemas de computación voluntaria y los sistemas *desktop grid* siguen el modelo de computación paralela maestro–trabajador (*master-worker*). Típicamente los *workers* comparten su capacidad de procesamiento, sus unidades de procesamiento gráfico y su espacio en disco duro [28] [36]. Por su parte, el *master* se encarga de dividir las tareas costosas en tareas pequeñas que puedan ser ejecutadas por los *workers* en forma paralela y de consolidar sus resultados.

Estos sistemas buscan maximizar el uso de los recursos siempre y cuando los usuarios interactivos no perciban un deterioro significativo en el rendimiento del computador [11] o vean comprometida su seguridad. Según Kondo [8], los sistemas *desktop grid* hacen parte de los sistemas de cómputo distribuido más importantes del mundo. Dada la enorme capacidad agregada de ciclos de CPU, los sistemas *desktop grid* pueden ser utilizados como infraestructura computacional de gran escala a bajo costo. Esto se convierte en una ventaja para el desarrollo de proyectos científicos que no cuentan con la capacidad financiera necesaria para su ejecución en infraestructuras dedicadas, como las que ofrecen los proveedores tradicionales en sus centros de datos.

Los computadores participantes de los sistemas de computación voluntaria y los sistemas *desktop grid* se caracterizan por su alta volatilidad, ya que pueden entrar y salir al sistema en cualquier momento, lo que los hace parcialmente disponibles.

En la taxonomía de los sistemas *desktop grid*, Choi, et al. [35] sugieren dos categorías: los *Desktop Grid Voluntarios (DGV)*, con acceso desde cualquier lugar del mundo a través de Internet, y los *Desktop Grid Empresariales (DGE)*, con acceso desde la red local de la organización. Esta clasificación coincide con la planteada por otros autores, tales como Sarmenta [7] quien utiliza otros nombres para sistemas similares. Otros, como McGilvary y Andrew [12] señalan que los sistemas de computación voluntaria en ocasiones son descritos como sistemas *desktop grid*.

Los DGV son conformados por computadores propiedad de usuarios individuales (anónimos) quienes se encargan de la operación y mantenimiento de sus equipos. Estos computadores son altamente heterogéneos, con especificaciones de hardware y software particulares, y están parcialmente disponibles para el DGV porque son usados principalmente para las actividades de sus usuarios. La presencia de cada computador en el sistema depende de la voluntad de su propietario, quien determina la cantidad de tiempo y los recursos que pueden llegar a poner a disposición del DGV en un momento dado. En cuanto a la seguridad, existe la posibilidad de que algunos participantes envíen de manera indebida resultados falsos al DGV. Los DGV, por lo general, ejecutan aplicaciones HPC, incluyendo BoT. Por las condiciones mencionadas, los computadores participantes en los DGV son altamente volátiles, propensos a fallas y no confiables.

Por su parte, los DGE son conformados por computadores propiedad de una organización, la cual se encarga de su mantenimiento. En este caso, la operación de los equipos es realizada por usuarios de la organización. Los computadores

pueden llegar a ser heterogéneos, pero en menor proporción que en los DGV. Al igual que en los sistemas DGV, los computadores participantes en un DGE ofrecen disponibilidad parcial porque son usados para las labores rutinarias del personal. El administrador del sistema o la persona encargada en la organización de tomar las decisiones en *Tecnología Informática (TI)*, determina la cantidad de recursos de cada computador que pueden ser puestos a disposición del DGE. Además, al igual que los DGV, los DGE ejecutan aplicaciones HPC, pero estas pueden ser privadas, y aplicaciones BoT o similares con alguna clase de comunicación entre ellas. Por las condiciones habituales en una organización, la volatilidad en los participantes de un DGE es media, y aunque estos computadores son propensos a fallas, lo son en menor medida que los participantes en un DGV. Lo mismo ocurre con la seguridad y la confiabilidad, donde los DGE pueden contar con usuarios maliciosos, pero en menor proporción que los DGV, y son no confiables, pero la confiabilidad es mayor en los DGE.

El *middleware* encargado de la administración de un sistema DGV es más complejo que el software correspondiente en los DGE, debido a que debe considerar la ubicación de los recursos, la probabilidad de fallas y los asuntos de seguridad, así como la cantidad potencial de computadores participantes.

Una comparación entre los sistemas DGV y DGE se puede apreciar en la Tabla 2.1 [35].

Tabla 2.1: Comparación entre DG Voluntario y DG Empresarial

	DG Voluntario	**DG Empresarial**
Volatilidad	Alta	Media
Conexión	Internet	LAN
Propiedad	Usuario individuales (anónimos)	Empresa
Operación	Usuario	Usuario empresarial
Mantenimiento	Usuario	Empresa
Heterogeneidad	Alta	Media
Dedicación	No dedicado	No dedicado
Decisión de presencia	Usuario	Administrador
Fallas en el servicio	Propenso a fallas	Propenso a fallas, menos que en DGV
Seguridad	Posibilidad de usuarios maliciosos	Posibilidad de usuarios maliciosos, menos que en DGV
Confiabilidad	No confiable	No confiable, más que en DGV
Tipo de aplicaciones	Independientes, de cómputo intensivo, *HPC*, proyectos de interés general	Independientes, de cómputo intensivo, *HPC*, proyectos de interés general, pero puede ser privado
Comunicación entre aplicaciones	*BoT*, Típicamente no se comunican	*BoT*, Típicamente no se comunican, pero puede ser posible
Escala	Global	Local, limitado a la dimensión de la red corporativa
Complejidad del *middleware*	Alta	Alta, menos que en DGV

2.2. Computación en la nube y sistemas desktop cloud

2.2.1. Computación en la nube

La computación en la nube o *cloud computing* es un modelo basado en virtualización en el cual los usuarios pueden acceder a recursos computacionales tales como procesamiento, almacenamiento, redes, aplicaciones y servicios a través de Internet. Los servicios que ofrece la computación en la nube se ejecutan sobre infraestructura dedicada, generalmente alojada en centros de datos especializados. Los usuarios seleccionan los recursos de acuerdo con sus necesidades (por demanda) y pagan con base en el consumo de los recursos. Los servicios son aprovisionados y liberados de una forma ágil y con facilidad de uso para promover el autoservicio, pese a que los detalles tecnológicos internos permanecen ocultos al usuario. Además, la computación en la nube permite a los proveedores ofrecer servicios de TI a múltiples usuarios sobre la misma infraestructura (*multitenancy*), y permite a los usuarios aumentar o disminuir la capacidad de recursos requerida, de nuevo, de acuerdo con sus necesidades (dando la sensación de elasticidad). Para hacer posible este modelo de servicio, los proveedores deben tener un mecanismo que les permita medir el consumo y habilitar el uso de un medio de pago, por ejemplo una tarjeta de crédito [37]. Usualmente, los proveedores de CN ofrecen Acuerdos de Nivel de Servicio — *Service Level Agreements (SLA)* incluyendo algunas medidas de confiabilidad.

La computación en la nube es ofrecida a sus usuarios en tres modelos de servicio [38]: Infraestructura como Servicio — *Infrastructure as a Service (IaaS)*, Plataforma como Servicio — *Platform as a Service (PaaS)* y Software como Servicio — *Software as a Service (SaaS)*. IaaS es la forma más básica de proporcionar recursos de computación en la nube. En un modelo IaaS, los usuarios pueden desplegar MVs funcionales, cada una con su propio sistema operativo y aplicaciones. Los usuarios pueden instalar, configurar y ejecutar aplicaciones en estas MVs de acuerdo con sus necesidades y preferencias, y controlan el sistema operativo, pero no conocen los detalles internos del proveedor de la infraestructura. PaaS es un servicio de computación en la nube que proporciona todas las herramientas de desarrollo de software requeridas, dentro de un contexto específico. Como resultado, los desarrolladores pueden desplegar aplicaciones sin conocer los detalles subyacentes de la infraestructura. PaaS les permite tener control sobre las aplicaciones desplegadas y algunas opciones de configuración. Por su parte, SaaS es un servicio de computación en la nube donde los usuarios pueden acceder a aplicaciones a través de Internet; en este caso, los usuarios no tienen acceso a ninguna herramienta de control de la infraestructura [37].

2.2.2. Sistemas desktop cloud

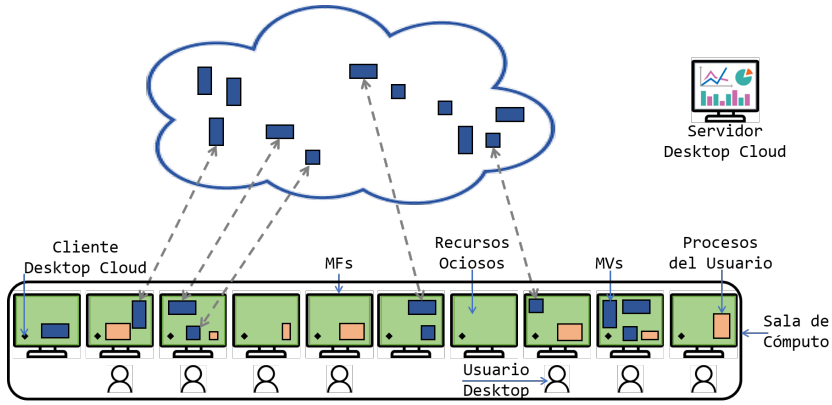
El concepto de *Desktop Cloud* surge de la combinación de la tecnología de virtualización con los sistemas de los sistemas de computación voluntaria y los sistemas *desktop grid*, de acuerdo con Alwabel, et al. [9], Cunsolo, et al.[13] y Fedak [14], entre otros. En forma similar a los sistemas de computación

voluntaria y los sistemas *desktop grid*, los DCs aprovechan la capacidad ociosa de los computadores participantes (típicamente a nivel empresarial) para la ejecución de MVs. Lo anterior con el fin de ofrecer capacidad de cómputo a investigadores para apoyar sus proyectos académicos y científicos, e incrementar el aprovechamiento de los recursos computacionales.

Los sistemas DC están pensados para ofrecer los servicios de computación en la nube, usando computadores de escritorio, los cuales se encuentran habitualmente en salas de cómputo de universidades y otras instituciones similares. Estos equipos son utilizados principalmente por estudiantes quienes realizan sus actividades de rutina, dejando ociosa una parte significativa de los recursos disponibles [4].

Un DC administra los recursos para ejecutar MVs funcionales con sus sistemas operativos y aplicaciones, sin interferir en las actividades que llevan a cabo los usuarios finales en los equipos de escritorio.

Figura 2.1: Visión general de un sistema *desktop cloud*



La Figura 2.1 presenta una visión general de un DC. Típicamente, un DC utiliza una arquitectura cliente-servidor en la que existe un programa servidor DC encargado de recibir las solicitudes de los usuarios del sistema (administrador y usuarios del DC) y un programa cliente DC en cada computador. El servidor DC tiene un manejador de MFs y un mecanismo de asignación de MVs a MFs. El cliente DC recibe las instrucciones del servidor DC y las envía al hipervisor para su ejecución sobre las MVs. Se puede observar que cada computador o MF tiene una capacidad ocupada por los procesos del usuario local y una capacidad ociosa que puede ser aprovechada en la ejecución de MVs, creando así una nube basada en computadores de escritorio (*desktop cloud*).

Es de anotar que distintos usuarios locales usan de manera diferente los computadores, por lo que la capacidad ociosa en cada equipo es distinta. Es decir, aunque los computadores de una misma sala de cómputo por lo regular son homogéneos, al tener usuarios locales en los computadores realizando cada uno sus propias actividades, la disponibilidad de recursos en cada MF es heterogénea y dinámica. Puede darse el caso de un computador completamente

disponible porque no hay un usuario local trabajando en ese computador, o el caso opuesto, un usuario local que ocupe tantos recursos de la MF que no haya espacio para alojar MVs. A pesar de esa heterogeneidad en la capacidad disponible de los computadores de las salas de cómputo, al consolidar los recursos no utilizados, la capacidad agregada puede llegar a ser significativa [4].

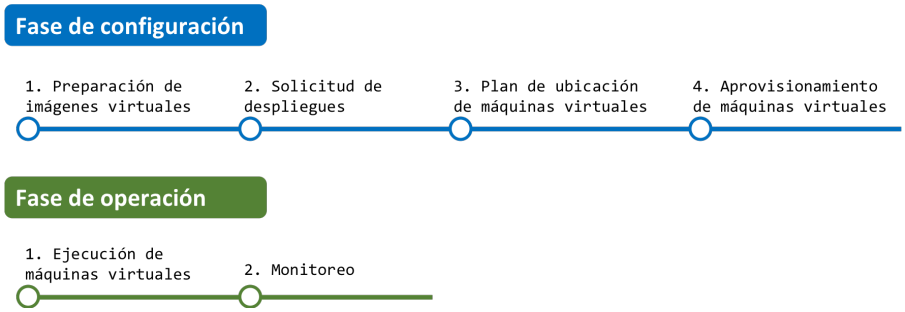
Un sistema DC comparte algunas características con un proveedor tradicional de computación en la nube. Ambos están basados en virtualización, y en ambos casos, los usuarios deben ser capaces de usar fácilmente una consola de administración para gestionar los servicios ofrecidos por el proveedor, típicamente a través de una interfaz web. Por otra parte, hay aspectos en que difieren. Los proveedores tradicionales de computación en la nube dependen de centros de datos especializados, mientras que los sistemas DC usan recursos no dedicados, con algún grado de heterogeneidad. Además, los proveedores tradicionales ofrecen a los usuarios una mayor variedad de herramientas y cuentan con soluciones más robustas, incluyendo facturación y pago. Por el contrario, los DC cuentan con una limitada variedad de herramientas de administración. Sin embargo, para ciertas necesidades pueden llevar a cabo satisfactoriamente el trabajo asignado con costos más bajos y en algunos casos completamente gratuitos [39]. Además, los usuarios obtienen el mejor servicio posible, dadas las limitaciones de estos sistemas.

Un DC puede proporcionar servicios esenciales de la computación en la nube, tales como autoservicio por demanda, acceso a través de la red, agrupación de recursos, rápida elasticidad y servicio medido [37]. Sin embargo, la oferta final de servicios depende de la implementación, ya que "DC es un concepto, no un producto" [40].

2.2.2.1. Funcionamiento de un sistema desktop cloud

La Figura 2.2 muestra las dos fases en las que hemos separado las actividades que se realizan durante el funcionamiento de un DC: (1) fase de configuración y, (2) fase de operación.

Figura 2.2: Funcionamiento de un sistema *desktop cloud*



Estas fases se describen a continuación.

Acondicionamiento La etapa de acondicionamiento agrupa cinco actividades:

1. **Preparación de las IVs.** Consiste en la creación de las IVs, incluyendo el sistema operativo y las aplicaciones debidamente configuradas y personalizadas por el usuario para su ejecución.

2. **Solicitud de Despliegue de una o más MVs a partir de una IV.** El usuario del DC solicita que sea ejecutada una cantidad específica de MVs a partir de cada IV.

3. **Planificación.** El sistema selecciona, mediante un algoritmo de ubicación, las MFs que van a ser utilizadas para la ejecución de las MVs solicitadas. En este proceso, puede darse el caso en el que múltiples MVs sean asignadas a una misma MF.

4. **Aprovisionamiento.** El DC copia las IVs en las MFs, crea las MVs y a continuación las configura, si es requerido.

Operación La etapa de operación agrupa dos actividades:

1. **Gestión de MVs.** El DC realiza las siguientes operaciones sobre las MVs: iniciar, modificar su configuración en tiempo de ejecución y, apagar.

2. **Monitoreo.** El DC obtiene estadísticas de uso de las MFs y las MVs. El DC controla la ejecución de las MVs.

2.3. Fiabilidad en sistemas de cómputo

En esta sección se presentan los conceptos básicos de la fiabilidad en sistemas de cómputo incluyendo sus atributos, las amenazas a la fiabilidad y los medios para alcanzarla.

2.3.1. Concepto de fiabilidad

Fiabilidad. Del inglés *dependability.* Este concepto surge en los años cincuenta cuando los computadores eran construidos con componentes con tendencia a fallar. Los investigadores querían determinar si los sistemas podían confiar en esos componentes [41].

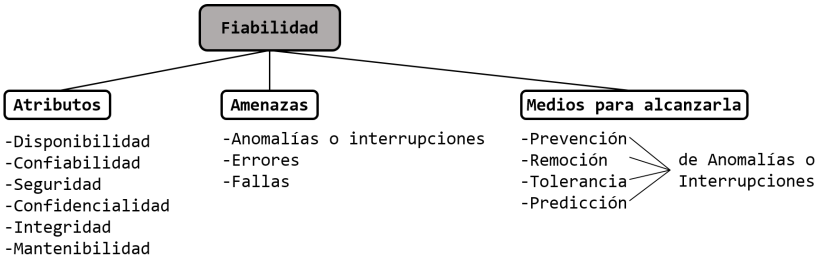
Avizienis [24] dice que, mientras algunos autores plantean la fiabilidad como un atributo deseable del sistema, otros autores han planteado definiciones matemáticas en torno a la probabilidad que un servicio funcione correctamente.

En una de las primeras definiciones, Laprie [42] definió la fiabilidad como "la propiedad de un sistema computacional de manera tal que se pueda depositar la confianza en el servicio que brinda". Por su parte, para Avizienis, et al. [43], "la fiabilidad de un sistema es la capacidad de un sistema de evitar fallas en el servicio que serían más frecuentes y severas que lo aceptable". Ellos dicen que

la fiabilidad abarca cinco atributos: disponibilidad, confiabilidad, seguridad, integridad y mantenibilidad. Teniendo en cuenta que los tipos de justificación varían según los enfoques del usuario, Prasad et al. [41] observaron la dificultad de medir la confiabilidad a partir de un conjunto de atributos preestablecidos y plantearon que la confiabilidad requiere conocer todo lo posible acerca de las anomalías e interrupciones que pueden ocurrir en un sistema y establecer mecanismos para su medición, evaluación, pronóstico y control.

La Figura 2.3, tomada de Avizienis et al. [43], muestra la taxonomía de la fiabilidad la cual relaciona: (1) *Atributos* usados para analizar la fiabilidad, (2) *Amenazas* que podrían impedirla, y (3) *Medios* para alcanzarla.

Figura 2.3: Taxonomía de la fiabilidad



Por otra parte, Avizienis menciona que existen *atributos secundarios* como robustez, resiliencia, usabilidad e interoperabilidad, que también se pueden usar para medir la fiabilidad de un sistema.

2.3.2. Atributos de la fiabilidad

Para estudiar la fiabilidad de un sistema de cómputo, es necesario definir los atributos a ser analizados. Basado en los atributos propuestos por diferentes autores [44][45][46][41], IEEE ha sugerido que la fiabilidad tiene los siguientes cinco atributos [43]:

Disponibilidad. Del inglés *availability*. Es la capacidad de un sistema de cómputo de proporcionar el servicio correcto cuando ocurre una solicitud. Es la probabilidad de que un sistema esté listo para entregar un servicio.

Confiabilidad. Del inglés *reliability*. Es la probabilidad de que un sistema ofrezca los servicios correctos dentro de un tiempo particular.

Seguridad. Del inglés *safety*. Es la ausencia de consecuencias catastróficas para los usuarios de un sistema de cómputo, sus datos y su entorno cuando se ofrece el servicio.

Integridad. Del inglés *integrity*. Es la capacidad de un sistema de cómputo de prevenir alteraciones inapropiadas.

Mantenibilidad. Del inglés *maintainability*. Es la capacidad de hacer modificaciones y reparaciones a un sistema de cómputo.

2.3.3. Amenazas contra la fiabilidad

Con base en lo que establece Avizienis et al. [43], las amenazas contra la fiabilidad de un sistema de cómputo son: anomalías o interrupciones, errores y fallas.

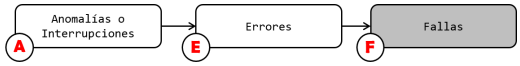
Anomalía/Interrupción. Del inglés *fault* (en computación, también conocido como *trap* o *exception*) [47] y se refiere a un tipo de interrupción a nivel de software. Es una condición anormal o defecto en un sistema de cómputo que conduce a un error. No siempre que se presenta una interrupción se produce un error, ni se produce como consecuencia inmediata debido a que el entorno puede activarla o no.

Error. Manifestación de una anomalía o interrupción en un sistema. Parte del estado del sistema que como consecuencia de una anomalía o interrupción, es responsable de conducir a una falla.

Falla. Del inglés *failure*. Evento en el sistema que ocurre cuando el servicio entregado es diferente al servicio especificado. Una falla también se produce cuando el servicio o el producto no cumple con las especificaciones o cuando produce resultados no esperados.

Estas amenazas se pueden ver como una cadena en la cual una anomalía o interrupción puede causar un error, y a su vez, un error puede provocar a una falla en el servicio, como se puede apreciar en la Figura 2.4, tomada de Avizienis et al. [43]. Al completarse la sucesión de estos eventos, un sistema de cómputo puede entregar servicios incorrectos o dejar de prestar el servicio.

Figura 2.4: Cadena de amenazas



Para ilustrar el concepto de la propagación de las amenazas contra la fiabilidad, considere el caso de un usuario que apaga un computador donde se está ejecutando una MV que hace parte del DC. En este caso, la interrupción en la ejecución del computador ocasiona un error de conexión al agente de software en la MF. Este error conduce a la falla en el servicio en la que el agente del DC en la MF no es accesible. Es importante resaltar que varias anomalías o interrupciones pueden causar el mismo error, y de la misma forma, varios errores pueden conducir a una misma falla.

Avizienis et al. [43] proponen una taxonomía para clasificar las anomalías o interrupciones. Incluyen diferentes criterios, tales como: *dimensión*: si están en el hardware o el software; *objetivo*: si son maliciosos o no; *persistencia*: si son permanentes o transitorios; *causa*: si son naturales o causadas por el hombre; *intención*: si son deliberados o no; y fase de su *ocurrencia*: si ocurren durante el desarrollo o en la operación.

2.3.4. Medios para alcanzar la fiabilidad

Muchos autores han propuesto varias formas de lograr la fiabilidad. Por ejemplo, Aviziens et al. [43] identifica cuatro tipos de técnicas: prevención, remoción, tolerancia y predicción de anomalías o interrupciones.

Prevención de anomalías o interrupciones. Son técnicas aplicadas durante el diseño y desarrollo del sistema para evitar la ocurrencia de anomalías o interrupciones. Estas técnicas incluyen una inspección minuciosa de la especificación, simulaciones de la ejecución del sistema y uso de herramientas de verificación.

Remoción de anomalías o interrupciones. Son técnicas orientadas a eliminar o reducir el número de anomalías o interrupciones que aparecen durante la operación del sistema. Estas técnicas involucran actividades de diagnóstico, modificaciones al sistema y pruebas en los cambios que se han realizado.

Tolerancia a anomalías o interrupciones. Son técnicas empleadas para asegurar que el sistema de cómputo continúa funcionando correctamente a pesar de la presencia de anomalías o interrupciones. Usualmente, esto puede ser hecho usando algún tipo de redundancia. Cuando una anomalía o interrupción ocurre, el sistema puede: (1) detectarla, (2) localizar el componente que la presenta, y (3) aislar el componente. Una vez el componente es identificado, una operación de recuperación es ejecutada en una de las siguientes formas: el sistema puede ser reconfigurado, deshabilitando el componente, o el sistema puede ejecutar una prestación de servicios con capacidades limitadas mientras vuelve a la normalidad.

Predicción de anomalías o interrupciones. Son técnicas que intentan predecir la ocurrencia de anomalías o interrupciones que ocurren durante la operación del sistema de cómputo. Normalmente, estas técnicas son usadas para evaluar el sistema o posibles daños en él, mediante evaluaciones cualitativas o cuantitativas.

2.4. *UnaCloud*

En esta sección vamos a presentar una descripción general de *UnaCloud*, siguiendo con el modelo actual del sistema. Posteriormente explicamos en detalle sus aspectos más importantes: los usuarios, los conceptos fundamentales, su arquitectura, el despliegue de MVs y la infraestructura disponible para la realización de las pruebas que permiten validar este trabajo de investigación.

2.4.1. Descripción general

UnaCloud es un DC que utiliza los recursos ociosos de los computadores de escritorio de las salas de cómputo de instituciones educativas para proporcionar IaaS, uno de los modelos de servicio de la computación en la nube [10].

UnaCloud es una plataforma actualmente usada en la Universidad de los Andes como infraestructura computacional, de bajo costo o totalmente gratuita, para soportar proyectos de investigación y tesis de doctorado en áreas tales como las ingenierías civil y química, procesamiento de imágenes, bioinformática y minería de datos [18].

La plataforma de *UnaCloud* está pensada para tener el menor impacto posible en las tareas desarrolladas por los usuarios de los computadores de escritorio [33]. Por lo tanto, esta plataforma ejecuta MVs con baja prioridad y en segundo plano, al mismo tiempo que las aplicaciones del usuario. *UnaCloud* es normalmente usada para ejecutar aplicaciones BoT. Sin embargo, recientemente ha sido usada para correr aplicaciones distribuidas que trabajan en otros esquemas, las cuales requieren constante comunicación.

2.4.2. Modelo actual del sistema

El modelo actual de *UnaCloud* está basado en una aplicación con arquitectura cliente-servidor, donde el servidor realiza la gestión de usuarios, la gestión de la infraestructura computacional, la asignación de MVs a MFs y la gestión de los despliegues de las MVs. Por su parte, el cliente o agente, es un programa que se ejecuta en los computadores de escritorio donde corren las MVs. Este programa cliente se encarga de interactuar con el servidor para controlar el hipervisor (actualmente *Oracle VirtualBox* versión 5) y a través de él, realizar las actividades en el funcionamiento de un DC, mencionadas en la Sección 2.2.2.1 de la página 16.

Los usuarios de *UnaCloud* son investigadores en distintas áreas del conocimiento. Ellos necesitan ejecutar sus aplicaciones y *UnaCloud* ofrece su capacidad computacional para que puedan hacerlo en MVs. Ellos realizan sus operaciones mediante una interfaz web, a través de la cual pueden subir una IV, crear y desplegar un clúster, obtener información acerca del despliegue, conocer las direcciones IP de las MVs que están en ejecución, detener una o varias MVs, entre otras.

Los usuarios de *UnaCloud* son los encargados de crear las IVs a partir de las cuales serán creadas las MVs en las respectivas MFs. Lo anterior debido a que, en un principio, *UnaCloud* fue creado con la idea de proporcionarle al usuario investigador un entorno que fuera lo más parecido posible a su ambiente de trabajo, y de esta manera, el sistema fuera fácil de usar. Luego de crear las IVs, los investigadores envían al servidor de *UnaCloud* los archivos correspondientes.

En *UnaCloud* el concepto de clúster es particular. Consiste en un conjunto de IVs. Entonces, un usuario de *UnaCloud* puede solicitar el despliegue de un clúster especificando la cantidad de MVs que desea ejecutar por cada IV. Por lo tanto, *UnaCloud* permite correr múltiples MVs basadas en una o varias IVs.

En el proceso de aprovisionamiento de las MVs, el servidor de *UnaCloud* realiza la transmisión de las IVs a las MFs para crear las MVs. *UnaCloud* utiliza discos virtuales de tamaño reservado dinámicamente y en modo de escritura normal, es decir, cada MV en *UnaCloud* usa su propio archivo de disco duro, el cual va creciendo a medida que se requiera más espacio, sin sobrepasar el

tamaño establecido inicialmente. En este archivo de disco virtual se almacena, de manera persistente, el sistema operativo, las aplicaciones y los datos. Este archivo está en el orden de los GB, razón por la cual la transferencia de archivos es un desafío para el funcionamiento de *UnaCloud*.

Si en la asignación de MVs a MFs, más de una MV va a ser ejecutada en el mismo computador, el agente de *UnaCloud* se encarga de crear las MVs a través de la clonación, es decir, obtiene una copia idéntica de la MV, modificando únicamente los identificadores que hacen que cada MV sea única. Como resultado de almacenar múltiples MVs en el mismo computador, se ocupa un espacio significativo en el disco de la correspondiente MF, en la cual se van a ejecutar las MVs. Por lo tanto, el manejo del espacio en disco es otro desafío para el funcionamiento de *UnaCloud*.

Con el fin de mejorar el desempeño en la transmisión de los archivos, cuando una IV es copiada en una MF, primero se guarda una versión original sin modificaciones. Posteriormente, a partir de esta IV se crea la MV solicitada. Esto quiere decir que antes de iniciar la transferencia de la IV a una MF, el agente verifica si ya la tiene almacenada o no. Aunque esta característica de *UnaCloud* utiliza la red de forma más eficiente, ocupa más espacio en el disco de cada MF.

Cuando las MVs están en ejecución, el usuario puede acceder a ellas a través de un protocolo de acceso remoto.

2.4.3. Usuarios

En *UnaCloud*, nosotros podemos identificar cuatro tipos de usuarios:

1. *Usuario de UnaCloud:* Es una persona que usa las capacidades computacionales ofrecidas por la plataforma. Un usuario de *UnaCloud* es capaz de crear y subir IVs al servidor, crear y desplegar clústeres de MVs, hacer seguimiento básico a las MVs en ejecución y conectarse a ellas a través de un protocolo de acceso remoto como *SSH*. Generalmente, se trata de investigadores sin mucho conocimiento técnico por lo que requieren alguna clase de asistencia.

2. *Administrador de UnaCloud:* Es la persona que administra la plataforma. Esta persona mantiene la información de los laboratorios, MFs, repositorios y despliegues. El administrador maneja usuarios y ejecuta tareas de configuración y monitoreo del sistema.

3. *Usuario del equipo de escritorio:* Normalmente es el estudiante, quien usa un computador que hace parte de *UnaCloud*. Aunque no es un usuario de la plataforma, sus aplicaciones tienen un prioridad mayor que la ejecución de las MVs. Las acciones llevadas a cabo por los usuarios en los computadores de escritorio podrían afectar el funcionamiento de *UnaCloud*.

4. *Administrador de la sala de cómputo:* Es la persona quien programa y ejecuta directa o indirectamente los mantenimientos preventivos y correctivos en los computadores de las salas de cómputo donde se ejecutan

las MVs administradas por *UnaCloud*. También es el encargado de otras tareas importantes como la instalación y configuración de software.

2.4.4. Conceptos fundamentales

La operación de *UnaCloud* está construida sobre cuatro conceptos fundamentales:

1. *Imagen virtual:* Una IV es un conjunto de archivos que forman una MV previamente configurada para su ejecución en una MF a través de un hipervisor. Corresponde a los archivos de configuración de una MV y los archivos con las imágenes de los discos. Por ejemplo, si se está usando el hipervisor *Oracle VirtualBox* [48], la IV corresponde a los archivos *.vbox* y *.vdi*. Se puede crear una MV completamente funcional a partir de una IV, siempre que esta cumpla con las condiciones requeridas para la ejecución, las cuales se enumeran en la Tabla 2.2 [18].

Tabla 2.2: Condiciones requeridas para ejecutar máquinas virtuales en *UnaCloud*

#	*Condición*
1	Tener un sistema operativo soportado por *UnaCloud*
2	Tener configurado su adaptador de red en modo puente
3	Tener configurada una dirección IP fija en el adaptador de red
4	Tener instaladas las extensiones del hipervisor en la versión compatible con *UnaCloud*
5	Tener correctamente configurada la cuenta de superusuario en el sistema operativo
6	Tener configurado un mecanismo de acceso remoto soportado por *UnaCloud*, por ejemplo *Secure SHell (SSH)**

* Aunque *UnaCloud* no requiere que las IVs tengan configurado un mecanismo de acceso remoto para desplegar una MV, es necesario para que el Usuario de *UnaCloud* pueda conectarse a sus MVs a través de la red.

Actualmente, *UnaCloud* soporta algunas versiones de las distribuciones de Linux *Debian* y *Ubuntu*. Dado que es habitual que las aplicaciones que corren en las MVs del DC puedan comunicarse con otras aplicaciones, es requerido que las MVs tengan configurado al menos un adaptador de red en modo puente y una dirección IP fija, la cual será reemplazada en el proceso de configuración de la MV. Actualmente *UnaCloud* utiliza rangos de direcciones IP tanto públicas como privadas.

Con respecto a las extensiones del hipervisor (por ejemplo, las *Guest Additions* de Oracle VirtualBox) son necesarias para que el Servidor de *UnaCloud* puede ejecutar comandos directamente sobre las MVs en el proceso de configuración. Es necesario que la versión de las extensiones

del hipervisor configuradas en la IV coincidan con la versión del hipervisor que está en ejecución en la MF.

Por otra parte, es importante anotar que distribuciones como Ubuntu no traen configurada la contraseña del superusuario. Por lo tanto, si el usuario de *UnaCloud* no hace esta configuración, el servidor de *UnaCloud* no puede ejecutar los comandos de configuración requeridos.

2. *Clúster:* Es un conjunto de IVs que un usuario de *UnaCloud* establece para ser usado en un despliegue (por ejemplo, si se ejecuta un sistema formado por un *master* y diez *workers*, el clúster es creado con dos IVs: la del *master* y la del *worker*). Estas imágenes son usadas como plantillas para crear MVs y deben ser preparadas cuidadosamente por los usuarios de *UnaCloud* para ser ejecutadas correctamente.

3. *Despliegue:* Es una solicitud hecha por un usuario de *UnaCloud* para ejecutar un clúster. En este caso, el usuario debe especificar cuántas instancias de cada IV van a ser ejecutadas y seleccionar el perfil de hardware para cada MV. El perfil de hardware en *UnaCloud* define las especificaciones de la MV en términos de núcleos de procesamiento y memoria RAM de acuerdo con la Tabla 2.3. Un despliegue es materializado en la ejecución del número solicitado de MVs en las MFs asignadas por el *UnaCloud*.

Tabla 2.3: Perfiles de hardware en *UnaCloud*

Nombre	*Núcleos de procesamiento*	*Memoria Ram [GB]*
Small	1	1
Medium	2	2
Large	3	4
Extralarge	4	8

4. *Algoritmos de ubicación:* Son los algoritmos empleados para seleccionar las MFs en las que van a ejecutarse las MVs. Cada usuario de *UnaCloud* tiene asociado un algoritmo de ubicación asignado por el administrador de *UnaCloud*. Estos algoritmos utilizan un identificador numérico para realizar la selección, y en los algoritmos más sofisticados, utilizan el perfil de hardware de la MV junto con la cantidad de núcleos de procesamiento y memoria RAM disponible en la MF. Los algoritmos disponibles se pueden ver en la tabla 2.4.

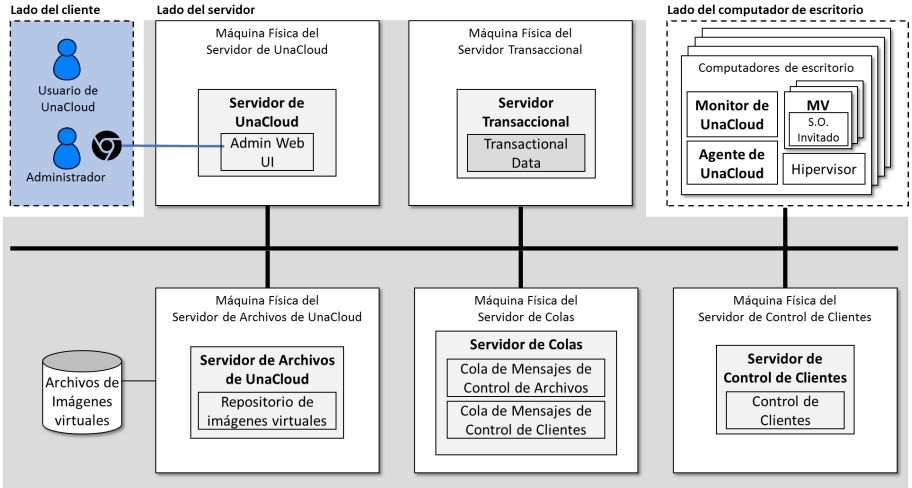
2.4.5. Arquitectura de *UnaCloud*

Una visión general de la arquitectura de *UnaCloud* se puede apreciar en la Figura 2.5. En esta arquitectura se pueden identificar los siguientes componentes [18]:

Tabla 2.4: Algoritmos de ubicación

Nombre	*Descripción*
Random	Selecciona una MF de manera aleatoria.
Round Robin	Asigna una MV por cada MF disponible. Si hay más MVs que MFs, al asignar la última MF, continúa desde la primera, hasta terminar con las MVs o con los recursos disponibles.
First Fit	Asigna las VMs que puedan ser ejecutadas en cada MF disponible de acuerdo con la disponibilidad de recursos de la MF y su perfil de hardware.
First Fit Decreasing	Es igual que el anterior, pero antes de iniciar ordena las MFs de acuerdo a la disponibilidad de recursos.
Best Fit	Igual que los dos anteriores. Ordena luego de cada asignación. Termina cuando haya ubicado todas las MVs.
Sorting	Es igual al algoritmo *Best Fit*. Selecciona MFs preferiblemente si hay algún usuario trabajando.
Singleton	Selecciona una sola MV por cada MF.

Figura 2.5: Arquitectura de *UnaCloud*



Lado del cliente:

1. *Navegador de Internet:* A través de cualquier navegador de Internet, los usuarios de *UnaCloud* pueden manejar sus propias IVs, clústeres y despliegues. Por su parte, un administrador de *UnaCloud*, además de lo ante-

rior, puede manejar usuarios y grupos, la infraestructura computacional y la configuración general del sistema.

Lado del servidor:

1. *Servidor de UnaCloud:* Es el componente que recibe las solicitudes del usuario de *UnaCloud* y del administrador de *UnaCloud* a través de la interfaz Web, aplica las reglas de negocio a las solicitudes, y determina la ubicación de las MVs para ejecutar los despliegues.

2. *Servidor Transaccional:* Es el componente que gestiona el modelo de datos de la aplicación y almacena la información de los elementos que son parte del sistema. También mantiene la información del estado de las MVs que son ejecutadas en las MFs.

3. *Servidor de archivos:* Es el componente encargado de administrar las IVs, las almacena y recibe solicitudes para transferirlas.

4. *Servidor de colas:* Este servidor incluye dos colas: una cola de mensajes que almacena las solicitudes de los usuarios para ser procesadas por el servidor de control de clientes; y otra cola para almacenar los mensajes relacionados con la administración de archivos. Este servidor mejora el desempeño porque en el sistema hay un componente encargado de gestionar las solicitudes y otro encargado de gestionar las respuestas.

5. *Servidor de control de clientes:* Es un componente encargado de administrar el estado de los agentes que llevan a cabo las acciones necesarias para atender las solicitudes de los usuarios y aquellas requeridas por el comportamiento normal del sistema.

Lado del usuario del computador de escritorio:

1. *Agentes de UnaCloud*: Cada computador de escritorio que hace parte de la plataforma incluye un agente de *UnaCloud*. Este recibe comandos del servidor y los traduce para que sean entendidos por el hipervisor para crear, configurar, encender y apagar MVs, entre otras operaciones. Los agentes también pueden controlar la ejecución de las MVs en sus correspondientes computadores y reportar los cambios de estado al servidor de *UnaCloud*.

2. *Monitor de UnaCloud:* Es el componente encargado de obtener información acerca del consumo de recursos en cada computador. El sistema de monitorización actual de *UnaCloud* recopila el consumo de: CPU (total del procesador y por núcleo de procesamiento), memoria RAM, red y energía, porcentaje de uso del disco duro y si hay un usuario trabajando en el computador de escritorio.

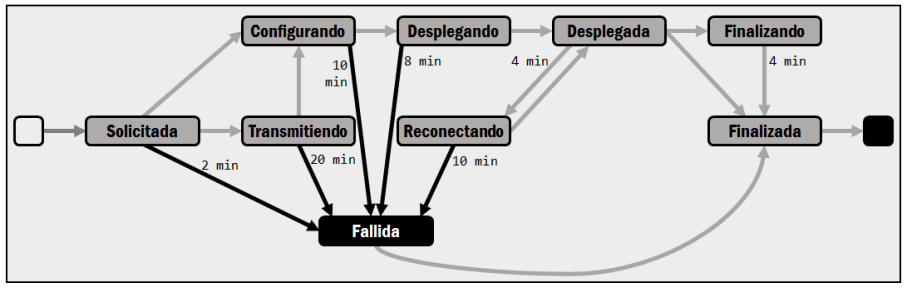
2.4.6. Despliegue de máquinas virtuales

UnaCloud incluye las etapas de acondicionamiento y operación mencionados en la sección 2.2.2.1 de la página 16. En la etapa de acondicionamiento, al usuario de *UnaCloud* le corresponde la preparación de las IVs y subirlas al servidor de *UnaCloud*, crear sus clústeres y solicitar los despliegues. La planificación y el aprovisionamiento son realizados por el DC, lo mismo que la etapa de operación.

Cuando el usuario de *UnaCloud* solicita que se despliegue un clúster, especifica la cantidad de MVs por cada IV que haga parte del clúster. Entonces, el servidor procesa la solicitud, planifica la ejecución del clúster asignando las MVs a las MFs en las que van a ser ejecutadas, transmite las IVs (si es necesario), configura las MVs e inicia su ejecución.

El despliegue y ejecución de cada MV es controlada a través de la máquina de estados que se puede apreciar en la Figura 2.6. La transición de un estado a otro se da por la realización de una acción o por la ocurrencia de un evento.

Figura 2.6: Máquina de estados de un despliegue en *UnaCloud*



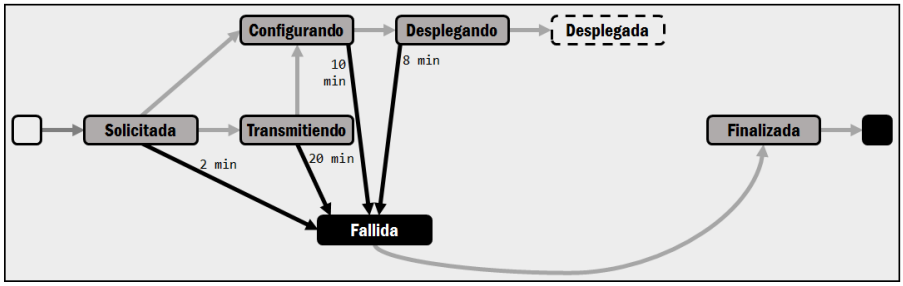
Después de la selección de las MFs, las actividades del despliegue se dividen en *aprovisionamiento* y *ejecución*.

Aprovisionamiento. La Figura 2.7 muestra los estados en los que se puede tener una MV antes de ser desplegada: SOLICITADA, TRANSMITIENDO, CONFIGURANDO y DESPLEGANDO. En condiciones normales, cuando termina el aprovisionamiento, termina la etapa de acondicionamiento y empieza la etapa de operación donde la MV pasa al estado DESPLEGADA y entra en ejecución.

En cada uno de los posibles estados en los que puede estar una MV durante el despliegue, esta debe realizar un conjunto de tareas dentro de un tiempo determinado. Si las tareas no son realizadas dentro de ese tiempo, se dispara un temporizador y, como consecuencia, la MV entra en un ESTADO DE FALLA. La MV entonces pasa a estado FINALIZADA y es apagada.

Como se puede apreciar en la Figura 2.7, el tiempo asignado antes de pasar a ESTADO DE FALLA depende de la actividad a ser ejecutada. Por ejemplo, mientras el tiempo asignado para el estado TRANSMITIENDO es de 20 minutos, el tiempo para el estado DESPLEGANDO es de 8 minutos.

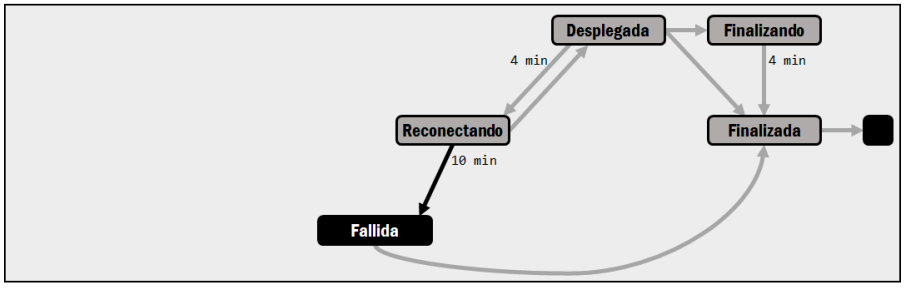
Figura 2.7: Despliegue en *UnaCloud* – Aprovisionamiento



En la fase de preparación, un despliegue puede caer en Estado de falla por diferentes razones. Por ejemplo, una imagen puede estar mal configurada, lo que impide que pueda iniciar su ejecución; la red puede tener problemas de operación, por lo que la imagen virtual no puede ser copiada en el computador donde será desplegada la MV; el servidor de *UnaCloud* no se puede comunicar con el agente en uno o más computadores; o el usuario del computador apaga el computador, lo reinicia, o ejecuta procesos altamente exigentes, por lo que la MV no se puede ejecutar como se esperaba debido a la falta de recursos.

Ejecución. En la etapa de operación, una MV está en ejecución y puede estar en los siguientes estados: Desplegada, Reconectando, Finalizando, y Finalizada (§ Figure 2.8).

Figura 2.8: Despliegue en *UnaCloud* – Ejecución



Durante la ejecución puede ocurrir una falla temporal. Las fallas temporales generalmente están asociadas con la pérdida de comunicación entre el servidor de *UnaCloud* y el agente que se ejecuta en un computador. Si la interrupción de la comunicación persiste durante cuatro minutos, la MV pasa al estado Reconectando. La MV puede permanecer hasta diez minutos en este estado, momento en el cual pasa a Estado de falla, luego a estado Finalizada y es apagada.

2.4.7. Infraestructura disponible

Actualmente, *UnaCloud* despliega las MVs solicitadas por los *Usuarios de UnaCloud* en las salas de cómputo *Waira* y *Alan Turing*, las cuales son administradas por el Departamento de Ingeniería de Sistemas y Computación de la Universidad de los Andes.

UnaCloud dispone de 117 computadores de escritorio, para ejecutar las MVs solicitadas por los usuarios. En cada computador, el *Administrador de la sala de cómputo* ha asignado un espacio en disco de 60 GB, para un total agregado de aproximadamente 7 TB. Este espacio es utilizado para almacenar MVs y datos de monitoreo.

Las especificaciones técnicas de los computadores de escritorio que hacen parte de estas dos salas de computadores se pueden ver en las tablas 2.5 y 2.6.

Tabla 2.5: Especificaciones de los computadores de la sala Waira

Característica	*Especificación*
Cantidad de MFs	78
Procesador	Intel (R) Core (TM) i7-7700 CPU @ 3.60Hz [8 core(s) x64]
Memoria RAM	32 GB RAM DDR4 @1.20 GHz.
Disco duro	1 TB.
Tarjeta de video	Intel HD Graphics 630
Tarjeta de red	Intel 1 Gb/s
Sistema operativo	Microsoft Windows 10 Enterprise

Tabla 2.6: Especificaciones de los computadores de la sala Alan Turing

Característica	*Especificación*
Cantidad de MFs	39
Procesador	Intel (R) Core (TM) i7-7700 CPU @4.20GHz [8 core(s) x64]
Memoria RAM	64 GB RAM DDR4 @1.20 GHz.
Disco duro	1 TB.
Tarjeta de video	Nvidia Quadro P600 Graphics
Tarjeta de red	Intel 1 Gb/s
Sistema operativo	Microsoft Windows 10 Enterprise

2.5. Resumen

En este capítulo hemos presentado los conceptos más relevantes para la investigación. Describimos los DC, incluyendo su relación con desarrollos anteriores, los sistemas de computación voluntaria y los sistemas *desktop grid*.

Examinamos en detalle el funcionamiento de los DC y las actividades que realiza tanto en la etapa de configuración como en la etapa de operación. En este capítulo mostramos los elementos básicos de la fiabilidad en sistemas de cómputo incluyendo sus atributos, las amenazas a la fiabilidad y los medios para alcanzarla. Finalizamos el capítulo con una amplia descripción de *UnaCloud* incluyendo el modelo actual del sistema, los tipos de usuarios de la plataforma, los conceptos fundamentales sobre los cuales está desarrollado, la arquitectura, el proceso de despliegue de MVs y la infraestructura disponible en la cual *UnaCloud* ejecuta las MVs de sus usuarios. Conocer la plataforma es el primer paso para hacer un análisis que permita identificar las fallas que se presentan en las dos etapas de su funcionamiento y de esta forma proponer soluciones que puedan ser extendidas a otros sistemas DC para mejorar su confiabilidad. El siguiente capítulo presenta un contexto general acerca de fallas que se presentan en los sistemas DC y similares, y un análisis de fallas del modelo actual de *UnaCloud*, nuestro caso de estudio.

Capítulo 3

Análisis de fallas en desktop clouds

Como lo habíamos mencionado en el Capítulo 1, los DCs se ven afectados por problemas diferentes a los que ocurren en otras plataformas, debido principalmente a dos razones: (1) la presencia del usuario de la MF donde se ejecutan las MVs de los usuarios del DC, y (2) el acceso oportunista a la infraestructura donde el usuario del computador es quien tiene la prioridad en el uso los recursos.

Sin embargo, hemos notado que los problemas de confiabilidad en los DCs empiezan cuando se incrementa el número de MVs que van a ser ejecutadas, las cuales primero deben ser aprovisionadas en las MFs donde van a ser alojadas.

Un DC puede experimentar fallas en el servicio en cualquier momento. No siempre se puede determinar el origen de una falla debido a que diferentes anomalías o interrupciones pueden ocasionar el mismo error y varios errores pueden conducir a la misma falla [49].

Por ejemplo, una falla en el servicio puede ocurrir cuando el servidor del DC ha seleccionado una MF para ejecutar una MV pero el aprovisionamiento no pudo ser finalizado correctamente. Esto puede ser causado, entre otras, porque: (1) el computador fue apagado o reiniciado; (2) el computador de destino no es alcanzable por la red; (3) hay congestión de red; (4) el computador seleccionado no tiene espacio en el disco duro para alojar la MV; (5) la IV no fue preparada correctamente para ser utilizada en el DC. Esto quiere decir que una misma falla en el servicio puede ser causada por anomalías o interrupciones ocasionadas por distintos actores, como en el caso de una MF que puede ser apagada por el usuario del computador escritorio o por el administrador de la sala de cómputo.

Es importante resaltar que una falla en el servicio puede tener consecuencias diferentes dependiendo de lo que haya sucedido al momento de presentarse y de las aplicaciones que están corriendo en las MVs. Para el usuario del DC no es lo mismo que se presenten fallas en la etapa de aprovisionamiento, es decir, antes de iniciar la ejecución de las MVs, o que las fallas se presenten cuando las MVs llevan un tiempo en la etapa de operación. Además, distintas aplicaciones pueden reaccionar de manera diferente a las fallas. Por ejemplo, en una aplicación de tipo BoT las consecuencias de una interrupción son mucho

menores que si la aplicación hace parte de un sistema distribuido donde hay comunicación entre diferentes procesos que corren en las MVs.

En este capítulo presentamos un contexto general acerca de fallas que se presentan en un DC, incluyendo algunas implementaciones de estos sistemas y su forma de enfrentar fallas, seguido de un análisis de fallas de *UnaCloud*, nuestro caso de estudio.

3.1. Implementaciones de desktop clouds y sistemas similares y su forma de enfrentar fallas

En los últimos años se han publicado trabajos relacionados con DCs, usando nombres diferentes para describir sistemas inspirados en los mismos principios o similares. Por ejemplo, *cloud voluntario* [13] [27] [50]; *nebulas* [51] [52]; *cloud no dedicado* [9]; *peer-to-peer/cloud* [53]; *ad hoc cloud* [12] [54]; y *cloud oportunista* [55] [10]. Estos sistemas tienen características comunes y diferencias que debemos tener en cuenta para entender mejor su comportamiento, así como la forma en que fue diseñado cada sistema para enfrentar fallas en el servicio.

En 2009, Cunsolo et al. [13] presentaron el proyecto *Cloud@home*. Este proyecto resulta de combinar CV y computación en la nube. Los autores señalan que el modelo *cloud* también puede ser aplicado en una escala más pequeña. El sistema incluye funciones de ejecución y almacenamiento. En el artículo se resalta la importancia de añadir redundancia de recursos al sistema, así como políticas de recuperación de las MFs para entregar un servicio confiable.

BOINC [34] es la plataforma de CV con mayor reconocimiento en el mundo. En este sistema, un número muy grande de usuarios, dispersos alrededor del mundo, ejecutan tareas para contribuir en el desarrollo de proyectos científicos [56]. Este sistema sufre los problemas de volatilidad y disponibilidad de la CV debido a que los usuarios en cualquier momento pueden empezar a utilizar los recursos que estaban ociosos y, como consecuencia, detener las tareas en ejecución [27]. Además, *BOINC* ofrece créditos y un escalafón por la participación de los usuarios como una forma de estimular y retener a los participantes. Sin embargo, deben manejar a los usuarios deshonestos que entregan resultados ficticios con el fin de reclamar créditos y subir posiciones en el escalafón. Este sistema utiliza la replicación de la misma tarea en múltiples MFs. De este modo, si una falla, las demás pueden reportar sus resultados. Adicionalmente, cuando un participante entrega datos falsos, puede ser detectado al comparar sus resultados con otros nodos.

Como *BOINC*, muchos otros sistemas se han desarrollado para aprovechar los recursos no utilizados por los usuarios. Basados en *BOINC*, existen proyectos que han extendido su funcionalidad a través del despliegue de MVs (usando principalmente *Oracle VirtualBox* como hipervisor), configuradas para ejecutar un cliente de este sistema en las MFs de usuarios voluntarios y en entornos empresariales [27] [57] [58] [19].

En el artículo presentado en 2013 por Marosi et al. [27], los autores introducen la virtualización de sistemas voluntarios con soporte para *BOINC* [34],

Condor [32] y *XtremWeb* [59]. Ellos proponen crear *GBAC* (*Generic BOINC Application Client*), como un primer paso hacia los sistemas de computación en la nube donde los recursos son proporcionados por nodos voluntarios a pesar de sus problemas inherentes de volatilidad y disponibilidad. Los autores mencionan las dificultades de proporcionar un servicio confiable sobre recursos no confiables y consideran la redundancia como un factor clave en otras soluciones propuestas. En su implementación, manejan la volatilidad de los voluntarios mediante el *checkpointing*, usando el mecanismo proporcionado por el hipervisor. De este modo, es posible suspender o detener el sistema y reanudarlo en la misma MF o en otra. Además, ellos proponen una solución para la transferencia de archivos entre los participantes usando un sistema de almacenamiento *peer-to-peer*.

Más recientemente, en 2018, Mengistu et al. [50] presentaron *cuCloud* un sistema de Computación Voluntaria como Servicio — *Volunteer Computing as a Service (VCaaS)*. Este es un sistema de nube voluntaria que integra MFs cuando están ociosas a la plataforma *CloudStack* IaaS. A diferencia de los anteriores, está implementado usando un sistema existente, lo que permite aprovechar todos los componentes ya desarrollados, adaptándolos a un entorno de computación voluntaria. A través de un sistema de monitoreo, *cuCloud* determina el estatus de cada MF con base en la información de disponibilidad de recursos (CPU, RAM y disco). Con el fin de ofrecer un servicio confiable, *cuCloud* predice la disponibilidad y confiabilidad futura con base en información histórica de los voluntarios usando las siguientes métricas de confiabilidad y disponibilidad: (1) tiempo hasta la siguiente falla, (2) tiempo promedio entre fallas y (3) tiempo para reparar. Esta información es utilizada en la asignación de MVs a MFs.

De igual manera, en 2013, Ryden, et al. [52] proponen el uso de *nebulas* de recursos voluntarios para crear una plataforma *cloud* sensible a la ubicación, enfocada en procesamiento y almacenamiento de datos. *Nebula* implementa tolerancia a anomalías e interrupciones mediante la replicación de datos y reejecución de tareas. En un trabajo anterior, publicado en 2009, Chandra, et al. [51], señalan que la confiabilidad de los nodos debe incorporarse en las decisiones de implementación del servicio, además de los criterios de rendimiento, como la velocidad computacional y el ancho de banda de la red.

El término *cloud no dedicado* es usado por Alwabel et al. en [9] (2014) para describir sistemas que proporcionan servicios de computación en la nube sobre infraestructuras no dedicadas, las cuales no fueron desarrolladas para este propósito. Los autores resaltan que la confiabilidad y mantenimiento del estado en caso de fallas son ocasionadas por la falta de administración central en los DCs. Además, para mejorar la confiabilidad de esta clase de sistemas, y debido a la alta volatilidad de los participantes, es necesario contar con mecanismos de recuperación de fallas.

Por otra parte, Amoretti et al. [53] (2016) proponen una combinación de computación en la nube y sistemas *peer-to-peer* para obtener las ventajas de la alta disponibilidad de la computación en la nube junto con el uso de recursos compartidos de bajo costo de los sistemas *peer-to-peer*. Esta integración de

sistemas *peer-to-peer* y computación en la nube incluye componentes y protocolos que se autoconfiguran para alcanzar confiabilidad, disponibilidad y rentabilidad. Para enfrentar la confiabilidad del servicio utilizan un mecanismo de autorregulación que se enfoca en la administración de réplicas en aplicaciones basadas en la nube, usando una red *overlay* sobre la red *peer-to-peer*.

McGilvary en 2014 [12], basado en el término computación en la nube *ad hoc* (*ad hoc cloud computing*), introducido por Kirby en 2010 [60], propone un *framework* en el que combina el uso de *BOINC* con el concepto de virtualización. El *framework* se fundamenta en el aprovechamiento de recursos de infraestructura no dedicados, esporádicamente disponibles y no confiables tales como en la CV. Según McGilvary et al. [54] (2015), el sistema implementado mantiene en servicio la operación a pesar de la constante entrada y salida de computadores participantes del sistema o la presencia de fallas. La asignación de una MV a un host es realizada con base en una medida de confiabilidad a partir de: (1) la cantidad de trabajos enviados, (2) la cantidad de trabajos completados, (3) el número de fallas en la MF (por ejemplo, MF apagada, falla de hardware o falla del sistema operativo), (4) el número de fallas en las MVs que corren en esa MF (por ejemplo, errores en la configuración de la MV, errores en la instanciación de la MV, errores en la ejecución de la MV o apagado de la MV), y la carga actual de recursos en la MF. En la ejecución utiliza *checkpointing* periódicos que son replicados, mediante sistemas *peer-to-peer*, en otras MFs participantes. Un componente se encarga de tomar periódicamente *snapshots* de las MVs individuales y de enviarlos a otros participantes en paralelo. En caso de falla, el servidor se encarga de reanudar la MV en el último *snapshot*. Debido a que las MVs no tienen aplicaciones que se comunican entre ellas, cada *checkpoint* guarda el estado completo de cada MV, lo que permite dar continuidad a su ejecución en otro computador.

CernVM [20] (2008) es proyecto de CV basado en *BOINC* en el cual se implementa un dispositivo virtual de software (*virtual software appliance*) utilizado para la realización de experimentos en el *LHC* (*Large Hadron Collider*), el acelerador de partículas en la Organización Europea para la Investigación Nuclear (*CERN*) [61]. En lugar de utilizar IVs con el sistema operativo y un conjunto de aplicaciones, las cuales deben ser transmitidas cada vez, en este sistema, todas las MVs se crean con base en la misma imagen de disco. El software para ejecutar se aprovisiona mediante el uso del sistema de archivos propietario llamado *CernVM-FS*, en el cual se almacena el software en forma de archivos de solo lectura. En lugar de instalar el software en cada MV, cada una accede de forma remota a los archivos binarios que necesita. Por lo tanto, esta solución reduce el espacio requerido en cada MF, pero aumenta el uso de la red y requiere de un equipo encargado de mantener y configurar el software con los archivos adecuados.

Finalmente, *UnaCloud* [10] es un proyecto que está en desarrollo desde 2011. Este sistema es nuestra implementación de un DC, el cual brinda soporte a las necesidades computacionales para el desarrollo de trabajos científicos y de investigación en un ambiente académico como el que se encuentra en una universidad. Como se mencionó en la sección 1.1, *UnaCloud* monitorea el

estado de la ejecución de las MVs. Sin embargo, este proceso no es usado para identificar las fallas que se presentan ni para tomar decisiones acerca de nuevos despliegues, por lo que se hace necesario analizar con más detalle las fallas que se presentan en *UnaCloud* para poder establecer correctivos y mecanismos de protección, y de este modo, ofrecer mejores garantías de éxito en la finalización de la ejecución de MVs para los usuarios.

La Tabla 3.1 recoge las estrategias empleadas para manejar las anomalías e interrupciones por los DCs o similares que mencionamos en esta sección.

Tabla 3.1: Estrategias de protección frente a fallas para ofrecer un servicio confiable en desktop clouds

Estrategia	*Sistema desktop cloud*
Incentivos a los participantes	*BOINC* [34]
Replicación/Redundancia	*cloud@home* [13], *GBAC* [27], *Nebulas* [51] [52], *peer-to-peer/cloud* [53]
Checkpointing/Restart	*GBAC* [27], *Ad hoc cloud computing* [12]
Reejecución de tareas	*Nebulas* [51] [52]
Predicción de la confiabilidad de un nodo	*cuCloud* [50], *Ad hoc cloud computing* [12]
No determinado	*Cloud* no dedicado [9], *CernVM*[55], *UnaCloud*[10]

A continuación presentamos una breve descripción de las distintas estrategias que aparecen en la Tabla 3.1:

Incentivos a los participantes. De acuerdo con Anderson [34], solo las personas interesadas en un proyecto voluntario donan sus recursos. *BOINC* desarrolló un sistema de créditos para usarlo como incentivos a los usuarios participantes. Un crédito es una medida que se calcula con base en los recursos donados, teniendo en cuenta la CPU, el disco y la red utilizados por el proyecto. El sistema lleva una cuenta por cada participante y publica la lista ordenada de la contribución de cada uno.

Replicación/Redundancia. Es el método más común. Consiste en duplicar las tareas, las MFs, las conexiones de red o los datos para utilizar el duplicado en caso de fallas [62] [63]. Dependiendo del tipo de aplicación que se ejecute en el DC, la replicación o redundancia puede ser la medida más efectiva, por ejemplo, cuando se trata de aplicaciones *BoT*. El principal inconveniente de esta técnica es el uso de recursos adicionales que se requiere para los componentes replicados.

Checkpointing/Restart. Consiste en guardar en archivos el estado de una MV en un momento específico creando un punto de verificación (*checkpoint*). Se puede recuperar la ejecución a partir del estado almacenado y tener rápidamente el sistema en operación, evitando iniciar de nuevo la tarea, ya que

conserva el trabajo realizado hasta el momento en que se toma el *checkpoint* (también llamado *snapshot*) [62].

Reejecución de tareas. Es el método más sencillo de todos. Consiste en enviar de nuevo la tarea para que sea ejecutada en la misma MF [62].

Predicción de la confiabilidad de un nodo. Es la aplicación de un modelo matemático para calcular un valor numérico a partir de datos recolectados por un sistema de monitoreo. Este valor es actualizado de manera permanente y utiliza tanto datos históricos como recientes. La confiabilidad de un nodo es utilizada en el proceso de asignación de MVs a MFs.

Al considerar las distintas estrategias que han empleado otros los sistemas DC y similares presentados en esta sección, encontramos que ofrecen soluciones para sistemas paralelos basados en aplicaciones BoT, las cuales no se comunican entre ellas. Sin embargo, estos mecanismos no solucionan otros sistemas tales como aquellos basados en MPI, susceptibles a fallar cuando uno de sus nodos no puede completar exitosamente las tareas.

En la siguiente sección presentamos una análisis de fallas en plataformas DC que hemos estado llevando a cabo y refinando durante el desarrollo de esta investigación. En este análisis tuvimos en cuenta no solo *UnaCloud*, nuestro caso de estudio, sino otras plataformas.

3.2. Análisis de fallas

Vamos a presentar el análisis de fallas para sistemas DC, haciendo énfasis en *UnaCloud*. Con cerca de 10 años en el desarrollo de este sistema, hemos acumulado una experiencia valiosa que nos permite identificar los problemas en su funcionamiento, tanto en la etapa de acondicionamiento como en la etapa de operación.

Para el análisis de fallas nos basamos en el siguiente conjunto de suposiciones, con el ánimo de considerar un caso general:

1. El DC tiene una arquitectura cliente-servidor con acceso a través de una interfaz web.

2. El usuario del DC conoce la plataforma y la puede utilizar correctamente a través de su interfaz web.

3. La implementación del DC no tiene defectos de desarrollo.

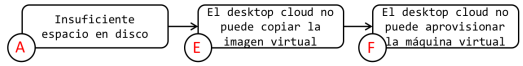
4. Los servicios que presta el DC en el lado servidor no se ven afectados en su ejecución por caídas en el servicio de su infraestructura.

5. El software que el usuario del DC ejecuta en las MVs no tiene defectos de desarrollo ni es sensible al tiempo.

6. Los hipervisores están instalados correctamente en las MFs que hacen parte del DC.

Para realizar nuestro análisis de fallas, inicialmente utilizamos la cadena de amenazas (§ Figura 2.4 de la página 19). Para ilustrarlo, considere el ejemplo de la Figura 3.1, en el cual se analiza una falla en el sistema debido a la imposibilidad de aprovisionar una MV en el computador seleccionado para ejecutarla. En este ejemplo, la falla se presenta porque el DC no puede copiar la IV en el computador seleccionado, lo que a su vez se debe a que hay insuficiente espacio en disco en el computador de destino.

Figura 3.1: Ejemplo de una cadena de amenazas



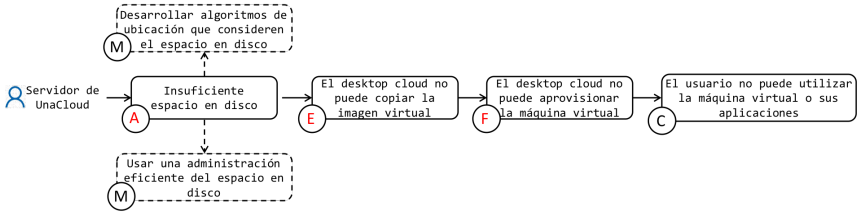
Aunque claramente se puede entender que hay una anomalía (insuficiente espacio en disco) que puede causar un error (el DC no puede copiar la IV) y este error puede provocar una falla concreta para el servicio (El DC no puede aprovisionar la MV), esta cadena de propagación de amenazas es bastante limitada en la identificación del problema y su correspondiente solución.

Esta anomalía tiene lugar cuando, en la planificación, el servidor del DC selecciona un computador específico para alojar la MV, pero el computador seleccionado no tiene espacio suficiente en el disco, ocasionando que la IV no pueda ser aprovisionada. En este caso, el resultado de no poder aprovisionar la MV en la MF es que el usuario no va a poder utilizar la MV, basada en esa IV, o sus aplicaciones.

Nosotros proponemos extender la cadena de amenazas para incluir al causante de las anomalías o interrupciones y sus estrategias de mitigación, además de las consecuencias de las fallas en el servicio. Al tener en cuenta estos nuevos elementos, vamos a poder distinguir mejor las amenazas contra la fiabilidad del sistema y, de esta forma, facilitar la automatización de correctivos.

La Figura 3.2 muestra el mismo ejemplo de la Figura 3.1, incluyendo además, al actor o usuario causante de la anomalía o interrupción, una o más estrategias para mitigar sus efectos, y las consecuencias de la falla.

Figura 3.2: Ejemplo de una cadena extendida de amenazas



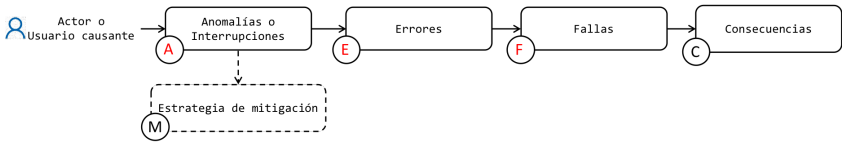
Se puede ver en la Figura 3.2 que la anomalía (Insuficiente espacio en disco) es causada por el servidor del DC, en particular, el componente encargado de seleccionar la MF donde va a ser ejecutada una MV. Si no hay espacio en disco, el DC no puede copiar la IV, y por lo tanto, el DC no puede aprovisionar una

MV con base en esa IV. Como consecuencia de esa falla, el usuario del DC no puede utilizar la MV o sus aplicaciones.

En este caso, la cadena extendida de amenazas sugiere dos estrategias para mitigar la anomalía: (1) desarrollar algoritmos de ubicación que consideren el espacio en disco antes de hacer la asignación de las MV a las correspondientes MFs. (2) usar una administración eficiente del espacio en disco.

Entonces, proponemos la cadena extendida de amenazas de la Figura 3.3, a través de la cual no solo se puede determinar el actor o usuario que causa la anomalía o interrupción, sino también las consecuencias de las fallas. Proponemos además, incluir algunas estrategias para enfrentar las anomalías o interrupciones que afectan la operación normal del sistema.

Figura 3.3: Cadena extendida de amenazas



Observe que estamos utilizando la notación de la Tabla 3.2 para identificar los distintos elementos que hacen parte de la cadena extendida de amenazas. Esta notación será empleada en el resto del documento.

Tabla 3.2: Notación para identificar los elementos de la cadena extendida de amenazas

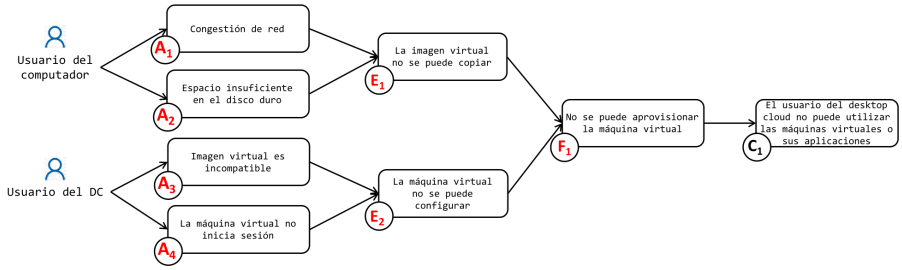
Elemento	*Notación*
Actor o usuario causante	
Anomalía	A
Error	E
Falla	F
Consecuencia	C
Estrategia de Mitigación	M

El análisis de fallas en el DC fue realizado utilizando la cadena extendida de amenazas ya mencionada. Este análisis incluye las fallas más relevantes que hemos identificado, una en la etapa de acondicionamiento: *el DC no puede aprovisionar la MV*; y otra en la etapa de operación: *la ejecución de la aplicación es interrumpida.*

3.2.1. El desktop cloud no puede aprovisionar la máquina virtual

El aprovisionamiento de una máquina virtual puede fallar debido a, por una parte, porque la imagen virtual no se puede copiar en el disco duro de la máquina física donde va a ser ejecutada; y por otra parte, porque la máquina virtual no se puede configurar. La Figura 3.4 muestra la cadena extendida de amenazas para la Falla F_1.

Figura 3.4: Análisis de la falla F_1: *El desktop cloud no puede aprovisionar la máquina virtual*



Se puede observar que el DC no puede aprovisionar la máquina virtual si no le es posible copiar la imagen virtual, error que a su vez, puede ser causado por las anomalías: A_1 *Congestión de red* por errores de comunicación (por ejemplo *timeouts*); A_2 *Espacio insuficiente en el disco duro* de la máquina física donde se va a ejecutar la máquina virtual, dado que los archivos generalmente ocupan varios GB; A_3 *La imagen virtual es incompatible* con el hipervisor instalado en la máquina física porque, por ejemplo, no hay coincidencia con la versión instalada del hipervisor; o A_6 *La máquina virtual no inicia sesión*, por ejemplo, porque la contraseña configurada en el sistema operativo de la máquina virtual es diferente a la suministrada por el usuario del DC.

En algunos DCs, tales como *UnaCloud* y *BOINC*, la configuración de una máquina virtual puede fallar si la imagen virtual no satisface algunos requerimientos. Por ejemplo, estos DCs usan tipos de red especiales y requieren que la imagen virtual incluya algunas configuraciones específicas. Si estos requerimientos no se cumplen, las máquinas virtuales no pueden ser configuradas exitosamente y no se pueden desplegar.

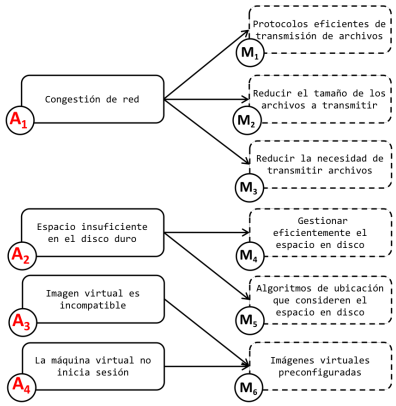
En el caso de *UnaCloud*, las imágenes virtuales deben cumplir con las condiciones especificadas en la Tabla 2.2 de la página 23, las cuales son indispensables para el aprovisionamiento exitoso y la correcta ejecución de las MVs.

Estas condiciones son difíciles de cumplir para usuarios no expertos, como es habitual entre los usuarios de un DC. Lamentablemente, *UnaCloud* no cuenta con mecanismos para validar la configuración de las IVs. Por lo tanto, una MV creada a partir de una IV que no cumpla estas condiciones no podrá ser configurada por el DC.

Por ejemplo, si el usuario de *UnaCloud* instaló un sistema operativo no soportado, o no instaló las extensiones del hipervisor en la versión adecuada, la IV que suba al servidor no será compatible con el hipervisor instalado en las MFs. En el caso de *BOINC* en MVs, es posible que el usuario voluntario cree su propia MV. Sin embargo, algunas de las instrucciones requieren conocimiento avanzado para poder realizar la configuración del sistema. Entre los requisitos podemos resaltar los siguientes: (1) descargar una imagen con el sistema operativo recomendado, (2) instalar el sistema operativo en la MV, (3) eliminar completamente los paquetes innecesarios para hacer más liviana la MV, incluyendo archivos instaladores, archivos de configuración y carpetas, (4) configurar el sistema de arranque de la MV, (5) configurar el inicio de sesión automático en el sistema, (6) deshabilitar el chequeo periódico que se realiza al sistema operativo cada que la MV se enciente 30 veces, (7) instalar y compactar una herramienta para limpiar el espacio libre en el disco, (8) compactar el disco de la MV, (9) configurar el puerto asignado para recibir las comunicaciones, y (10) configurar el acceso por escritorio remoto [58].

Como consecuencia del no aprovisionamiento de una máquina virtual, el usuario del DC no puede utilizar la máquina virtual ni las aplicaciones que corren en ella.

Figura 3.5: Análisis de la falla F_1: *El desktop cloud no puede aprovisionar la MV* — Anomalías y Estrategias de mitigación



La Figura 3.5 muestra las anomalías consideradas para la Falla F1, junto con sus estrategias de mitigación, las cuales incluyen:

M_1 *Usar protocolos eficientes de transmisión de archivos*, tales como los protocolos de archivos compartidos basados en *peer-to-peer* implementados en *Ad hoc Cloud Computing* [12] y *UnaCloud* [64].

M_2 *Usar mecanismos para reducir el tamaño de los archivos a transmitir*, tal como lo hace *CernVM* al manejar una única imagen virtual optimizada [20].

M_3 *Usar mecanismos para reducir la necesidad de transmitir archivos*, similar a la estrategia empleada por *CernVM* al almacenar imágenes frecuentes

en caché [20] acelerando la transmisión a través de la red.

Además de las estrategias de mitigación mencionadas, estamos proponiendo otras dos estrategias extraídas de algunos experimentos realizados en *UnaCloud*.

M_4 *Usar mecanismos para gestionar eficientemente el espacio en disco*, tales como el uso de discos clonados enlazados (*linked clone disks*) y discos virtuales en modo de escritura *multiattach* disponibles en hipervisores como VirtualBox [48], KVM [65], y VMware [66]. Estas técnicas pueden ser usadas para ejecutar múltiples MVs que comparten discos entre ellas y, por lo tanto, manejar eficientemente el espacio en disco en los computadores de escritorio.

M_5 *Usar algoritmos de ubicación que consideren el espacio en disco* que impidan al DC seleccionar MFs que no tengan suficiente espacio disponible.

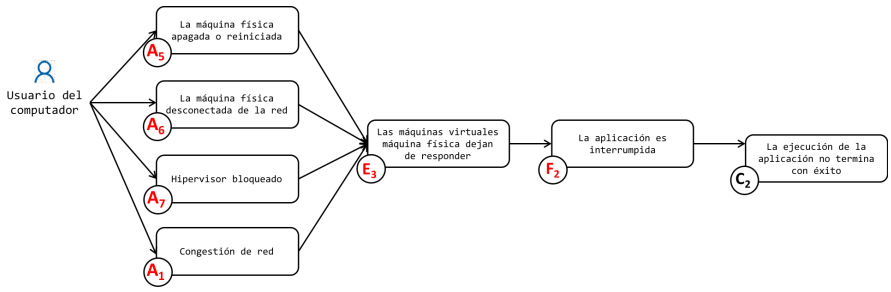
M_6 *Usar plantillas de IVs preconfiguradas*, ya probadas por integrantes del equipo responsable del sistema, en lugar de usar imágenes arbitrarias personalizadas por los usuarios del DC. Esta estrategia es usada por *CernVM* [20] y *cuCloud* [50], plataformas que ofrecen catálogos de imágenes que los usuarios pueden seleccionar para crear sus MVs.

En el capítulo 5 presentamos un nuevo modelos de aprovisionamiento en el que incluimos las estrategias de mitigación M_3 *Usar mecanismos para reducir la necesidad de transmitir archivos*, M_4 *Usar mecanismos para gestionar eficientemente el espacio en disco* y M_6 *Usar plantillas de IVs preconfiguradas*, las cuales al ser utilizadas en nuestro contexto, permiten que un DC sea más eficiente y efectivo en el aprovisionamiento de máquinas virtuales.

3.2.2. La ejecución de la aplicación es interrumpida

La ejecución de una aplicación puede ser interrumpida si las MVs que están corriendo en una MF dejan de responder, lo cual puede ser causado por anomalías en la MF. La Figura 3.6 muestra la cadena extendida de amenazas para la Falla F_2.

Figura 3.6: Análisis de la falla F_2: *La ejecución de la aplicación es interrumpida*



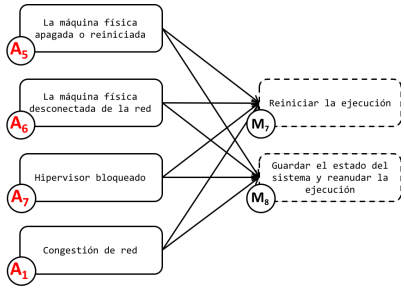
Se puede observar en la Figura 3.6 que si se presenta alguna de las siguientes anomalías: A_5 *La MF es apagada o reiniciada*; A_6 *La MF es desconectada de la red*; A_8 *Hipervisor bloqueado*; o A_3 *Congestión de red*, se desencadena una

interrupción en la ejecución de la aplicación, lo que trae como consecuencia que *la ejecución de la aplicación no termina con éxito.*

En la ejecución de las MVs las anomalías A_5 y A_6 interrumpen la ejecución de las MFs y por lo tanto, todas las aplicaciones que corren en ellas se interrumpen, incluyendo el hipervisor. La anomalía A_7 puede ser temporal (si el bloqueo del hipervisor es momentáneo) o requerir que sea reiniciada la aplicación o la MF donde se encuentra en ejecución el hipervisor. Esta anomalía pueda ocasionar que las MVs que corren en la MF donde estaba corriendo el hipervisor bloqueado, dejen de responder. Finalmente, la anomalía A_1 puede ocasionar una interrupción en la comunicación. Esta interrupción generalmente es temporal, pero dependiendo de la aplicación, esta interrupción puede causar que se alteren los resultados esperados o que se dañe la ejecución.

En caso de tener cualquiera de las cuatro anomalías mencionadas, la ejecución de la aplicación puede terminar sin éxito.

Figura 3.7: Análisis de la falla F_2: *La ejecución de la aplicación es interrumpida* — Anomalías y Estrategias de mitigación



La Figura 3.7 muestra las anomalías consideradas para la Falla F2, junto con sus estrategias de mitigación, las cuales incluyen:

M_7 *Reiniciar la ejecución.* En este caso, si la ejecución es interrumpida, se repite el trabajo y se pierde todo lo que se haya podido adelantar. Esta estrategia se utiliza si se puede asumir la pérdida del trabajo que se ha realizado hasta el momento de la interrupción, o si el esfuerzo por recuperar el estado del sistema no justifica hacerlo y es más conveniente repetir la ejecución desde el principio. En Nebulas [52], cuando un nodo de procesamiento falla en la ejecución de una tarea, la tarea será iniciada de nuevo en la misma MF o en otra.

M_8 *Guardar el estado del sistema y reanudar la ejecución.* Se puede suspender la ejecución de una MV cuando los recursos ociosos van a ser utilizados por el usuario de la MF, y reanudar su ejecución cuando el computador está ocioso de nuevo. McGilvary en su propuesta *Ad hoc Cloud Computing* [12] ejecuta *checkpoints* periódicos que son replicados en otras MFs a través de una solución de archivos compartidos basada en *peer-to-peer*, para que en caso de falla se pueda continuar con la ejecución de la MV en otra MF a partir del último *snapshot*. Por su parte, Marosi, et al. en *GBAC* [27] han utilizado *checkpoints* o *snapshots* para manejar la volatilidad de las MFs.

Utilizar *snapshots* o *checkpoints* para reanudar la ejecución a partir de un estado saludable de una MV, bien sea en la misma MF o en otra, es un asunto trivial, dado que los hipervisores proporcionan la funcionalidad para obtener *snapshots* para MVs individuales. Lo que no es trivial es obtener un *snapshot* global de un sistema distribuido formado por aplicaciones con procesos que se comunican entre ellos debido a que no es posible garantizar que los *snapshots* van a ser tomados al mismo tiempo, ni que los protocolos existentes para controlar la confiabilidad permiten el restablecimiento del sistema sin mensajes perdidos o duplicados.

En el capítulo 5 presentamos nuestra solución de *snapshot* global a través de la cual implementamos la estrategia de mitigación M_8 *Guardar el estado del sistema y reanudar la ejecución*, para hacer frente a las fallas ocasionadas por la interrupción en la ejecución de una o más máquinas virtuales corriendo en un desktop cloud.

Es importante resaltar que el análisis de fallas debe realizarse periódicamente, ya que nuevas amenazas pueden surgir de acuerdo con la evolución de la plataforma, las aplicaciones que se van ejecutando y el comportamiento que tienen los usuarios. Además, tal como se mencionó en la Sección 1.5 de la página 6), el análisis de fallas es un proceso iterativo.

3.3. Discusión

En este capítulo presentamos una caracterización de las principales fallas en el servicio que ocurren en los sistemas *desktop cloud* utilizando *UnaCloud* como un caso de estudio. Esta caracterización describe las posibles causas de las fallas (anomalías, interrupciones y errores) y sus consecuencias. Establecemos un conjunto inicial de estrategias de mitigación para contrarrestar los efectos de las anomalías e interrupciones para mejorar la confiabilidad de los servicios ofrecidos al usuario del *desktop cloud*. Identificamos una falla en la etapa de configuración: *El desktop cloud no puede aprovisionar la máquina virtual* y una falla en la etapa de operación: *La ejecución de la aplicación es interrumpida*.

Si bien las anomalías en la fase de configuración pueden ser más frecuentes debido a las dificultades para el aprovisionamiento de las MVs derivadas de la forma ineficiente en que se utilizan el espacio en disco y la dependencia de la red para la ubicación de las IVs en los computadores donde van a ser ejecutadas las MVs, las interrupciones en la ejecución de las MVs son una preocupación mayor porque se puede perder el trabajo completado hasta ese momento.

Los capítulos 4 y 5 discuten un nuevo modelo de aprovisionamiento de máquinas virtuales y un sistema de *snapshot* global como solución a las fallas en ejecución, respectivamente.

Capítulo 4

Nuevo modelo de aprovisionamiento de máquinas virtuales

Como ya habíamos mencionado, en el aprovisionamiento de una máquina virtual, el desktop cloud copia las imágenes virtuales en las máquinas físicas donde van a ser ejecutadas, crea las máquinas virtuales a partir de esas imágenes y a continuación las configura, si es requerido.

En el Capítulo 3 presentamos un análisis de fallas en sistemas desktop cloud, haciendo énfasis en *UnaCloud*. Para esta etapa, identificamos la falla: *El desktop cloud no puede aprovisionar la máquina virtual*, la cual se presenta cuando no se puede copiar la imagen virtual o luego de copiar la imagen virtual y crear la máquina virtual, ésta no se puede configurar. Gracias a la aplicación de la cadena extendida de amenazas, pudimos identificar las limitaciones significativas de escalabilidad en el aprovisionamiento de máquinas virtuales en el modelo actual de aprovisionamiento de *UnaCloud*. Lo anterior debido a que los archivos de las imágenes virtuales son voluminosos y ocupan un espacio significativo en disco y que dadas las condiciones en el uso de la red, la transmisión de las imágenes virtuales es lenta y propensa a fallas.

De acuerdo con nuestro análisis, el aprovisionamiento de máquinas virtuales depende de la calidad de las imágenes virtuales, del espacio ocupado en disco y de la red, la cual es compartida por los usuarios de las máquinas físicas. Para mitigar los efectos que pueden ser ocasionados por las anomalías o interrupciones que pueden terminar en que el desktop cloud no aprovisione una máquina virtual, proponemos un nuevo modelo de aprovisionamiento de máquinas virtuales basado en una combinación de las siguientes tres estrategias, las cuales, aunque son conocidas en otros contextos, no han sido consideradas en desktop clouds o sistemas similares:

1. Usar plantillas de imágenes virtuales preconfiguradas.

2. Usar mecanismos para gestionar eficientemente el espacio en disco.

3. Usar mecanismos para reducir la necesidad de transmitir archivos.

En primer lugar, podemos utilizar un catálogo de imágenes virtuales preconfiguradas que hayan pasado pruebas de calidad para garantizar que están listas para ser utilizadas para la creación de máquinas virtuales para disminuir los errores derivados de la configuración errónea de las imágenes virtuales por parte de los usuarios del desktop cloud. En segundo lugar, configurar los discos virtuales de tal manera que múltiples máquinas virtuales puedan utilizar una misma plantilla preconfigurada, y de esta forma, administrar de manera eficiente el espacio en disco. En tercer lugar, realizar una copia anticipada (precarga) de los archivos de las imágenes virtuales en los computadores para evitar su transferencia por la red. La implementación de estas estrategias se describe en las siguientes tres secciones.

4.1. Ofrecer un catálogo de imágenes virtuales preconfiguradas

La mayoría de los proveedores de infraestructura como servicio permiten a sus usuarios crear y ejecutar instancias (máquinas virtuales) con el sistema operativo y algunas aplicaciones básicas previamente instaladas y configuradas.

Sin embargo, otros como *UnaCloud* en su modelo de servicio permiten al usuario del desktop cloud la posibilidad de ejecutar su entorno habitual. Como consecuencia, el usuario debe crear y configurar las imágenes virtuales, y enviar al servidor los archivos correspondientes. Posteriormente, el usuario puede usar esas imágenes en el despliegue de sus máquinas virtuales y, de esta manera, ejecutar sus aplicaciones.

Si bien es cierto que para el usuario puede ser más fácil ejecutar sus aplicaciones en un entorno muy similar o idéntico al que usa habitualmente, también es cierto que cumplir con las condiciones requeridas por el desktop cloud para el aprovisionamiento exitoso de máquinas virtuales es una tarea difícil para sus usuarios. Vale la pena resaltar que cada desktop cloud establece sus propias condiciones para el aprovisionamiento y despliegue de máquinas virtuales, como lo mencionamos en la sección 3.2.1 en la página 40.

Para minimizar los problemas relacionados con los errores de configuración en las máquinas virtuales, proponemos el uso de imágenes especialmente diseñadas para su ejecución en el desktop cloud específico, creadas por usuarios expertos en la plataforma. En lugar de confiar en los científicos para producir las imágenes de disco utilizadas para crear las máquinas virtuales, proponemos el diseño de un conjunto de imágenes preconfiguradas que incluyan el sistema operativo y un conjunto de aplicaciones en las máquinas virtuales.

Esta estrategia es similar a la utilizada por *CernVM*, donde un grupo de expertos crea y mantiene un conjunto de sistemas de archivos de solo lectura para facilitar el uso de las aplicaciones. Sin embargo, a diferencia de ellos, proponemos utilizar el mismo tipo de imágenes virtuales utilizados por los hipervisores en lugar de una nueva capa de software. Nuestra estrategia es

similar a los catálogos de dispositivos virtuales (*virtual appliances*) que ofrecen los proveedores de infraestructura como servicio, tales como Amazon EC2 [16] y Google Compute Engine [67]. A diferencia de estos catálogos, usamos imágenes con software de propósito general preconfigurado. A partir de estas imágenes, el usuario puede personalizarlas de acuerdo con sus preferencias y desplegarlas en la infraestructura del desktop cloud.

Por ejemplo, es posible instalar el sistema operativo Ubuntu Server versión 16.04 en una máquina virtual. A continuación seleccionar el conjunto básico de aplicaciones que se le va a ofrecer a cada usuario. Este conjunto de paquetes podría incluir, por ejemplo, un compilador de C/C++, el kit de desarrollo de Java, el compilador de Python, un servidor SSH, el servidor Apache (servidor HTTP) y el motor de base de datos MySQL. Cada uno de estos paquetes debe quedar debidamente instalado y con la configuración básica.

4.2. Establecer un mecanismo eficiente de gestión de espacio en disco

Teniendo en cuenta que es una práctica común que los usuarios tengan máquinas virtuales con contenido similar, las cuales tienen en común el sistema operativo y algunas aplicaciones, queríamos solucionar la siguiente pregunta de investigación:

¿Cómo utilizar un disco virtual común en múltiples máquinas virtuales en tiempo de ejecución?

Los hipervisores tipo 2, tales como Oracle VirtualBox, KVM y VMware Workstation han implementado algunos conceptos en esa dirección. Por ejemplo, los hipervisores pueden configurar los discos duros virtuales en diferentes modos de operación, tales como normal, inmutable, compartible y multiconexión); usar discos diferenciales; snapshots; y clones enlazados.

El modo de escritura seleccionado para cada disco determina su comportamiento después de las operaciones de escritura de una MV y determina el funcionamiento de los *snapshots*.

Para destacar, los discos en modo de escritura normal permiten la conexión de múltiples máquinas virtuales al mismo disco, pero solo una en ejecución. En esta clase de discos, las modificaciones al disco se guardan en el mismo archivo.

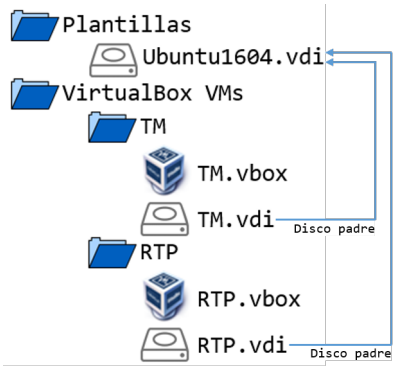
Es muy común que un desktop cloud utilice discos virtuales en el modo de escritura normal, en el que las máquinas virtuales pueden leer y escribir como lo hacen los computadores en discos reales. Este modo de escritura en los discos utilizados en las máquinas virtuales funciona muy bien cuando solo hay unas pocas almacenadas en el disco de un computador. Sin embargo, cuando varias máquinas virtuales están almacenadas en el mismo computador, cada máquina virtual tiene su disco en un archivo separado, el cual generalmente ocupa varios GB.

Por su parte, los discos en modo de escritura *multiattach* (multiconexión) también se pueden conectar a múltiples máquinas virtuales, las cuales pueden ser ejecutadas al mismo tiempo.

Para gestionar de manera eficiente el espacio en disco y minimizar los problemas relacionados con el consumo innecesario, proponemos el uso de discos en modo de escritura *multiattach*. Como característica especial, una vez que un disco es configurado como disco *multiattach*, el disco queda protegido y su contenido no se puede modificar. En nuestro caso, el contenido común entre varias máquinas virtuales (el sistema operativo y algunas aplicaciones) es almacenado solo una vez. Al utilizar las máquinas virtuales, es normal que algunos bloques en el disco común se modifiquen. Para mantener la consistencia de cada máquina virtual, los bloques modificados se almacenan en discos diferenciales que quedan relativamente pequeños.

Es de anotar que esta estrategia, que no hemos visto en otros DCs, ayuda a minimizar los problemas relacionados con el consumo de espacio en el disco de las máquinas físicas.

Figura 4.1: Ejemplo del uso de discos *multiattach*



La Figura 4.1 muestra un ejemplo del uso de discos *multiattach* en la creación de máquinas virtuales. Observe la carpeta `Plantillas`, la cual contiene el archivo `Ubuntu1604.vdi`. Este archivo corresponde a un disco *multiattach*. En la carpeta `VirtualBox VMs` estan almacenadas dos máquinas virtuales, una llamada `TM` y otra llamada `RTP`. En este ejemplo, tanto la carpeta correspondiente a la máquina virtual, como su archivo de configuración (con extensión `.vbox`) y el disco principal de la máquina virtual (con extensión `.vdi`), llevan el mismo nombre. Además, se puede observar que tanto el disco `TM.vdi` como el disco `RTP.vdi` son discos diferenciales que almacenan los cambios con respecto a su padre, el disco `Ubuntu1604.vdi`.

Para configurar un disco en modo *multiattach*, el disco debe estar en modo de escritura normal en una máquina virtual y tener almacenado su contenido. Por ejemplo, el sistema operativo y las aplicaciones. Posteriormente, se realiza la configuración del disco a modo de escritura *multiattach*.

Por lo general, los archivos de disco plantilla se guardan en una carpeta específica del disco, lo que quiere decir que cuando el disco se convierte en modo *multiattach*, se mueve a esa carpeta. En el hipervisor *Oracle VirtualBox*, el siguiente comando es utilizado para convertir un disco normal a disco *mul-*

tiattach. Como requisito, el disco debe estar registrado en el administrador de medios virtuales del hipervisor.

```
VBoxManage modifymedium disk <nombreDisco> --type multiattach
```

En este ejemplo, `<nombreDisco>` corresponde al nombre del archivo (generalmente con extensión `.vdi`) e incluye la ruta completa.

4.3. Establecer un mecanismo para reducir la transmisión de archivos

Como complemento a las dos actividades anteriores, proponemos copiar de manera anticipada (precargar) los archivos de las imágenes virtuales en las máquinas físicas donde van a ser aprovisionadas las máquinas virtuales.

Existen diferentes formas de copiar estos voluminosos archivos. Por ejemplo, manualmente o mediante un proceso automatizado, el cual se puede llevar a cabo en horarios de mantenimiento o de alistamiento de las salas de cómputo o laboratorios, de acuerdo con la planificación realizada por el administrador de las salas de cómputo. El objetivo es que las plantillas se copien en los computadores de las salas sin que este proceso afecte la red o el trabajo de los usuarios de las máquinas virtuales.

Estas imágenes virtuales corresponden a las plantillas preconfiguradas mencionadas en la sección 4.1 y almacenadas en discos que se han configurado en modo de escritura *multiattach*. Esto nos permite obtener una reducción aún mayor en el tiempo de de aprovisionamiento y los problemas ocasionados por el alto consumo de red debido a la transmisión de los archivos con las imágenes virtuales. Por consiguiente, el aprovisionamiento de máquinas virtuales será realizado sin necesidad de transmitir las imágenes virtuales lo que redunda en una mayor escalabilidad en los desktop clouds. Además, el uso de imágenes precargadas facilita de manera significativa la migración de máquinas virtuales de una máquina física a otra en la que se tenga precargada la imagen virtual tomada como base.

Es importante resaltar que al crear una máquina virtual a partir de un disco plantilla, esta tendrá el sistema operativo y los programas con la configuración establecida. Al usuario le corresponde la personalización de las máquinas virtuales creadas a partir del disco plantilla, de acuerdo con sus necesidades. Por ejemplo, modificar la contraseña del sistema, instalar un programa adicional y cargar archivos de datos. Las modificaciones al estado de disco a partir del momento en que fue creada la máquina virtual serán almacenadas en un archivo diferencial, el cual se convierte en el archivo principal de la máquina virtual, con el disco plantilla como disco padre.

Aprovechando esta estrategia, si tenemos las imágenes precargadas, al momento de desplegar una máquina virtual, solo se requiere la transmisión del archivo diferencial con el contenido personalizado por parte del usuario.

4.4. Validación

Nosotros hemos estado trabajando en mejorar la confiabilidad en el aprovisionamiento de máquinas virtuales en *UnaCloud*. Después de haber tenido problemas en la implementación de clústeres de más de 20 nodos [64], ahora hemos desplegado de manera consistente y sin fallas, hasta 448 máquinas virtuales, creando 7 máquinas virtuales en 64 máquinas físicas.

En la siguiente sección, presentamos nuestro análisis con respecto al consumo de espacio en disco y al tiempo de aprovisionamiento de hasta 200 máquinas virtuales. Hemos puesto atención en la realización de estos experimentos porque los computadores que actualmente se utilizan en las salas de cómputo de las universidades y empresas tienen la capacidad de ejecutar múltiples máquinas virtuales, por lo que la escalabilidad en el aprovisionamiento es crítico para aprovechar este tipo de plataformas. Por lo tanto, hemos realizado pruebas funcionales y de desempeño de la solución propuesta para el nuevo modelo de aprovisionamiento de máquinas virtuales para un desktop cloud, usando *UnaCloud* como plataforma y *Oracle VirtualBox* como hipervisor.

4.4.1. Aprovisionamiento de grupos grandes de máquinas virtuales

Para analizar el tiempo de aprovisionamiento de grupos de MVs y los errores que se presentan en el proceso, realizamos un experimento en *UnaCloud* con hasta 50 computadores de escritorio y aprovisionamos hasta 200 máquinas virtuales. Utilizamos una imagen virtual con el sistema operativo *Ubuntu Server 16.04*, la cual ocupaba 3.51 GB para implementar máquinas virtuales con 1 GB de RAM, 5 GB de disco duro virtual y 1 núcleo de procesamiento. Las máquinas virtuales se ejecutaron en MFs con un procesador *Intel Core i7-4770*, 20 GB de RAM y 500 GB de disco duro. Utilizamos una sala de cómputo con 78 computadores de escritorio, conectados mediante una red Ethernet de 1GB.

La tabla 4.1 muestra el tiempo promedio de aprovisionamiento. Dado que al usar las estrategias propuestas no es necesario transmitir archivos a las MFs, el tiempo de aprovisionamiento es el tiempo empleado en la creación de la máquina virtual y en la configuración necesaria para que estén listas para su ejecución. Es importante tener en cuenta que en nuestros experimentos creamos hasta cuatro máquinas virtuales en la misma máquina física y que el 100 % de las máquinas virtuales fueron aprovisionadas exitosamente, sin fallas durante el proceso.

La Tabla 4.1 incluye el tiempo máximo de aprovisionamiento en los experimentos realizados. En (a), vemos el tiempo de 1 a 50 máquinas virtuales usando 1 máquina física para alojar cada máquina virtual. En (b), (c) y (d), las proporciones son 2 MVs, 3 MVs y 4 MVs en cada máquina física, respectivamente. Usando una proporción de 1 MV/1MF, nosotros podemos notar que el tiempo de aprovisionamiento de 1 a 50 máquinas virtuales varía entre 0.98 y 1.10 segundos.

Tabla 4.1: Tiempo de aprovisionamiento de máquinas virtuales en *UnaCloud*

#MVs/MFs	time
1/1	0,98
2/2	1,04
5/5	1,02
10/10	1,09
20/20	1,10
50/50	1,10

(a)

#MVs/MFs	time
2/1	9,45
4/2	10,32
10/5	10,24
20/10	11,33
40/20	11,32
100/50	11,38

(b)

#MVs/MFs	time
3/1	18,55
6/2	19,90
15/5	20,89
30/10	20,40
60/20	21,27
150/50	21,75

(c)

#MVs/MFs	time
4/1	28,38
8/2	28,98
20/5	30,72
40/10	31,09
80/20	31,39
200/50	31,38

(d)

La tabla 4.1 (b) muestra que al cambiar la proporción de MVs/MFs, los tiempos aumentan. Lo anterior debido a que en el proceso de crear la primera MV, esta se conecta a un disco precargado en modo *multiattach*. Posteriormente se crean las siguientes MVs, una por una en la misma MF. Cuando la proporción es de 2 MVs en el mismo *host*, las MVs pueden ser aprovisionadas entre 9.45 y 11.38 segundos. Es de anotar que nosotros podemos aprovisionar 100 MVs en 50 MFs en solo 11.38 segundos, usando una conexión de red Ethernet de 1 Gbps compartida con los estudiantes mientras navegan por Internet y realizan sus actividades regulares.

La tabla 4.1 (c) presenta el tiempo de despliegue de 3 a 150 MVs en 1 a 50 MFs con una proporción de 3 MVs en cada *host*. Los tiempos obtenidos fueron desde 18.55 hasta 21.75 segundos. En este experimento, el tiempo de aprovisionamiento de 150 MVs fue solo de 21.75 segundos.

Finalmente, la Tabla 4.1 (d) reporta los tiempos de aprovisionamiento al usar una proporción de 4 MVs en cada MF, aprovisionando desde 4 hasta 200 MVs en tiempos que van desde 28.38 a 31.38 segundos.

4.4.2. Errores en el aprovisionamiento de MVs

En el pasado se han realizado varios intentos por desarrollar un sistema de monitoreo que aporte información oportuna y confiable. Lamentablemente, hemos tenido información acerca de fallas en los despliegues sin identificar el origen de estas fallas. Debido a la falta de esta información, no podemos

comparar la eficiencia de nuestras estrategias. Sin embargo, presentamos una discusión de las anomalías que ocurren en el aprovisionamiento, y la forma como las estrategias implementadas pueden prevenirlas. Las siguientes son las fallas que resultaron del análisis de fallas aplicando la cadena extendida de amenazas presentado en el capítulo 3.

Congestión y errores en la red. Las estrategias mencionadas eliminan la necesidad de transmitir IVs a las MFs. Los usuarios por lo general pueden iniciar MVs utilizando una plantilla precargada en los computadores. Estos despliegues no necesitan transferir ningún archivo.

Espacio insuficiente en disco. Considerando que las MVs usan discos en modo de escritura *mutiattach*, se reduce el espacio requerido en disco en las MFs. Por ejemplo, de acuerdo con los resultados obtenidos en nuestros experimentos, en lugar de requerir 3.51 GB para cada MV que se ejecute en una MF, el uso de discos *mutiattach* requiere 0.29 GB para cada MV adicional que se ejecuta en el mismo computador.

La imagen virtual no cumple con las especificaciones requeridas. Dado que proporcionamos una IV probada, lista para ser utilizada en la creación y ejecución de MVs, nos aseguramos de que cumple con todos los requisitos del sistema. En nuestras pruebas, hemos podido configurar e iniciar todas las MVs utilizando nuestra IV predefinida.

Hay dos fallas que no pueden evitarse con las estrategias propuestas si suceden en el momento en que se están copiando las IVs o configurando las MVs: (1) cuando las MFs son apagadas o reiniciadas y, (2) cuando las MFs se desconectan de la red. Estas fallas son inevitables dada la naturaleza no dedicada del hardware utilizado en los sistemas DC.

4.5. Discusión

Las estrategias propuestas son de fácil aplicación y los beneficios se pueden obtener al realizarlas en conjunto.

Cuando vamos a precargar un disco, asumimos que el disco corresponde a una imagen virtual previamente configurada y verificada. Aunque un disco en modo de escritura normal se puede precargar, y de este modo evitar el consumo de red que sería empleado en la transferencia de la imagen, crear máquinas virtuales adicionales a partir de una imagen con esta clase de discos se realiza mediante la clonación, lo que consume tiempo y espacio en el disco.

Nosotros sugerimos que la misma plataforma proporcione las imágenes virtuales en forma de catálogo con imágenes listas para ser utilizadas en el aprovisionamiento de MVs. Aunque es una tarea que parece ser simple, es necesario tener un equipo a cargo de crear las imágenes e implementar los cambios cuando sea necesario. Las modificaciones a la IV son un desafío debido al impacto

que las modificaciones pueden tener en las MVs que se han creado. Por lo tanto, se debe entender que en el futuro será necesario desarrollar un sistema de control de versiones para hacer frente a esta circunstancia.

Al usar discos en modo de escritura *multiattach*, precargados con IVs preconfiguradas, los cuales pueden ser conectados a múltiples MVs en tiempo de ejecución, no solo administramos el espacio de manera más eficiente, sino que también evitamos la transferencia de archivos voluminosos por la misma red a través de la cual los usuarios acceden a Internet.

Esto, por un lado, disminuye el tiempo de aprovisionamiento y, por otro, mejora significativamente el rendimiento de la red para los usuarios.

Además, dado que luego de crear una MV, esta se conecta al disco con el sistema operativo y las aplicaciones instaladas y configuradas, la MV queda rápidamente lista para su ejecución. Por lo tanto, la creación de una o más MVs en una misma MF es un proceso mucho más rápido, en comparación con el proceso equivalente a partir de la clonación de las MVs con las funciones proporcionadas por el hipervisor.

Por otra parte, al contar con IVs almacenadas en discos en modo de escritura *multiattach*, nuestras estrategias habilitan la posibilidad de migrar MVs en tiempo de ejecución. En este caso, es suficiente mover los archivos diferenciales a la MF en la que se ejecutará una MV y solo se requiere el registro de la MV en el hipervisor para poder ejecutarla.

Finalmente, para la implementación de estas estrategias utilizando el hipervisor *Oracle VirtualBox* se desarrollaron nuevas aplicaciones basadas en *VBoxManage*, la interfaz de línea de comandos. Las aplicaciones creadas nos permiten precargar un disco la MF y registrarlo en el hipervisor, crear una MV a partir de una imagen almacenada en un disco *multiattach* y crear una MV a partir de otra que está conectada a un disco *multiattach*, entre otras.

Capítulo 5

Snapshot global como solución a las fallas en ejecución

Estrategias para enfrentar interrupciones en la ejecución de MVs independientes incluyen *snapshots* (*checkpoints*), los cuales permiten guardar el estado de la ejecución de una MV y, en caso de ser necesario, reanudar la ejecución a partir de un estado correcto previamente almacenado. Esta solución es trivial porque los hipervisores proporcionan esta función. Sin embargo, no es trivial obtener un *snapshot* global consistente de un sistema distribuido formado por aplicaciones con procesos que se comunican entre ellos, ya que el concepto de reloj global no existe y no se puede garantizar que los *snapshots* de todas las MVs sean tomados al mismo tiempo.

En este capítulo presentamos un protocolo de *snapshot* global como estrategia de mitigación a las interrupciones que se presentan en la ejecución de MVs en un DC.

Esta solución difiere de otras propuestas que usan una red virtual para inspeccionar y manipular el tráfico que circula entre MVs haciendo difícil su aplicación en ambientes realistas. Nosotros obtenemos un *snapshot* global de un sistema distribuido corriendo en MVs que mantiene la semántica del sistema sin modificar las aplicaciones que corren en las MVs o los hipervisores.

5.1. Conceptos generales

5.1.1. *Checkpointing*

De acuerdo con el análisis de fallas presentado en el capítulo 3 (página 31), la principal estrategia para enfrentar las interrupciones en la ejecución de MVs consiste en guardar el estado de todo el sistema y reanudar su ejecución a partir de un estado consistente. Para reanudar la ejecución se pueden usar las mismas MFs u otras, si alguno de los computadores empleados en la ejecución no puede ser utilizado. La técnica empleada para guardar el estado de la ejecución de una MV es denominada *checkpointing* (*snapshots*) y es una función

que proporcionan los hipervisores.

Checkpointing es una técnica empleada para contrarrestar la pérdida de trabajo que representa la interrupción en la ejecución de un sistema. Esta técnica permite almacenar el estado de un sistema en un momento dado para continuar su ejecución a partir de ese estado sin tener que empezar de nuevo desde el principio, por ejemplo, cuando ocurre una falla. Naturalmente, si se reanuda la ejecución del sistema a partir de un *checkpoint*, ninguna modificación en el sistema después de que se haya tomado ese *checkpoint*, es incluida. Por lo tanto esas modificaciones se pierden.

Esta técnica consiste en dos componentes, uno proactivo y otro reactivo, llamados *checkpoint* y *restart* [68]. El primero guarda todo el estado del sistema. El segundo es el proceso de reanudar la ejecución a partir de un estado anterior.

Es posible tomar múltiples *checkpoints* completos o incrementales [69]. Cada vez que se toma un *checkpoint* completo, el estado de todo el sistema es guardado en archivos, mientras que el incremental solo guarda las diferencias con respecto al anterior, en lugar de guardar el estado completo. Por lo tanto, es necesario guardar todos los archivos de los *checkpoints* anteriores para poder reanudar la ejecución del sistema.

Hay cuatro técnicas para aplicar el concepto de *checkpointing* en aplicaciones distribuidas: a nivel de aplicación, a nivel de librería, a nivel del sistema y a nivel de virtualización [22] [70] [68].

A nivel de aplicación es necesario modificar el código fuente para desarrollar las funciones de *checkpointing*. Por el contrario, a nivel de librería, las aplicaciones no requieren modificaciones; ellas son enlazadas con una librería de *checkpointing*. Esta técnica es limitada a un ambiente específico [22]. A nivel de sistema, el *kernel* del sistema operativo es modificado por lo que la solución es mucho más específica a un ambiente particular, haciendo más difícil su portabilidad [22] [68].

Checkpointing a nivel de virtualización es realizado mediante la función de instantáneas (*snapshots*) proporcionada por los hipervisores y no es necesario saber cómo fue desarrollada la aplicación que corre en la MV. Esta técnica es más general que las anteriores, pero el desempeño es menor debido a la cantidad de espacio de almacenamiento requerido, el cual incluye el estado de la memoria (si la MV está encendida en el momento en que se toma el *snapshot*), el estado de los discos conectados a la MV, y los metadatos de la MV [71]. En este sentido, *checkpointing* de MVs individuales, como las que se utilizan en las aplicaciones masivamente paralelas, es trivial [68], ya que ejecutan aplicaciones independientes (que no se comunican entre ellas). Sin embargo, los hipervisores no han incluido mecanismos para guardar el estado de sistemas distribuidos que corren en múltiples MVs.

Algunos estudios acerca de mecanismos de confiabilidad basados en *checkpointing* fueron analizados para identificar técnicas, procedimientos y avances. McGilvary [54] usa *snapshots* de MVs, los cuales son replicados en múltiples nodos. McGilvary en su tesis doctoral *Ad hoc cloud computing* [12] usa un mecanismo de confiabilidad basado en *snapshots* de MVs en ejecución y replicación en múltiples nodos. El trabajo de McGilvary hace referencia a *checkpoints* de

MVs individuales, y no considera aplicaciones distribuidas que se comunican entre ellas. Por su parte, Marosi, et al. [27] utiliza el mecanismo proporcionado por el hipervisor para implementar checkpointing en *GBAC*.

En la siguiente sección se hablará con más detalle acerca del *checkpointing* a nivel de virtualización.

5.1.2. *Checkpointing* a nivel de virtualización

Los hipervisores como *Oracle VirtualBox*, *VMware Workstation* y *Microsoft HyperV* almacenan una representación de los *checkpoints* en una estructura jerárquica llamada árbol de *snapshots*. En estos árboles, hay un *snapshot* raíz con el estado de la MV al tomar el primer *snapshot*. Los otros *snapshots* se almacenan en archivos diferenciales con respecto al *snapshot* anterior (su *snapshot* padre). Cualquier *snapshot* puede ser el padre o la base de uno o más *snapshots* [21].

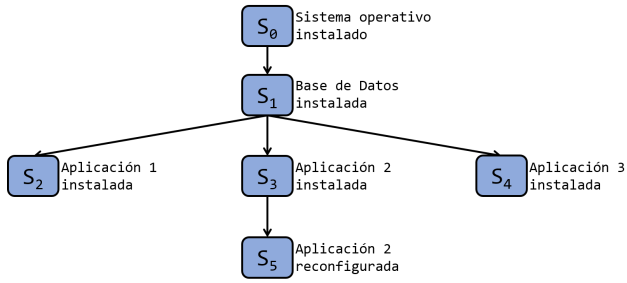


Figura 5.1: Ejemplo de un árbol de *snapshots*

La Figura 5.1 muestra un ejemplo de un árbol de *snapshots*. En esta figura, el *snapshot* S_0 fue tomado después de haber instalado el sistema operativo. El *snapshot* S_1 (basado en el *snapshot* S_0), tiene el estado de la MV después de haber instalado una base de datos. Los *snapshots* S_2, S_3 y S_4 tienen al *snapshot* S_1 como su padre y representan el estado justo después de que se instalaron algunas aplicaciones. Finalmente, el *snapshot* S_5, cuyo padre es el *snapshot* S_3, guarda el estado cuando una de las aplicaciones fue reconfigurada. Cuando se está ejecutando una máquina virtual, su estado actual debe basarse en cualquiera de los *snapshots* del árbol.

Cada hipervisor realiza un seguimiento de los *snapshots* y sus dependencias a su manera. Por ejemplo, *Oracle VirtualBox* almacena la información correspondiente a cada *snapshot* en diferentes secciones del archivo *.vbox*, mientras que *VMware WorkStation* lo hace en archivos separados [21].

Para entender la forma como un hipervisor maneja los *snapshots* y los archivos que se crean, nosotros elegimos el *Oracle VirtualBox* como caso de estudio y lo presentamos en la siguiente sección.

5.1.3. *Snapshots* en *Oracle VirtualBox*

Como se menciona en la Sección 5.1.1 (*Checkpointing*) en la página 55, el estado de una MV en un momento dado se representa como un conjunto de archivos.

En *Oracle VirtualBox*, se utilizan dos tipos de archivos para crear una MV: un archivo *.vbox* con la configuración de la MV y un archivo *.vdi* por cada disco duro virtual conectado a la MV. Cada vez que se toma un *snapshot*, se crea un archivo *.vdi* diferencial con el estado de cada disco. En estos archivos diferenciales, se almacenan los cambios con respecto al correspondiente disco padre. Además, en un archivo *.sav* se guarda el estado de la memoria. Lamentablemente, los nombres de estos archivos no aportan información acerca de su significado.

Como es de suponer, a medida que se obtienen *snapshots*, el número de archivos que mantiene el estado de la MV crece, y como consecuencia, la complejidad en su manejo también. *Oracle VirtualBox* no ofrece ninguna herramienta para facilitar el manejo de los archivos de disco virtual. Es importante resaltar que la información de los discos virtuales, incluyendo los *snapshots*, se puede obtener de diferentes maneras: (1) leyendo en el archivo de configuración de la MV, consultando en el *Administrador de medios virtuales* en la interfaz gráfica de usuario, o mediante el comando `VBoxManage list hdds`. En los tres casos es muy difícil hacer cualquier seguimiento manualmente a los nombres de los archivos de los discos en este hipervisor. La Figura 5.2 muestra un ejemplo de la salida del comando `VBoxManage list hdds`.

Figura 5.2: Información de un disco en *Oracle VirtualBox*

```
UUID:           82fd3f92-1174-4ac0-ab49-f9e09b8c6b06
Parent UUID:    e900ce1a-1546-4c4f-be8a-c7f8f7f8ba6f
State:          created
Type:           normal (differencing)
Location:       C:\Users\Profesor\VirtualBox VMs\VMa\Snapshots/{82fd3f92-1174-4ac0-ab49-f9e09b8c6b06}.vdi
Storage format: VDI
Capacity:       20480 MBytes
Encryption:     disabled
```

El conjunto de archivos que forman una MV en *Oracle VirtualBox* se define de acuerdo a la cantidad de discos conectados y a la cantidad de *snapshots* obtenidos, de acuerdo con la siguiente definición.

Estado inicial de una MV. Comprende el archivo *.vbox* y los archivos *.vdi* con los discos principales:

$$S_0 = \langle .vbox,\ .vdi_a,\ .vdi_b, \dots \rangle$$

Estado de una MV en un *snapshot* específico. Es el conjunto formado por el archivo *.vbox*, los archivos *.vdi* principales, los archivos *.vdi* ancestros y el correspondiente archivo *.sav* para ese *snapshot*:

$$S_i = \langle .vbox, .vdi_{a_i}, ancestros(.vdi_{a_i}), .vdi_{b_i}, ancestros(.vdi_{b_i}), \dots, .sav_i \rangle$$

5.1.4. Algoritmos para *Snapshots* globales

Un *snapshot* global es una copia del estado de un sistema distribuido. En sistemas DC, se refiere a una copia del estado de un conjunto de MVs corriendo en diferentes MFs y de las comunicaciones entre las aplicaciones que corren en las MVs. Tomar uno de estos *snapshots* globales no es trivial porque no se puede garantizar que todos los *snapshots* serán tomados al mismo tiempo. Además, algunos mensajes que hacen parte del estado del sistema distribuido podrían quedar en tránsito cuando es tomado el *snapshot* global. El *snapshot* global debe ser consistente para que sea posible que el sistema pueda recuperarse de una falla y continúe su ejecución, sin mensajes perdidos o duplicados.

Hay diferentes algoritmos y técnicas usadas para implementarlos. Por ejemplo, Chandy y Lamport [72] propusieron un algoritmo para determinar el estado global de sistemas distribuidos cuya comunicación estaba basada en canales FIFO. Posteriormente Mattern [73], basado en el trabajo de Chandy y Lamport, presentó su algoritmo para *snapshots* distribuidos entre aplicaciones con canales de comunicación no-FIFO. Más adelante, Kangarlou et al. [25] simplificó el algoritmo de Mattern y lo implementó usando MVs. Sin embargo, esta solución requiere *switches* virtuales personalizados que no son comunes en configuraciones de sistemas DC. En el marco de esta tesis, en [40] propusimos un protocolo de *snapshot* global para sistemas distribuidos corriendo en MVs sin switches virtuales que pueden ser aplicados a DCs.

De acuerdo con Chandy y Lamport [72], el estado global (EG) de un sistema distribuido formado por n procesos y los canales de comunicación entre cada par de procesos, incluye el estado local (EL) de cada proceso en el sistema junto con los estados de los canales de comunicación entre ellos (EC).

Estado global. Comprende el estado local de todos los procesos del sistema y el estado de todos los canales de comunicación:
$EG = \langle \bigcup_{i=0}^{n-1} EL_i, \bigcup_{i=0,j=0}^{n-1} EC_{i,j} \rangle$

En esta expresión se representa la relación del estado global de un sistema distribuido con los estados locales y los estados de los canales de comunicación, donde $0 \leq i, j < n$ representan el número de los procesos en el sistema, EL_i representa el estado del proceso i, y $EC_{i,j}$ representa el estado del canal unidireccional entre los procesos i y j.

Cada vez que se envía un mensaje entre dos procesos del sistema, cambia el estado interno de las aplicaciones involucradas. Por lo tanto, el estado de un proceso en un momento dado es la consecuencia de los eventos ejecutados por el proceso hasta ese momento. Si algo ocurre después, no pertenece a ese estado.

Se dice que un mensaje enviado por el proceso i al proceso j está en tránsito (pertenece al estado del canal entre los procesos i y j) si ha sido enviado por el proceso i pero aún no ha sido entregado en al proceso j, tal como lo representa

la Ecuación 5.1.

$$Mensaje\ en\ tránsito_{i,j} = \{m_{i,j} | enviado(m_{i,j}) \in EL_i \wedge recibido(m_{i,j}) \notin EL_j\} \tag{5.1}$$

En el mismo sentido, Kshemkalyani [74] plantea que un *snapshot* global es consistente si cumple las siguientes dos condiciones:

- Si un mensaje es enviado por un proceso, o el mensaje es recibido en el destino, o el mensaje está en el canal de comunicaciones (§ Ecuación 5.2).

$$enviado(m_{i,j}) \in EL_i \Rightarrow m_{i,j} \in EC_{i,j} \otimes recibido(m_{i,j}) \in EL_j \tag{5.2}$$

- Si un mensaje que ha sido enviado por un proceso no pertenece al estado local del transmisor, ni el mensaje debe hacer parte del estado local del receptor, ni el mensaje debe hacer parte del canal de comunicaciones (§ Ecuación 5.3).

$$enviado(m_{i,j}) \notin EL_i \Rightarrow m_{i,j} \notin EC_{i,j} \wedge recibido(m_{i,j}) \notin EL_j \tag{5.3}$$

5.1.5. *Snapshots* en ambientes de red virtualizados

Mattern [73] propuso un algoritmo de *snapshot* para determinar un estado global causalmente consistente de un sistema distribuido con canales *no FIFO (First-In, First-Out — Primero en llegar, primero en ser atendido)*, basado en la propuesta de Lai y Yang [75]. Ellos usan el concepto de colores (*coloring*) en cada mensaje para indicar si el mensaje fue enviado antes o después de tomar el *snapshot* local mediante una marca (un color) insertado por el transmisor dentro de los mensajes de capa de aplicación.

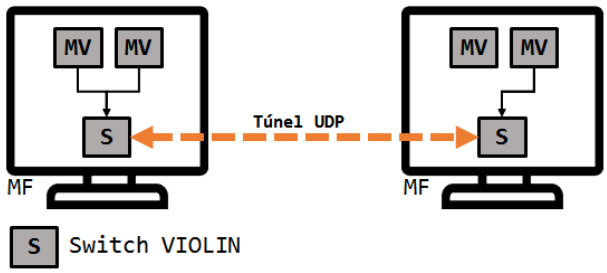
De acuerdo con Lai y Yang, cada proceso inicia su ejecución con el color blanco. Cuando un proceso toma su *snapshot* local, el proceso se convierte en rojo. Cada mensaje enviado por un proceso lleva el color del proceso que lo envía. Mattern introdujo un tercer color que permite al algoritmo ser ejecutado repetidamente dejando siempre un color disponible. Los colores son representados por los valores 0, 1 y 2 en forma circular.

Para ambientes de red virtualizados, el algoritmo de Mattern en [73] tuvo que ser adaptado. Kangarlou en su tesis doctoral *Improving the Reliability and Performance of Virtual Cloud Infrastructures* [25] presenta *VNsnap*, un sistema que toma *snapshots* (*checkpoints*) distribuidos en infraestructuras de red virtuales con mínima inactividad. El trabajo es inspirado en [73]. VNsnap implementa una simplificación del algoritmo de Mattern sin modificar las aplicaciones que corren en las MVs. El algoritmo toma un *snapshot* distribuido de un ambiente de red virtual completo llamado VIOLIN [76], el cual será descrito brevemente a continuación.

VIOLIN [76] es un ambiente de red virtual que proporciona servicios similares a aquellos proporcionados por una red física, en términos de direccionamiento IP, privilegios administrativos, librerías y servicios. VIOLIN consiste

en un conjunto de MVs conectadas mediante *switches* VIOLIN, uno en cada máquina física, tal como se muestra en la Figura 5.3, la cual ilustra el ambiente de red VIOLIN en operación. Los *switches* VIOLIN interceptan el tráfico entre las MVs y envía los *frames Ethernet* a la MV destino a través de túnel UDP, si la MV está corriendo en otra máquina física.

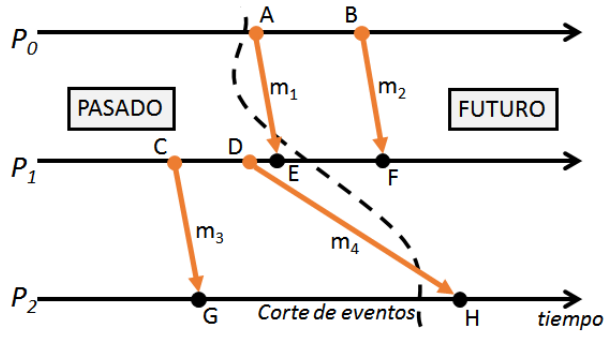
Figura 5.3: Operación de VIOLIN



Kangarlou, at al. [70] plantean que un estado global consistente sería el conjunto formado por los estados locales de los procesos y canales si ellos fueran guardados simultáneamente. Desafortunadamente, no es posible tomar los estados locales al mismo tiempo. La estrategia de Kangarlou, at al. [70] es obtener "un posible estado global causalmente consistente". Esto es, el conjunto de los estados locales de los procesos, obtenidos luego de un corte de eventos consistente. Un corte de eventos puede ser representado en un diagrama de tiempo, en el cual los procesos tienen una línea de tiempo. En este diagrama, los mensajes y eventos son representados por flechas y puntos, respectivamente.

La historia de cada proceso es dividida, mediante un corte de eventos, en dos partes: el pasado y el futuro. Así, la secuencia de eventos es llamada corte de eventos. Por lo tanto, un corte es consistente, de acuerdo con Kangarlou, at al. [70] y [73], si no es posible que un proceso transmisor envíe un mensaje en el futuro y que el proceso receptor lo reciba en el pasado, como el mensaje m_1 mostrado en la Figura 5.4. En esta figura, la línea punteada representa el corte de eventos.

Figura 5.4: Corte de eventos



Determinar cuáles mensajes pertenecen a cada estado local y cuándo es el momento apropiado para tomar cada *snapshot* local, de modo que el estado global sea consistente, es un desafío [74].

5.1.6. Procesamiento de paquetes de red y *coloring*

Un paquete de red es una unidad de datos de protocolo (PDU - *Protocol Data Unit*) que es transmitida a través de una red de conmutación de paquetes (*packet-switched network*), como es el caso de una red TCP/IP. En una red de conmutación de paquetes, una unidad de datos de protocolo consiste en un encabezado (formado por uno o varios campos) y una parte de datos, también conocida como carga útil (*payload*). Cada protocolo especifica la información requerida en los diferentes campos del encabezado y su longitud en bits. Un paquete es típicamente enviado por una aplicación que corre en una MF a otra aplicación que corre en otra MF. Dependiendo de la ubicación de las MFs donde se ejecutan las aplicaciones y la topología de la red o redes involucradas en la comunicación, un paquete puede pasar a través de múltiples dispositivos intermedios como enrutadores o *switches*, y a través de otras clases de dispositivos o sistemas intermedios tales como *firewalls* o servidores *proxy*. En cada etapa del camino entre las aplicaciones origen y destino el paquete puede ser leído y analizado, y algunas veces procesado, haciendo algunas modificaciones en los encabezados en las diferentes capas, sin necesidad de modificar el contenido de los mensajes de las aplicaciones.

Hay una variedad de servicios ampliamente conocidos que, para funcionar, requieren el procesamiento de paquetes de red en diferentes capas de la pila de protocolos. Entre ellas se destacan las siguientes: VLAN (*Virtual Local Area Network*), para segmentación de redes LAN; NAT (*Network Address Translation*), para facilitar el intercambio de paquetes entre redes con diferentes esquemas de direccionamiento; priorización de paquetes, para calidad de servicio en aplicaciones multimedia; servidores *proxy* caché, para acelerar la navegación en Internet; y *firewall* para controlar el tráfico entrante o saliente de una red.

5.2. Desafíos

Hemos propuesto una estrategia de mitigación a las fallas que se presentan en la fase de ejecución de las MVs en *UnaCloud*, la cual consiste en guardar el estado completo de un sistema distribuido formado por aplicaciones que corren en MVs y se comunican entre ellas.

Obtener un *snapshot* global de un sistema distribuido no es una tarea fácil [74]. En este caso, un *snapshot* global es el conjunto de estados locales de las MVs, junto con un mecanismo que permite al sistema distribuido conservar las comunicaciones en progreso. Un *snapshot* global es consistente si permite a los usuarios reanudar la ejecución del sistema sin mensajes perdidos o duplicados. Los mensajes enviados antes de tomar un estado local y que no han alcanzado su destino, es decir, aquellos que han quedado en tránsito, pertenecen al

snapshot local del proceso transmisor y por lo tanto al *snapshot* global. Por lo tanto, es necesario garantizar que después de reanudar la ejecución a partir de un *snapshot* global, estos mensajes continúan hacia su destino sin alterar la integridad de las aplicaciones.

Al abordar esta estrategia de mitigación, nosotros encontramos los siguientes desafíos:

1. Los hipervisores proporcionan una función para obtener *snapshots* de MVs individuales, pero no para sistemas distribuidos.

2. No se puede garantizar que los *snapshots* locales de las MVs sean tomados al mismo tiempo. En sistemas distribuidos los conceptos de memoria globalmente compartida o reloj global no existen, además, la red causa retardos impredecibles e inevitables.

3. Después de recuperar la ejecución de las aplicaciones que corren en las MVs, a partir de un *snapshot* global, no se puede garantizar el restablecimiento exitoso de las comunicaciones. Si el sistema usa TCP como protocolo de capa de transportes, no es suficiente el uso de sus mecanismos de confiabilidad. Si el protocolo es UDP, este protocolo no ofrece confiabilidad.

Además, se puede notar que VIOLIN depende de *switches* VIOLIN corriendo como MVs en cada MF. También, debido a que los *switches* VIOLIN son MVs, es necesario considerar los recursos requeridos para su ejecución. Con respecto al desempeño de la plataforma, es evidente que todas las comunicaciones entre dos MVs pasan a través de los *switches* VIOLIN, lo cual se convierte en un punto único de falla e incrementa la latencia ya que cada *switch* VIOLIN es un cuello de botella en su respectiva MF. Por otra parte, cuando se usan túneles UDP para enviar y recibir *frames* pasando de una MF, el rendimiento también se ve afectado debido a la sobrecarga causada por el uso de capas adicionales en el proceso de encapsulación en la pila de protocolos.

Por lo tanto, nosotros proponemos un nuevo mecanismo para obtener un *snapshot* global consistente de sistemas distribuidos corriendo en DC, basado en VNSnap, en el cual se mantenga la semántica del sistema sin modificar las aplicaciones o los hipervisores. Teniendo en cuenta algunas características de VNSnap que se pueden mejorar, y las limitaciones de los sistemas DC, hemos establecido los siguientes cuatro principios generales, los cuales van a estar presentes en la toma de decisiones de diseño del software de *snapshot* global.

1. No utilizar infraestructura adicional.

2. No tener puntos únicos de fallo.

3. Tener un bajo impacto en el desempeño del DC.

4. Proporcionar una solución liviana.

5.3. Global Snapshot

En esta sección presentamos *Global Snapshot*, un sistema que hemos desarrollado para mitigar los efectos de las interrupciones en la ejecución de MVs en un DC. Este sistema es utilizado en la validación del trabajo de investigación. A continuación presentamos los conceptos fundamentales, los elementos empleados en el desarrollo de la solución, la implementación del prototipo y la arquitectura del sistema.

5.3.1. Elementos empleados en la solución

Para enfrentar los desafíos mencionados en la sección 5.2, nosotros utilizamos los siguientes conceptos: (1) los mecanismos de confiabilidad (si aplican); (2) colores en los paquetes en los nodos, en la capa de red; (3) filtrado de paquetes, de nuevo, en la capa de red y adaptamos un protocolo simple de coordinación para obtener un *snapshot* global en cualquier momento.

5.3.1.1. Mecanismos de confiabilidad de TCP

TCP es un protocolo de capa de transporte que ofrece un servicio confiable, orientado a conexión, soportado por los siguientes mecanismos: números de secuencia, acuses de recibo, temporizadores y retransmisión de segmentos que típicamente llevan mensajes de capa de aplicación [77].

En nuestra aproximación, si las aplicaciones del sistema distribuido usan TCP como protocolo de capa de transporte, los mecanismos de confiabilidad pueden ser parte de la solución, ya que su operación usual contribuye a la preservación de las comunicaciones. Sin embargo, como fue mencionado en la sección 5.1.5 (*Snapshots* en ambientes de red virtualizados), es posible que en una hipotética recuperación de un sistema a partir de un *snapshot* global, se presente el caso del mensaje m_1 de la Figura 5.4 (§ página 61). Para prevenir esto, nuestra solución usa el concepto de colores en los datagramas salientes y filtrado de red en los datagramas entrantes. Estos dos conceptos son descritos a continuación.

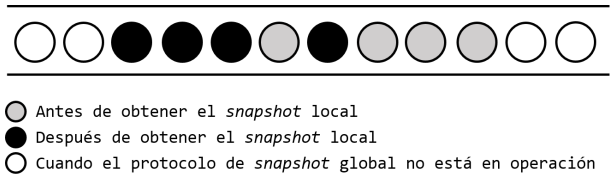
5.3.1.2. Colores

El concepto de colores en los nodos, a nivel de capa de red, consiste en escribir información de estado (o de control) en algunos campos del encabezado del datagrama IPv4, sin modificar los datos (típicamente un segmento de capa de transporte). El color es un valor previamente establecido, el cual es escrito en el origen y leído en el destino para tomar alguna decisión con respecto al color que viene en el paquete.

Nosotros usamos el concepto de colores en los nodos, en la capa de red, de forma similar al algoritmo mencionado en la sección 5.1.5. Al marcar los datagramas con un color (representado por un valor determinado), es posible saber si el protocolo de *snapshot* global está en ejecución o no, y si en el momento de

enviar un datagrama, el proceso transmisor ya había tomado o no su correspondiente *snapshot* local. Aunque el proceso transmisor hace modificaciones a los encabezados de la capa de red, no hace ninguna clase de modificación al segmento de capa de transporte, el cual al mismo tiempo, lleva los mensajes de capa de aplicación del sistema distribuido. La Figura 5.5 ilustra el concepto de colores.

Figura 5.5: Colores en los nodos

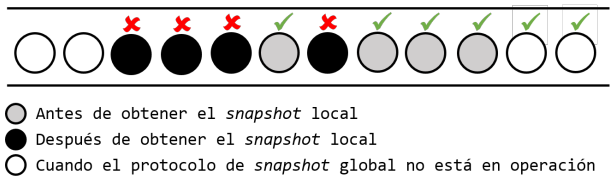


Observe en la Figura 5.5 que los paquetes tienen de color blanco son usados para llevar segmentos de capa de transporte cuando el protocolo de *snapshot* global no está en ejecución. Los datagramas de color gris son usados para indicar que el *snapshot* local en el nodo no ha sido tomado. Finalmente los datagramas de color negro indican que el *snapshot* local ya fue obtenido.

5.3.1.3. Filtrado de paquetes de red

Como ya lo mencionamos, no debe suceder que un mensaje enviado en el futuro sea recibido en el pasado, como lo muestra el mensaje m_1 en la Figura 5.4 (Página 61). Para prevenirlo, nosotros ponemos una marca o color (un valor determinado) en los datagramas salientes para indicar si el *snapshot* local ha sido tomado o no. Al llegar el datagrama a la MV destino, nosotros examinamos y analizamos su contenido. Si el *snapshot* local de la MV en el destino no ha sido tomado, los datagramas con color que indiquen que el *snapshot* local en la MV origen ya había tomado al momento de enviar el datagrama, serán descartados, tal como se puede apreciar en la Figura 5.6.

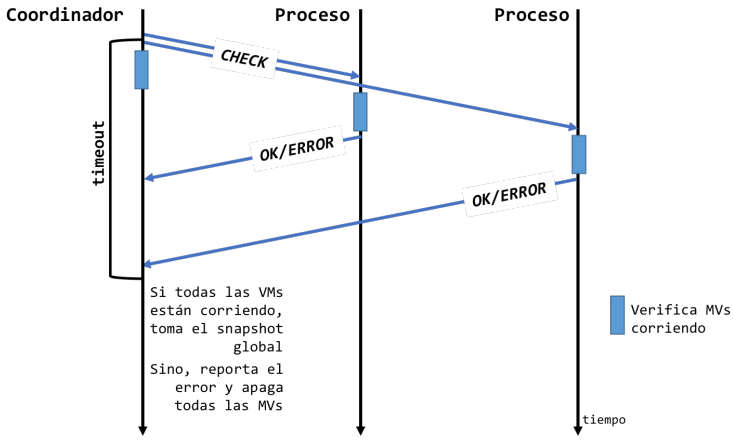
Figura 5.6: Filtrado de paquetes



5.3.1.4. Protocolo simple de coordinación

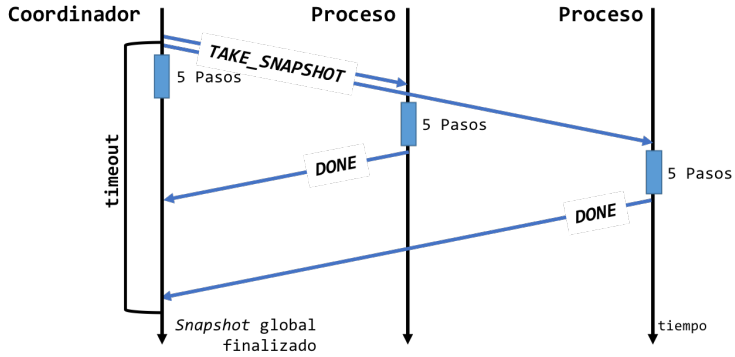
Teniendo en cuenta que no es posible garantizar que los *snapshots* locales son tomados al mismo tiempo, nosotros diseñamos un protocolo simple de coordinación (de dos fases), el cual se puede ver en las Figuras 5.7 y 5.8.

Figura 5.7: Protocolo de *snapshot* global — Fase 1: Verificación previa



La primera fase, consiste en una verificación previa en la que el coordinador consulta a cada proceso si las MVs correspondientes a la MF están en ejecución, tal como se muestra en la Figura 5.7. Si todas las MVs están corriendo, pasamos a la fase 2, en la cual se toma el snapshot global.

Figura 5.8: Protocolo de *snapshot* global — Fase 2: Obtener el snapshot global



La segunda fase consiste la toma del *snapshot* global. Aplicando el concepto de colores para identificar los datagramas, nosotros utilizamos los datagramas sin color cuando el protocolo de *snapshot* global no está en operación, es decir, antes de iniciar y luego de terminar, y asumimos que un *snapshot* global no va a iniciar antes de terminar el anterior.

Luego de iniciar el protocolo, los datagramas que son enviados antes de tomar el *snapshot* local se marcan con el color *PRESNAPSHOT*, mientras que aquellos que son enviados después de obtener el *snapshot* local se marcan con el color *POSTSNAPSHOT*.

En este protocolo, todos los procesos son idénticos. El proceso que inicia

su ejecución antes que los demás toma el rol de *coordinador* y es el encargado de activar el protocolo. El coordinador envía a todos los demás procesos participantes un mensaje con el texto *TAKE_SNAPSHOT*. Después de enviar el mensaje, el coordinador ejecuta la siguiente secuencia de seis pasos:

1. Colores. Todos los datagramas salientes van a llevar el color *PRESNAPSHOT*.
2. Filtrado de paquetes. Todos los datagramas entrantes que vienen con el color *POSTSNAPSHOT* son descartados. Si llegara un datagrama sin color, este datagrama habría sido enviado antes de tomar el *snapshot* local en la MV origen, por lo tanto pertenece al *snapshot* global.
3. Se obtiene el *snapshot* local.
4. Colores. Todos los datagramas salientes van a lleva el color *POSTSNAPSHOT*.
5. Filtrado de paquetes. Todos los datagramas entrantes que lleguen con el color *POSTSNAPSHOT* son aceptados.
6. Colores. Ningún datagrama saliente va a llevar color.

Note que los colores *PRESNAPSHOT* y *POSTSNAPSHOT* hacen referencia a las marcas que se harán en los datagramas salientes para indicar que al momento de enviar cada datagrama en particular el *snapshot* local en el proceso transmisor ha sido tomado o no.

Después de recibir el mensaje *TAKE_SNAPSHOT* enviado por el coordinador, cada proceso ejecuta la misma secuencia de seis pasos y le envía como respuesta un mensaje *DONE* indicando que su *snapshot* local ya ha sido tomado.

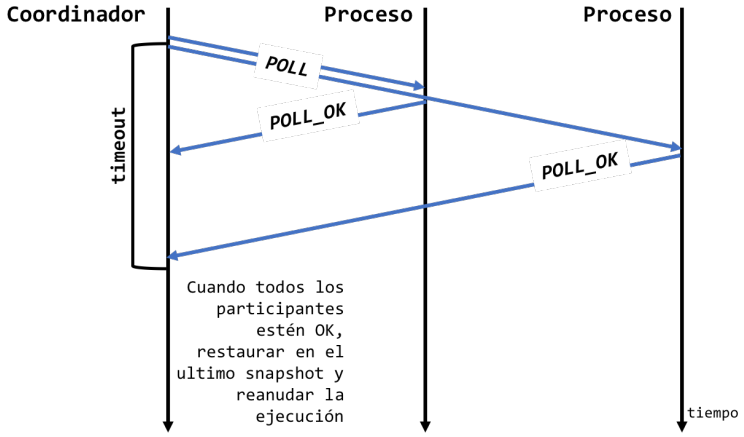
El coordinador mantiene registro de los mensajes *DONE* recibidos. Cuando recibe un mensaje *DONE* de todos los demás procesos, y ya ha ejecutado la secuencia de seis pasos, termina la ejecución del protocolo y el *snapshot* global está completo.

Para que el coordinador pueda dar por terminado el *snapshot* global, es necesario que se cumplan dos condiciones: (1) que el coordinador haya ejecutado la secuencia de seis pasos, y (2) que el coordinador haya recibido todos los mensajes *DONE* enviados por los demás procesos participantes. Dado que todos los participantes (incluyendo al coordinador) se ejecutan de manera concurrente, lo normal es que el coordinador termine primero que los demás participantes. Sin embargo, también puede darse el caso en que el coordinador primero reciba todos los mensajes DONE y cuando esto ocurra, no haya terminado la secuencia de los seis pasos. Esta eventualidad es tenida en cuenta en la implementación de la solución.

Por su parte, la solución cuenta con un protocolo de reanudación para coordinar el proceso cuando se desea continuar la ejecución de un sistema distribuido a partir de un *snapshot* global. En este caso, tampoco se puede garantizar

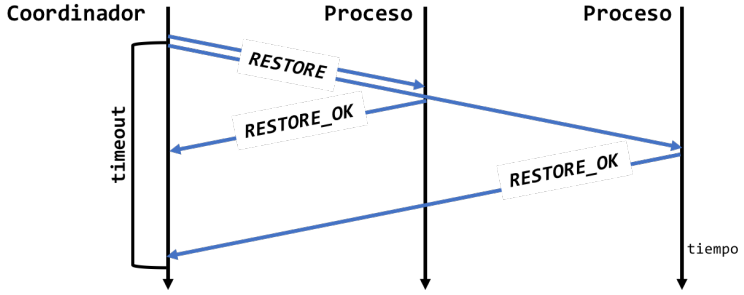
que al reanudar la ejecución de las MVs a partir de un *snapshot* global, estas lo hagan al mismo tiempo. Por lo tanto, nosotros creamos protocolo de tres fases, para coordinar la reanudación del sistema, buscando que las comunicaciones sean restablecidas y que el sistema pueda finalizar correctamente su ejecución. Este protocolo se puede apreciar en las Figuras 5.9, 5.10, y 5.11.

Figura 5.9: Protocolo de reanudación — Fase 1: Sondeo



La primera fase consiste en verificar que las máquinas físicas, donde se ejecutan las MVs, están corriendo. Esta es una verificación periódica. El coordinador envía un mensaje *POLL* a los demás procesos participantes. Los otros procesos que reciben el mensaje *POLL* y envían un mensaje *POLL_ OK* al coordinador. Si alguna de las MFs no está lista, se activa un temporizador y luego de un tiempo de espera, la verificación vuelve a comenzar. Cuando todas las MFs hayan contestado con el mensaje *POLL_ OK* al coordinador, pasamos a la fase 2.

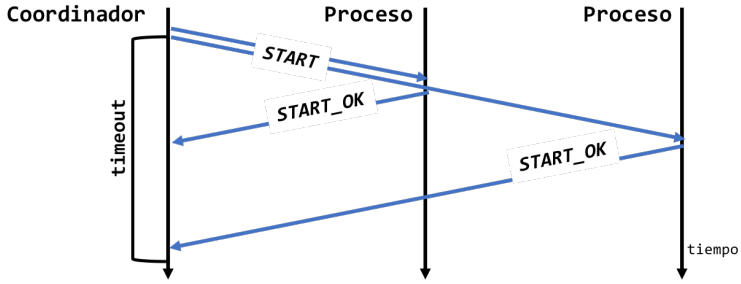
Figura 5.10: Protocolo de reanudación — Fase 2: Asignación de punto de restauración



En la segunda fase se establece el punto de restauración en el *snapshot* (típicamente el último), aunque podría ser cualquier snapshot intermedio. El

coordinador envía un mensaje *RESTORE* a los demás procesos participantes. Los otros procesos que reciben el mensaje *RESTORE* y envían un mensaje de respuesta al coordinador. Cuando todas las MFs hayan contestado con el mensaje *RESTORE_ OK* al coordinador, pasamos a la fase 3.

Figura 5.11: Protocolo de reanudación — Fase 3: Reinicio de ejecución



La tercera fase es similar a la segunda. El coordinador envía a los demás participantes un mensaje *START*. Los demás participantes envían de respuesta al coordinador un mensaje *START_ OK*. Cuando el coordinador determina que la reanudación ha sido exitosa cuando reciba la respuesta de todos los participantes.

En la sección 5.4 (Página 73) presentamos el análisis de correctitud del protocolo y explicamos detalladamente su funcionamiento, tanto en el proceso de obtener el *snapshot* global como en la reanudación de la ejecución. Esta explicación incluye los datagramas que fueron enviados después de haber tomado el *snapshot* local en la MV origen que hayan sido recibidos en la MV de destino antes de obtener el *snapshot* local, es decir, el caso en el que un paquete *POSTSNAPSHOT* es descartado.

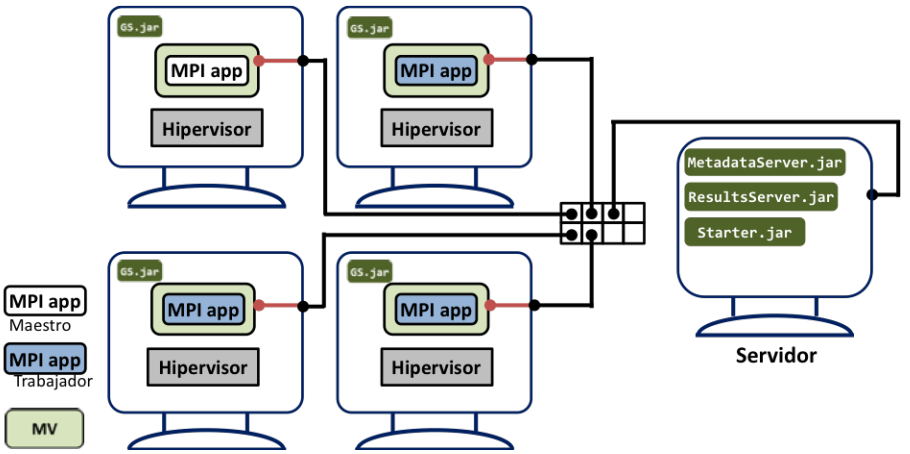
5.3.2. Implementación

Nosotros desarrollamos un prototipo funcional denominado *Global Snapshot (GS)*, en el cual implementamos el protocolo de coordinación para obtener un *snapshots* global consistente. Es un sistema *peer-to-peer*, es decir, la misma aplicación se ejecuta en todas las MFs que alojan las MVs administradas por *UnaCloud*. Los participantes actúan como servidores y clientes simultáneamente. Este sistema es ejecutado al mismo nivel que los hipervisores, como es mostrado en la Figura 5.12.

El sistema tiene la habilidad de obtener, en paralelo, los *snapshots* locales para más de una MV corriendo en la misma MF, siempre que pertenezcan al mismo despliegue. Las tecnologías usadas en el desarrollo de este sistema son: el lenguaje de programación *Java* y el firewall *iptables*. El desarrollo fue realizado para su operación con *Oracle VirtualBox* aunque en principio, UnaCloud tiene soporte para VMware WorkStation.

El sistema depende de un servidor de metadatos (*metadata server – MS*) que maneja la estructura del sistema y almacena la información de red de los

Figura 5.12: Vista general del sistema



participantes. En este caso, los participantes son las instancias del programa *GS*, corriendo una en cada MF donde se ejecutan las MVs a las que se le va a obtener un *snapshot* global.

El MS puede correr en cualquier computador que pueda ser accedido por la red. El MS se utiliza para varios propósitos: registra los participantes del sistema; permite a los usuarios consultar acerca de la información de red de un participante específico, la lista de participantes, y el número de participantes en el sistema; registra el tiempo inicial y final de cada *snapshot* local y del *snapshot* global; y coordina el inicio y fin de la operación del *snapshot* global. El MS reporta en un archivo de resultados los tiempos de *snapshot* local de los participantes y el tiempo de *snapshot* global del sistema distribuido en el DC en un formato apropiado para realizar el análisis estadístico.

El MS utiliza números enteros consecutivos (a partir de 0) como identificadores para los participantes. El participante que obtenga el identificador 0 asume el rol de coordinador. El coordinador envía una consulta al MS cuando necesita interactuar con los otros participantes.

El MS recibe un mensaje “StartGS"de una fuente externa y como resultado, da al coordinador la señal de inicio del protocolo. Esto significa que el protocolo de *snapshot* global puede ser activado de manera independiente al sistema que corre en el DC.

Como métrica de desempeño, el MS registra el tiempo que toma obtener el *snapshot* local de cada participante, y el tiempo que toma obtener el *snapshot* global de todo el sistema distribuido. Para simplificar esta tarea, el coordinador recibe en el mensaje *DONE* el tiempo local de cada participante, al terminar la ejecución del protocolo, calcula el tiempo de *snapshot* global. Estos resultados son enviados al MS para que los tiempos sean registrados.

Nosotros implementamos el concepto de colores en los nodos (en las MVs) y usamos *iptables* para establecer los valores que representan los colores usando

el campo *Diffserv* en los datagramas IPv4. Para hacer esto, *iptables*, mediante la tabla *mangle* modifica los datagramas salientes. La Figura 5.13 (a) muestra un ejemplo en el cual una regla es agregada a la tabla *mangle* para modificar el campo *Diffserv* (*DSCP - Diffserv Code Point*) de los datagramas salientes asignando como color el valor $0x10$. De manera similar, para el filtrado de paquetes, nosotros usamos *iptables* para analizar cada datagrama entrante y determinar si, de acuerdo al color del datagrama, este debe ser entregado a su destino o no. La Figura 5.13 (b) muestra un ejemplo de la configuración de *iptables* en la cual una regla es adicionada a la cadena *INPUT*, indicando que cualquier datagrama entrante que trae el valor $0x10$ en el campo *Diffserv* será descartado. Esta configuración es ejecutada en todas las MVs y no afecta significativamente el desempeño de las MFs o su comportamiento.

Figura 5.13: Ejemplo de configuración de reglas de *iptables* en máquinas virtuales

```
iptables -t mangle -A OUTPUT -j DSCP --set-dscp 0x10
```
(a)

```
iptables -A INPUT -m dscp --dscp 0x10 -j DROP
```
(b)

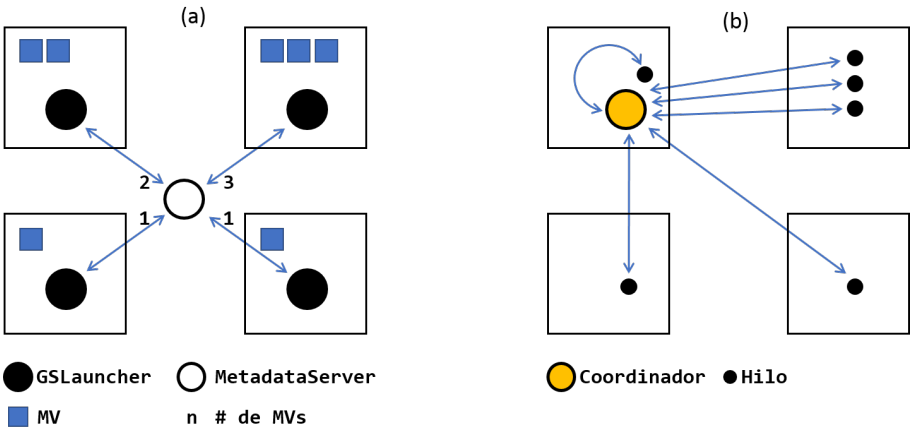
El sistema nos permite obtener *snapshots* globales consecutivos y restaurar la ejecución de las MVs a partir de ellos.

5.3.3. Arquitectura del sistema

Como fue mencionado, el sistema de *Snapshot* Global (GS) está soportado en un servidor de metadatos. Al comienzo, como se muestra en la Figura 5.14 (a), el MS es contactado por el *GSLauncher* (uno por cada MF) para insertar un registro para cada MV que se ejecuta en una máquina física. La información de red del MS es conocida por el *GSLauncher* a través de un archivo de propiedades. El MS recibe el comando *INSERT* seguido de la dirección IP de la máquina física donde se ejecuta una instancia del *GSLauncher*. El MS recibe el mensaje, lo analiza, lo valida y almacena el registro en una estructura de datos. En respuesta, el MS envía un identificador junto con un número de puerto. Con cada identificador, el *GSLauncher* crea un hilo, el cual es responsable de tomar los *snapshots* locales de una MV que le es asignada. Este identificador será el valor que corresponde al hilo participante del sistema de *snapshot* global dentro de la estructura de datos principal. El número de puerto será usado para enviar y recibir mensajes que son parte del protocolo. El identificador 0 está reservado para el coordinador. Los números de puerto inician en el número configurado como *base* en el archivo de propiedades del MS.

Por ejemplo, en la Figura 5.14 (a), se puede ver que hay cuatro MFs con 2, 3, 1 y 1 VMs respectivamente. La Figura 5.14 (b) muestra los hilos responsables de tomar los *snapshots* locales de cada MV, uno de ellos actuando como coordinador. Además, es importante resaltar que la comunicación es ejecutada solo entre el coordinador y los demás participantes del sistema.

Figura 5.14: Interacción de componentes — (a) Fase de registro (b) Fase de ejecución



La Figura 5.15 nos permite apreciar la secuencia de eventos que ocurre durante la fase de registro del sistema. En esta figura, nosotros podemos ver un ejemplo de un sistema con dos instancias del *GSLauncher*. En este ejemplo, hay dos MVs en la MF 1 y una MV en la MF 2. El *GSLauncher* que está corriendo en la MF 1 se registra en el MS mediante un mensaje *INSERT* a través del cual envía su dirección IP. En respuesta, el MS le envía el identificador 0 y el número de puerto que debe configurar para recibir mensajes por la red. A continuación, este GSLauncher crea el hilo coordinador (*GS0*), debido a que es el primer participante en registrarse en el MS y le envía la información requerida para iniciar su ejecución, incluyendo el nombre de la MV a la cual le va a obtener el *snapshot* local. Los nombres de las MVs son conocidos por el *GSLauncher* en cada MF cuando es ejecutado (en la Figura 5.15 estos nombres van escritos en cursiva). Note que el *GSLauncher* activa el coordinador, y éste queda bloqueado hasta que el MS le envíe un mensaje *START* para dar inicio a la ejecución del protocolo.

La Figura 5.16 muestra la fase de ejecución del protocolo de *snapshot* global. Desde cualquier fuente externa llega el mensaje "*StartGS*" al MS. Luego, el MS envía al coordinador el mensaje "*START*". De esta forma, el coordinador envía el mensaje "*TAKE_SNAPSHOT*" a los otros participantes, y todos ejecutan el protocolo mostrado en la Figura 5.8 (§ Página 66).

Al obtener el *snapshot* local, cada hilo envía el mensaje *"DONE"* correspondiente junto con el tiempo de *snapshot* local. Luego de que todos los participantes hayan enviado sus mensajes *"DONE"*, el hilo coordinador (*GS0*) envía al MS los tiempos de los *snapshots* locales y el global.

Con respecto al sistema de recuperación, este comparte las mismas características del sistema de *snapshot* global, excepto que no usa el *iptables* firewall.

Figura 5.15: Diagrama de secuencia – Fase de registro

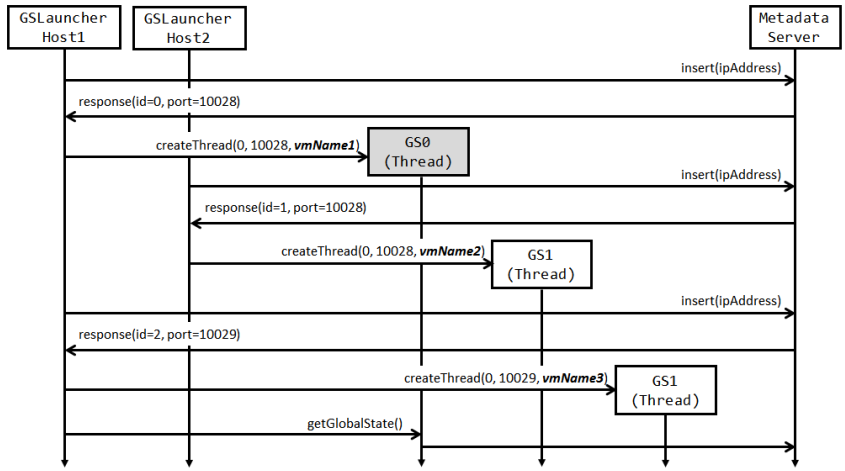
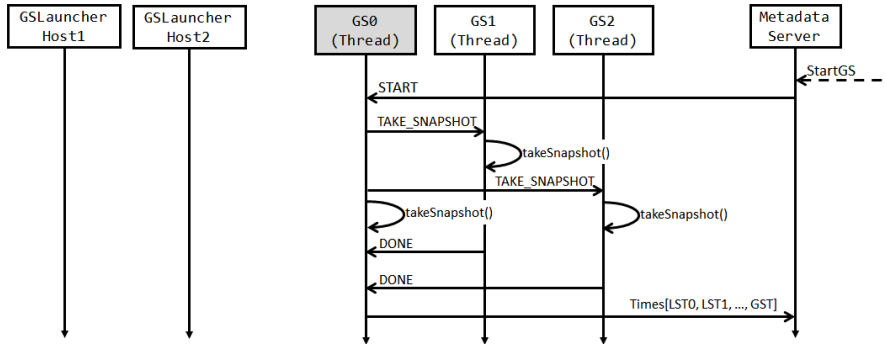


Figura 5.16: Diagrama de secuencia – Fase de ejecución



5.4. Análisis de correctitud

Esta sección presenta el análisis de correctitud de nuestro protocolo de *snapshot* global. El protocolo mantiene la semántica del sistema soportado por los principios de transferencia confiable de TCP, si aplica. De esta forma, las aplicaciones que corren en las MVs no se ven afectadas por la ejecución del protocolo de *snapshot* global. En caso de falla, las aplicaciones pueden reanudar su ejecución a partir del último *snapshot* global obtenido y finalizarla exitosamente. Si no hay falla, la ejecución continúa manteniendo la consistencia del sistema.

Nosotros hemos tomado como referencia la aplicabilidad del algoritmo presentada por Kangarlou, et al. en [29], ya que consideramos que está muy bien detallada y nos permite expresar la correctitud de nuestro algoritmo.

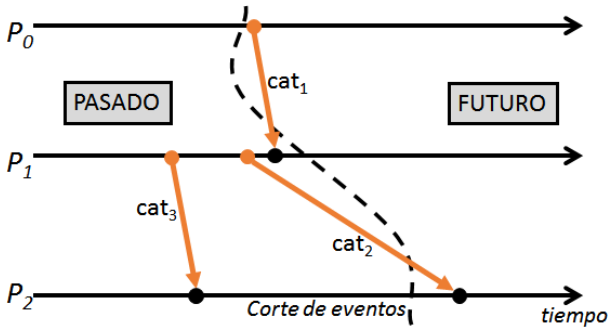
5.4.1. Aplicaciones con TCP como protocolo de capa de transporte

Suponga que un mensaje m es enviado por una aplicación que se está ejecutando en una MV y el mensaje está destinado a una aplicación que se ejecuta en otra MV. En este caso no importa en cuáles MFs se están ejecutando las MVs. Considere lo siguiente:

- Sea S el conjunto de segmentos TCP que llevan los mensajes de capa de aplicación en su campo de datos.
- Sea s cualquier segmento TCP, tal que $s \in S$.
- Sea a_s el segmento con el acuse de recibo que es enviado por el receptor para indicar que el segmento s ha sido recibido correctamente.
- Sea $d(s)$ el datagrama IP que lleva el segmento s en su campo de datos.
- Sea $d(a_s)$ el datagrama IP que lleva el acuse de recibo a_s en su campo de datos.

Los datagramas IP que llevan los segmentos de capa de transporte se pueden dividir en tres categorías, de acuerdo con la Figura 5.17.

Figura 5.17: Categorías de los mensajes



- Categoría 1. Cuando $d(s)$ fue enviado después de que fue tomado el *snapshot* local en la MV origen y fue recibido en la MV destino antes de que se tomara el *snapshot* local, es decir, el datagrama viene del futuro al pasado. Vea este mensaje con la etiqueta cat_1 en la Figura 5.17, donde la línea punteada que representa el corte de eventos, indica el tiempo en que se toma el *snapshot* local en cada proceso.

  De acuerdo con la segunda condición requerida para que un *snapshot* global sea consistente [74] (mencionada en la sección 5.1.5 *Snapshots* en ambientes de red virtualizados) de la Página 60, no es posible tener un

datagrama de categoría 1 en el sistema. Por lo tanto, el datagrama debe ser descartado.

Note que si un datagrama es descartado, como lo establece el protocolo propuesto, ni el envío del datagrama $d(s)$ hace parte del *snapshot* local de la MV origen, ni el *snapshot* local de la MV destino incluye la recepción del datagrama. Al reanudar la ejecución de la aplicación a partir del *snapshot* global, la MV origen envía de nuevo a su destino el datagrama $d(s)$, pero en esta oportunidad, es un nuevo datagrama. De manera similar, la MV destino recibe el datagrama $d(s)$ por primera vez (ya que el anterior fue descartado) y entrega el segmento s al proceso destino.

Por el contrario, si no hay falla en el sistema, pero un datagrama $d(s)$ de esta categoría ha sido descartado en el destino, el protocolo TCP en el lado transmisor es responsable de retransmitir el segmento s hasta que la MV en el destino haya obtenido su *snapshot* local y la regla en el *firewall iptables* sea modificada para permitir al mensaje alcanzar la capa de aplicación.

Es importante resaltar que descartar paquetes que vengan del futuro al pasado puede ocasionar la retransmisión de datagramas. Sin embargo, el impacto en el desempeño del sistema es mínimo debido a que esto solo ocurre durante uno de los seis pasos del protocolo y todo sucede en fracción de segundos, justo antes de tomar el *snapshot* local la MV de destino.

- Categoría 2. Cuando el datagrama $d(s)$ fue enviado antes de que la MV origen tomara el *snapshot* local, y al momento de tomar el *snapshot* local en la MV destino el datagrama no ha llegado. Es decir, el datagrama $d(s)$ viene del pasado y probablemente llegará en el futuro. Vea este mensaje con la etiqueta cat_2 en la Figura 5.17.

  Note que el *snapshot* local en la MV destino no incluye la recepción del datagrama, mientras que el *snapshot* local en la MV origen sí incluye el envío. Es claro que el *snapshot* local en la MV origen no incluye el acuse de recibo ya que no ha sido enviado por la MV destino. Después de reanudar la ejecución del sistema a partir de un *snapshot* global, el protocolo TCP en la MV origen (como consecuencia de un *timeout*) asume que el datagrama $d(s)$ se ha perdido y lo reenvía para continuar con su ejecución.

  Si no hay falla en el sistema, la ejecución continúa normalmente. Cuando el datagrama llegue a su destino, el destino enviará su acuse de recibo.

- Categoría 3. Cuando el datagrama $d(s)$ fue enviado y recibido en el pasado. Vea este mensaje con la etiqueta cat_3 en la Figura 5.17. En este caso, de acuerdo con la Figura 5.18 podemos considerar:

  - Categoría 3.1. Cuando el datagrama que lleva el acuse de recibo, $d(a_s)$, llega en el pasado. Vea este mensaje con la etiqueta $cat_{3,1}$

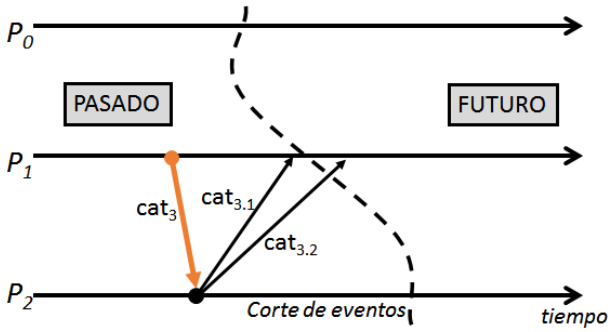


Figura 5.18: Mensajes y acuses de recibo

en la Figura 5.18. En este caso, se considera que el datagrama $d(s)$ ha sido exitosamente entregado en el destino. Como el mensaje fue enviado y recibido antes de ser obtenidos los *snapshots* locales en las MVs origen y destino, los dos eventos, el envío del datagrama y su recepción, hacen parte del *snapshot* global.

- Categoría 3.2. Cuando el datagrama que lleva el acuse de recibo, $d(a_s)$ llega en el futuro, es decir, en el momento de tomar el *snapshot* local en la MV destino, el datagrama que lleva el acuse de recibo, $d(a_s)$, no ha sido recibido. Vea el mensaje con la etiqueta $cat_{3,2}$ en la Figura 5.18. En este caso, el acuse de recibo probablemente llegará en el futuro, de manera que no es incluido en el *snapshot* local de la MV origen.

  Después de recuperar la ejecución del sistema a partir de un *snapshot* global, el protocolo TCP en la MV origen (como consecuencia de un *timeout*), asume que el datagrama $d(s)$ se ha perdido y lo reenvía para continuar con la ejecución normal del sistema. Por su parte, el protocolo TCP en la MV destino (usando el número de secuencia) identifica que el segmento ya había sido recibido (lo detecta como un duplicado) y lo ignora. Sin embargo, el receptor envía el correspondiente acuse de recibo al transmisor dentro de un datagrama.

  Si no hay falla, como pasa en los datagramas de la categoría 2, el sistema continúa su ejecución normalmente, soportada por el protocolo TCP.

5.4.2. Aplicaciones con UDP como protocolo de capa de transporte

Las aplicaciones con el protocolo UDP como protocolo de capa de transporte, a diferencia de aquellas que usan TCP, no envían acuses de recibo. En este caso, nosotros consideramos las mismas tres categorías como en el protocolo TCP, las cuales son mostradas en la Figura 5.17 de la Página 74.

- Categoría 1. Como en el protocolo TCP, un datagrama que viene desde el futuro al pasado debe ser descartado por el protocolo de *snapshot* global.

- Categoría 2. El *snapshot* local de la MV origen incluye el envío del datagrama $d(s)$, pero el *snapshot* local de la MV destino no incluye su recepción. Ya que UDP no envía acuses de recibo, el datagrama es considerado perdido. Es parte del comportamiento esperado.

- Categoría 3. En este caso, se considera que el datagrama fue entregado exitosamente y tanto el envío como la recepción del datagrama hacen parte de los *snapshots* locales de las MVs origen y destino, y naturalmente del *snapshot* global del sistema.

A continuación presentamos los resultados obtenidos de la evaluación realizada en términos de tiempo de ejecución del algoritmo de *snapshot* global y el sobrecosto de ejecutar el protocolo de *snapshot* global en el tiempo de ejecución del sistema distribuido que utilizaremos en la prueba.

Luego de obtener el *snapshot* global, nosotros simulamos una falla en el sistema apagando las máquinas físicas. A continuación, restauramos el sistema a partir del último *snapshot* global obtenido. El sistema distribuido finalizó su ejecución con los resultados esperados.

5.5. Evaluación y resultados

Esta sección reporta los resultados de nuestra evaluación usando los hosts donde se ejecuta *UnaCloud*. Nosotros estamos interesados en determinar la variación del tiempo necesario para obtener los *snapshots* locales y el *snapshot* global, dependiendo del número de MVs corriendo al mismo tiempo en la misma MF. Por lo tanto, consideramos las siguientes métricas: tiempo de *snapshot* global y tiempo de *snapshot* local.

Nosotros describimos la configuración de los experimentos y, a continuación, explicamos en detalle el sistema distribuido que corre en las MVs y al que se le toman los *snapshots* globales. Finalmente, presentamos los experimentos, los resultados y el análisis respectivo.

5.5.1. Configuración de Experimentos

Nosotros probamos el protocolo de *snapshot* global en 16 computadores de escritorio homogéneos en una sala de cómputo con la siguiente configuración. Procesador Intel (R) Core i5-2300 @ 2.80GHz, 8 GB de memoria RAM y Microsoft Windows 7 como sistema operativo. La sala de computadores estaba conectada por una red LAN de 1 Gbps y el hipervisor *Oracle VirtualBox*. Para nuestras pruebas, instalamos cuatro MVs idénticas en cada MF con 1 núcleo de procesamiento, 650 MB de memoria RAM, 20 GB de disco duro y Debian GNU/Linux 8.7 (jessie) como sistema operativo con interfaz de modo texto.

5.5.2. Sistema distribuido

Nosotros desarrollamos un sistema distribuido para realizar las pruebas funcionales y de desempeño de la solución de *snapshot* global. Al finalizar su ejecución, el sistema desarrollado permite comprobar que el protocolo de *snapshot* global funciona correctamente sin interferir con las aplicaciones que se ejecutan en las MVs, y sin importar si el protocolo se ejecutó sobre el sistema distribuido o no.

El software ha sido denominado *Ring-based Cloud Performance Benchmark Tool RCPBT* (herramienta de comparación de desempeño basada en anillo para cloud). Es un sistema *peer-to-peer* en el que los participantes forman un anillo ordenado de acuerdo a un identificador consecutivo (iniciando en 0), en el cual un *token* numerado es pasado de un participante al siguiente en el anillo hasta agotar un número determinado de *tokens*. El sistema ha sido ejecutado exitosamente en instancias de *Amazon Web Services*, y aunque está pensado para su ejecución en MVs, nada impide que se ejecute en MFs de uso común.

Desde el punto de vista funcional, el sistema tiene dos componentes: El *TestManager* (*TM*) y el *RingTokenPassing* (*RTP*).

5.5.2.1. *TestManager*

El *TM* es un componente requerido para el soporte de la aplicación. Se encarga de las siguientes tres funciones: (1) gestionar la configuración del sistema, (2) mantener la estructura del anillo, y (3) preparar las estadísticas de desempeño del sistema. Estas funciones se describen a continuación.

Gestionar la configuración del sistema. El *TM* inicia su ejecución leyendo, de un archivo de propiedades, los parámetros que serán utilizados en su configuración, junto con los parámetros que especifican la ejecución del anillo y otros necesarios para los resultados de las pruebas de desempeño.

Para la configuración del *TM*, en el archivo de propiedades se encuentra el número de puerto a través del cual va a escuchar los mensajes que le envíen. En la configuración del anillo se tienen el número máximo de mensajes que se van a intercambiar entre los participantes, el puerto base que será utilizado para establecer el número de puerto local en el que los participantes escucharán sus mensajes, y el número de participantes que tendrá el anillo. Además, para los resultados de las pruebas de desempeño, se puede especificar una etiqueta que permite identificar la prueba realizada.

Mantener la estructura del anillo. El *TM* recibe peticiones de los participantes en el sistema. Al iniciar su ejecución, cada participante se registra en el *TM*. Como respuesta a esta solicitud, el *TM* le retorna un identificador entero, junto con los parámetros de configuración de la ejecución del sistema. Lo anterior permite que se ingresen los parámetros de configuración del anillo en el *TM*. Estos parámetros son iguales para todos los participantes. El *TM* es el componente en el cual se especifican los parámetros de configuración y

los difunde a los participantes como respuesta al registro inicial. Estos parámetros se pueden cambiar para una nueva prueba, haciendo las modificaciones únicamente en el archivo de propiedades del *TM*.

El participante que se registre primero tendrá el identificador 0 y asumirá el rol de iniciador. Este participante será el encargado de enviar el primer mensaje a su sucesor en el anillo, cuando reciba una señal del *TM*. El *TM*, mediante una propiedad de configuración, conoce el número de participantes en cada prueba. Al registrar la cantidad esperada, el *TM* no recibe más participantes y envía al iniciador un mensaje `START`, como señal, para que inicie la transmisión de mensajes.

Cada participante consulta al *TM* la información de red de su sucesor (dirección IP y número de puerto) en el anillo, antes de enviarle su primer mensaje.

Preparar las estadísticas de desempeño del sistema. El *TM* recibe la solicitud de registro del tiempo inicial de la ejecución del anillo por parte del iniciador. Además, cada participante, al terminar, registra el tiempo final de ejecución. De este modo, cuando todos los participantes del anillo hayan salido del sistema, el *TM* puede calcular la duración total de la ejecución del anillo, y almacenar los resultados en un archivo en formato *.csv* (*comma separated value*) para su análisis. El formato del archivo se puede apreciar en la Tabla 5.1.

Tabla 5.1: Formato del archivo de resultados en el sistema desarrollado para pruebas

Tokens	*Duración [s]*	*Etiqueta*	Fecha de prueba
200000	122.1285	Prueba GS #1	Thu Mar 15 17:15:15 COT 2018
200000	125.2179	Prueba GS #1	Thu Mar 15 17:52:15 COT 2018
100000	98.5046	Prueba GS #2	Thu Mar 16 11:45:35 COT 2018
100000	97.1336	Prueba GS #2	Thu Mar 16 11:59:47 COT 2018

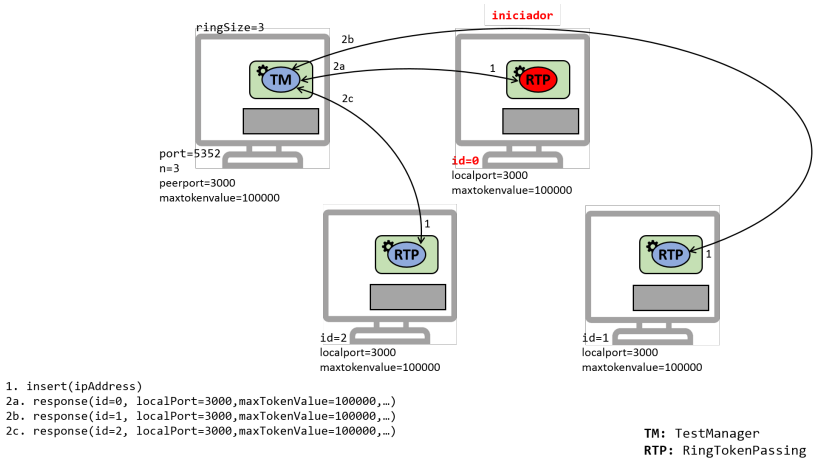
5.5.2.2. *RingTokenPassing*

RTP es el componente que implementa los participantes en un sistema *peer-to-peer*, cuyo comportamiento es de cliente y servidor al mismo tiempo. Las instancias del *RTP* forman el anillo en el cual cada participante: (1) recibe un *token* o mensaje numerado enviado por el predecesor; y (2) envía el *token* incrementado en 1 al sucesor.

El *RTP* obtiene de un archivo de propiedades la información de red del *TM* para interactuar con él de la siguiente forma: (1) conocer su identificador en el anillo, el número de puerto a través del cual va a recibir los mensajes, y el número máximo de *tokens* numerados que serán intercambiados por los participantes; (2) consultar la información de red del sucesor para enviarle mensajes; y (3) registrar el tiempo inicial y final de la ejecución del anillo para medir el tiempo total de la ejecución del sistema.

Hay tres fases en la ejecución del *RTP*: (1) registro, (2) operación, y (3) finalización. *RTP* ha sido desarrollado para ser una aplicación de ejecución automática en MVs, fácilmente replicable en entornos de computación en la nube como *UnaCloud* o *Amazon Elastic Compute Cloud* (*Amazon EC2*) donde se espera tener múltiples participantes.

Figura 5.19: RCPBT. Fase de registro.

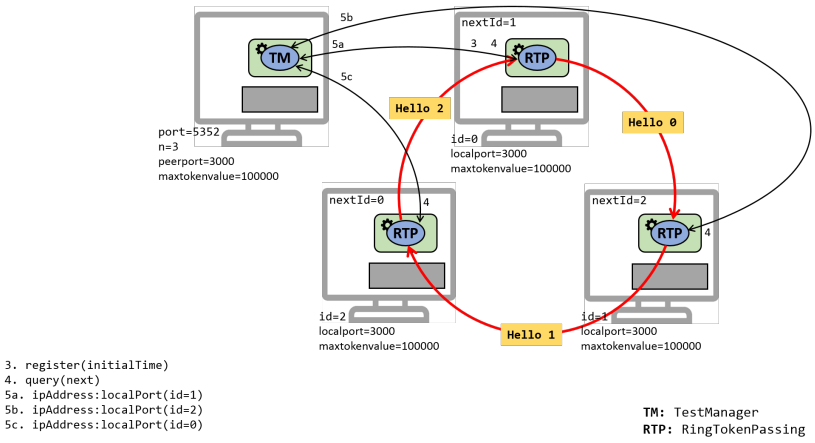


Fase de registro. Como se puede apreciar en la La Figura 5.19, en la parte superior izquierda, el *TM* corre en una MV que a su vez se ejecuta en una MF. También se pueden ver las propiedades del sistema que son leídas al iniciar la ejecución del *TM*. Mientras que *port* se refiere al número de puerto que utilizará el *TM* para escuchar sus mensajes, las otras tres propiedades son de configuración de la prueba. En la Figura 5.19 aparecen tres instancias del *RTP*, y los parámetros de configuración que recibe luego de registrarse en el *TM*. Cada instancia recibe su identificador y con base en este número se determinan el predecesor y sucesor de cada una.

Al iniciar su ejecución, cada instancia del *RTP* se registra con su dirección IP ante el *TM* y recibe el identificador que le da su posición en el anillo, el puerto que debe configurar para escuchar mensajes, y el número máximo de mensajes que van a ser intercambiados entre los participantes. El participante que reciba 0 como identificador, asumirá el rol de iniciador del anillo y quedará bloqueado esperando a que el *TM* le envíe un mensaje de activación, cuando se hayan registrado los participantes esperados para dar inicio al intercambio de mensajes.

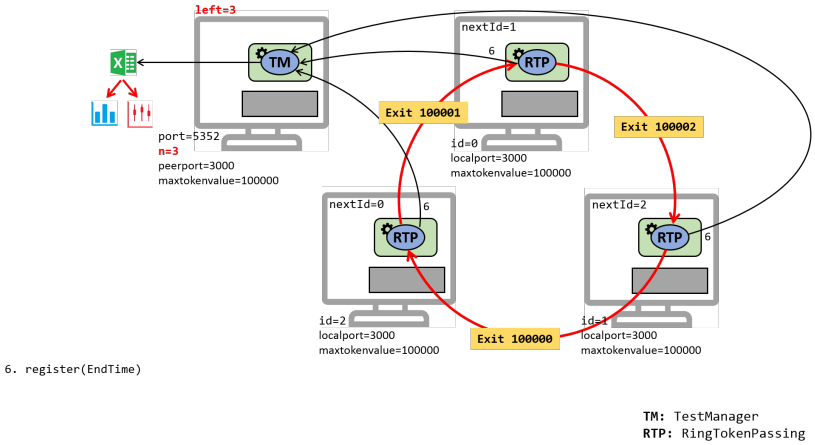
Fase de operación. La operación es la fase en la que se intercambia la cantidad de *tokens* que se ha establecido como propiedad de configuración en el *TM* y enviada durante el registro a cada instancia del *RTP*. Para poder enviar los

Figura 5.20: RCPBT. Fase de operación.



tokens a su sucesor, cada instancia del *RTP* solicita al *TM* la correspondiente información de red. El iniciador envía el mensaje *Hello 0* a su sucesor, el cual a continuación envía el mensaje *Hello 1* al sucesor que le corresponde. Éste, envía *Hello 2* al suyo, y así sucesivamente. Cada *token* es enviado por uno de los procesos justo después de recibirlo e incrementarlo en 1. La Figura 5.20 muestra el inicio de la fase de operación del sistema distribuido.

Figura 5.21: RCPBT. Fase de finalización



Fase de finalización. Al llegar al máximo de mensajes a intercambiar (por ejemplo 100000), el sistema entra en la fase de finalización. En esta fase, el participante que recibe el último *token* esperado, en este caso *Hello 99999*

(luego de 100000 mensajes), envía a su sucesor el mensaje *Exit 100000*, registra el tiempo de finalización de esa instancia y sale del sistema. El participante que recibe este *token*, envía a su sucesor el *token Exit 100001*, registra el tiempo de finalización y abandona el sistema y así sucesivamente, tal como se puede observar en la Figura 5.21.

Al finalizar el intercambio de mensajes, el *TM* genera un archivo de resultados a partir del cual se pueden realizar los análisis estadísticos.

5.5.3. Experimentos y Resultados

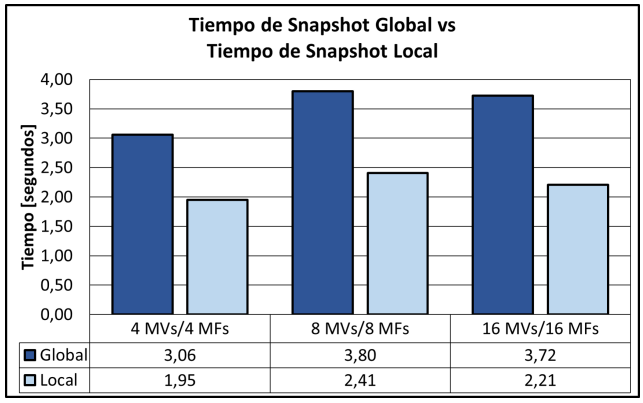
Nosotros reportamos los resultados usando el promedio obtenido después de realizar cinco ejecuciones por cada experimento, ya que los resultados pueden variar entre una ejecución y otra.

Nosotros consideramos varios escenarios de prueba usando hasta 4 MVs por cada MF y ejecutamos 4, 8 y 16 MVs. Nosotros estamos interesados en determinar la relación entre el promedio del tiempo de *snapshot* global y local cuando estamos corriendo diferentes combinaciones de cantidad de MVs y la relación del número de MVs por cada MF.

Tiempo de *snapshot* global vs tiempo de *snapshot* local: Una máquina virtual por cada máquina física

Para comparar el tiempo de *snapshot* global y el tiempo de *snapshot* local, nosotros primero ejecutamos el sistema distribuido en 4, 8 y 16 MVs, cada una corriendo en una MF.

Figura 5.22: Tiempo de *snapshot* global vs Tiempo de *snapshot* local: 1 MV por cada MF

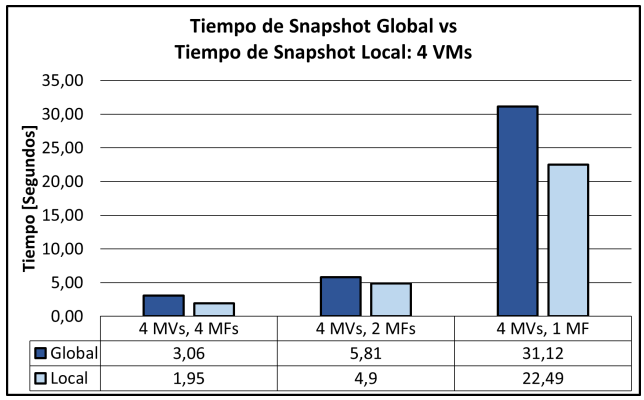


Los resultados mostrados en la Figura 5.22 muestran un comportamiento similar en los tres casos. El tiempo de *snapshot* global obtenido es entre 3.0 y 3.8 segundos mientras que el tiempo de *snapshot* local varía entre 1.9 y 2.4 segundos. Además, el incremento en el número MVs no representa un incremento considerable en los tiempos obtenidos.

Tiempo de *snapshot* global vs tiempo de *snapshot* local: Múltiples máquinas virtuales por cada máquina física

Para analizar el impacto de correr nuestra solución con múltiples MVs corriendo al mismo tiempo en una MF, nosotros probamos varias combinaciones de correr 4, 8 y 16 MVs en 4, 8 y 16 MFs.

Figura 5.23: Tiempo de *snapshot* global vs Tiempo de *snapshot* local: 4 VMs



La Figura 5.23 muestra el tiempo de *snapshot* global y el tiempo de *snapshot* local para la ejecución de 4 MVs en 4, 2 y 1 MFs, respectivamente. Analizando el comportamiento de 4 MVs en 4 y 2 MFs, de acuerdo con la Figura 5.23, el *snapshot* local pasa de 1.95 a 4.9 y el snapshot global pasa de 3.06 a 5.81. Sin embargo, cuando estamos corriendo 4 MVs en una sola MF, el desempeño cae significativamente, ya que obtener un *snapshot* local se demora 22.49 segundos y un *snapshot* global un poco más de 31 segundos.

Los tiempos de ejecución de nuestra solución para 8 MVs corriendo en 8 y 4 MFs son mostrados en la Figura 5.24.

En las pruebas realizadas con 8 MVs usando 8 y 4 MFs, se puede notar que en promedio, obtener un *snapshot* local usando 1 VM por cada MF tardó 2.41 segundos mientras que la misma operación al usar 2 MVs por cada MF el *snapshot* local demoró en promedio 4.73 segundos. En cuanto a los *snapshots* globales, el tiempo promedio pasó de 3.8 usando 1 MV por cada MF a 6.41 cuando la proporción entre MVs y MFs es de 2. Esto quiere decir que aunque hay un incremento en los tiempos, no es muy significativo.

El tiempo de *snapshot* global y el tiempo de *snapshot* local de 16 MVs usando 4, 8 y 16 MFs es mostrado en la Figura 5.25.

Figura 5.24: Tiempo de *snapshot* global vs Tiempo de *snapshot* local: 8 VMs

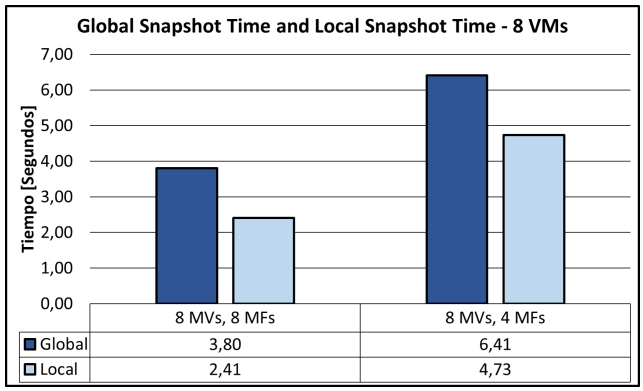
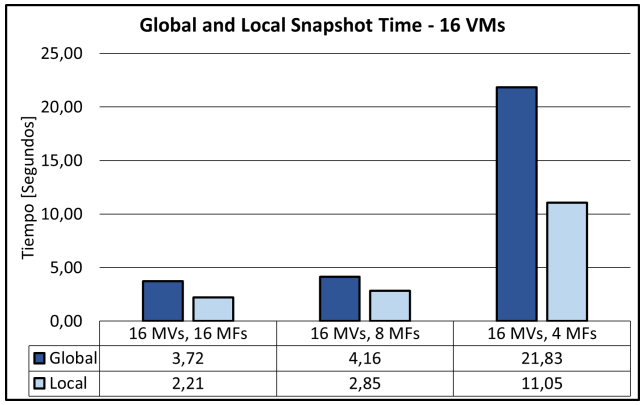


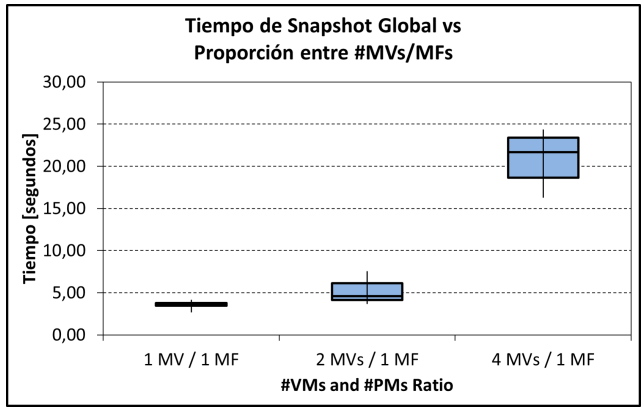
Figura 5.25: Tiempo de *snapshot* global vs Tiempo de *snapshot* local: 16 VMs



De manera similar a la ejecución de 8 MVs con una proporción de 1 o 2 MVs por cada MF, al ejecutar 16 MVs, no hay una diferencia significativa en los tiempos de *snapshot* local y global. Hay una diferencia notoria al obtener un *snapshot* local cuando la proporción es de 4 MVs por cada MF, donde el tiempo es más del doble. Y la diferencia es más significativa todavía en el tiempo de *snapshot* global la cual es incrementada más de cuatro veces.

La Figura 5.26 muestra la variabilidad del tiempo de *snapshot* global y la proporción entre el número de MVs por cada MF.

Figura 5.26: Tiempo de *snapshot* global vs Proporción entre máquinas virtuales y máquinas físicas



Los resultados de la Figura 5.26 muestran una alta consistencia en los datos, ya que se puede apreciar una baja variación con respecto al valor promedio. Además, nosotros observamos una variación significativa en el tiempo de *snapshot* global cuando se aplica el protocolo a 4 MVs corriendo al mismo tiempo en la misma MF. Este tiempo es debido principalmente a la competencia por el uso del disco cuando se almacena cada uno de los *snapshots* de las MVs. Sin embargo, no hay una diferencia importante si tenemos 1 o 2 MVs corriendo al mismo tiempo en la misma MF.

De acuerdo con lo anterior, para mitigar los efectos de la interrupción en la ejecución de MVs, es importante limitar a 1 o 2 MVs corriendo en el mismo computador al usar *snapshots* globales.

5.6. Reanudar la ejecución de una máquina virtual

Cuando se obtiene un *snapshot* global de un sistema distribuido que corre en MVs, a cada una de ellas se le toma su correspondiente *snapshot* local, invocando la función proporcionada por el hipervisor para este propósito. Como resultado, múltiples archivos pueden ser generados, los cuales almacenan el estado de la ejecución de la MV en ese momento.

De acuerdo con el análisis de fallas presentado en el Capítulo 3, una estrategia de mitigación sugerida para enfrentar las interrupciones que se pueden presentar en la ejecución de las MVs consiste en almacenar el estado del sistema y reanudar la ejecución, la cual puede realizarse, bien sea en la misma MF o en otra, dependiendo de las circunstancias de cada caso. Decidir acerca de la restauración de una MV a partir de un *snapshot* en la misma MF o en otra

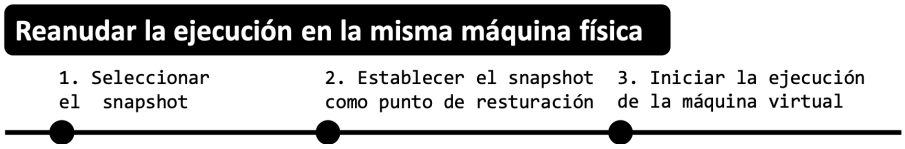
no es un asunto trivial. A continuación presentamos nuestro análisis, el cual puede servir para tomar esa decisión en el contexto de un DC.

5.6.1. Reanudar en la misma máquina física

Reanudar la ejecución de una MV en el mismo computador es la primera opción, ya que es mucho más sencillo. Los archivos que conforman la MV permanecen en la carpeta en la cual fueron creados cuando se generaron los *snapshots*, y es la ubicación en la que el hipervisor los va a buscar al reanudar la ejecución.

La Figura 5.27 muestra el proceso que se debe seguir para reanudar la ejecución de una MV en la misma MF.

Figura 5.27: Proceso para reanudar la ejecución de una MV en la misma MF.



El proceso es una secuencia de tres pasos que debe hacerse de manera coordinada en todas las MVs del sistema al que se le ha tomado uno o varios *snapshots* globales: El primer paso consiste en seleccionar el *snapshot* a partir del cual se desea reanudar la ejecución de la MV. Por lo regular, se reanuda la ejecución a partir del último *snapshot*, aunque se puede especificar uno diferente, si es necesario. El segundo paso consiste en dar la orden al hipervisor para que establezca el *snapshot* seleccionado como punto de restauración. El tercer paso es iniciar la MV.

5.6.2. Reanudar en otra máquina física

Reanudar la ejecución de una MV en otra MF es más complejo. Los archivos que conforman todas las MVs deben ser guardados en un sistema de almacenamiento externo de manera preventiva a medida que se van creando las MVs y se van obteniendo los *snapshots*. Es necesario almacenar los archivos de todas las MVs, dado que no es posible predecir cuál o cuáles MVs van a ser interrumpidas. En un sistema distribuido que corre en MVs, el número total de archivos que se generan a medida que se van tomando *snapshots* globales crece, al punto que puede llegar a ser significativo. Por lo tanto, es necesario gestionarlos adecuadamente. La ecuación 5.4, que aplica para *Oracle VirtualBox* permite calcular el número total de archivos por cada MV, donde s es el número de *snapshots* de la MV y d es el número de discos conectados, suponiendo que la

MV está en ejecución cuando los *snapshots* son tomados.

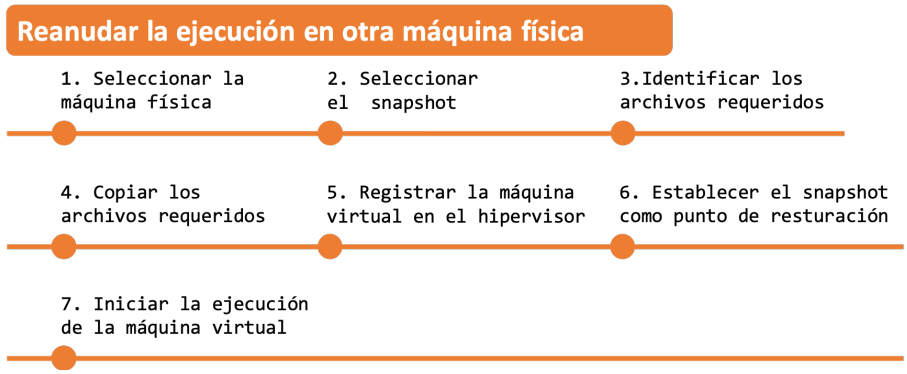
$$\#\ de\ archivos = \begin{cases} d+1 & \text{si, } s=0 \\ (d+1)\times(s+1) & \text{si, } s\geq 1 \end{cases} \tag{5.4}$$

Si un sistema distribuido se ejecuta por un largo período de tiempo, y múltiples *snapshots* globales son tomados (periódicamente o no), el número de archivos generados está dado por la ecuación 5.5, donde n es el número de MVs, d es el número de discos conectados a cada MV y s es el número de *snapshots* tomados a cada MV.

$$\#\ de\ archivos = \begin{cases} (d+1)\times n & \text{si, } s=0 \\ (d+1)\times(s+1)\times n & \text{si, } s\geq 1 \end{cases} \tag{5.5}$$

Por ejemplo, suponga que hay un sistema distribuido formado por 8 MVs, cada una con 1 disco virtual conectado. Luego de haber obtenido 4 *snapshots* globales, se han generado $(d{+}1){\times}(s{+}1){\times}n = (1{+}1){\times}(4{+}1){\times}8 = 2{\times}5\ {\times}8 = 80$ archivos. Es importante tener en cuenta que los hipervisores generan otros archivos, no esenciales, que no han sido considerados.

Figura 5.28: Proceso para reanudar la ejecución de una MV en otra MF.



La Figura 5.28 muestra el proceso que se debe seguir para reanudar la ejecución de una MV en otra MF. El proceso está formado por siete pasos, donde los dos últimos son los mismos que en el proceso de reanudación de la ejecución en la misma MF. Este proceso debe seguirse para cada MV que vaya a continuar su ejecución en otro computador. Eso quiere decir que al reanudar la ejecución de un sistema distribuido, a partir de un *snapshot* es posible que una o varias MVs tengan que ser ejecutadas en MFs diferentes. En este caso, se debe coordinar en todas las MVs del sistema, la realización de los dos últimos pasos del proceso.

El paso 1 se refiere a la selección de la MF que será empleada para reanudar la ejecución de la MV. La MF seleccionada deberá estar en condiciones de recibir los archivos que corresponden a la MV para ponerla en ejecución. El

paso 2 consiste en seleccionar el *snapshot* a partir del cual se desea reanudar la ejecución de la MV. Al igual que en la sección anterior, se puede especificar el *snapshot*, o reanudar a partir del último.

En el paso 3 se deben identificar los archivos requeridos. De acuerdo el trabajo publicado en [21], encontramos que para restaurar la MV a partir de un *snapshot* se pueden ignorar algunos archivos. Por lo tanto, para hacer un uso eficiente de los recursos, es necesario encontrar el conjunto mínimo de archivos para, de este modo, ocupar el menor espacio en disco, transmitir la menor cantidad de información por la red y hacerlo en el menor tiempo posible. En este trabajo analizamos el hipervisor *Oracle VirtualBox* como caso de estudio. Por lo tanto, para poder reanudar la ejecución de la VM en otra MF, es necesario contar con el archivo *.vbox*, el archivo *.vdi* de ese *snapshot* junto con todos los archivos diferenciales anteriores (sus ancestros), para cada disco conectado a la MV. Además, el archivo *.sav* que almacena el estado de la memoria en el *snapshot* correspondiente.

Como lo señalamos en la sección 5.1.3 de la página 58, el manejo de los archivos producto de los *snapshots* es una tarea muy difícil para ser realizada manualmente. Además, a medida que se toman *snapshots* a una MV, el número de archivos aumenta y cada vez es más compleja la administración. Con este propósito, nosotros desarrollamos una herramienta que facilita la interpretación de los archivos que se crean al tomar *snapshots* de MVs basado en *Oracle VirtualBox* [21]. Además, la herramienta desarrollada determina el conjunto mínimo de archivos que se requieren para continuar con la ejecución de una MV en otra MF a partir de un *snapshot*.

El paso 4 consiste en copiar los archivos requeridos en la MF seleccionada en el paso 1, conservando la misma estructura de carpetas que tenía en la ubicación original. El paso 5 corresponde al registro de la MV en el hipervisor de la MF seleccionada. El proceso termina con los pasos 6 y 7, en los cuales se le indica al hipervisor cuál es el *snapshot* a partir del cual se desea reanudar la ejecución, e iniciar la ejecución de la MV, tal como en el caso anterior.

5.6.3. Dónde reanudar la ejecución de una máquina virtual a partir de un *snapshot*

De acuerdo con lo planteado en las secciones 5.6.1 y 5.6.2, reanudar la ejecución de una MV a partir de un *snapshot* es un proceso mucho más sencillo y rápido si utilizamos la misma MF en la cual se tomaron los *snapshots*.

Vamos a analizar el costo que representa ofrecer una solución de *snapshot* global en la cual se puede recuperar la ejecución de una MV en un MF diferente a la que se utilizó para su ejecución inicial. Vamos a considerar el espacio en disco que ocupa, la cantidad de información que se debe enviar por la red y el tiempo necesario para que el sistema pueda continuar, a partir de un *snapshot* global.

Suponga que tenemos un sistema distribuido corriendo en 8 MVs. Cada MV tiene un disco duro virtual conectado y se han obtenido 4 *snapshots* globales.

Suponga además, que los tamaños de los archivos (10 de acuerdo con la ecuación 5.4) son los que aparecen en la Tabla 5.2.

Tabla 5.2: Archivos de una máquina virtual luego de 4 *snapshots*

Archivo	Espacio en disco
.vbox	10 KB
.vdi principal (snapshot 1)	3.51 GB
.sav (snapshot 1)	0.250 GB
.vdi (snapshot 2)	0.125 GB
.sav (snapshot 2)	0.250 GB
.vdi (snapshot 3)	0.125 GB
.sav (snapshot 3)	0.250 GB
.vdi (snapshot 4)	10.125 GB
.sav (snapshot 4)	0.250 GB
.vdi (estado actual)	0.125 GB
Total	5.01 GB

Como se puede apreciar en la Tabla 5.2, todos los archivos de la MV ocupan 5.01 GB. Sin embargo, de acuerdo con [21], algunos de estos archivos no son requeridos al momento de reanudar la ejecución en otro computador. En nuestro ejemplo, los archivos requeridos se pueden ver en la Tabla 5.3.

Tabla 5.3: Archivos requeridos para reanudar la ejecución de una máquina virtual con 4 *snapshots* en otra máquina física — Modelo actual de aprovisionamiento de máquinas virtuales

Archivo	Espacio en disco
.vbox	10 KB
.vdi principal (snapshot 1)	3.51 GB
.vdi (snapshot 2)	0.125 GB
.vdi (snapshot 3)	0.125 GB
.vdi (snapshot 4)	0.125 GB
.sav (snapshot 4)	0.250 GB
Total	4.14 GB

En este caso, los archivos a descartar (los archivos *.sav* de *snapshots* anteriores y el *.vdi* del estado actual de la máquina virtual) ocupan 0.88 GB, siendo necesario transferir por la red archivos que ocupan 4.14 GB, para una reducción de aproximadamente el 18 %, información que no se tiene que transmitir ni almacenar.

Teniendo en cuenta que cualquiera de las MFs empleadas para la ejecución de un sistema distribuido puede fallar en cualquier momento, y que no es posible

saber con anticipación cuál es la MF que puede llegar a fallar, sería necesario contar con un sistema de almacenamiento externo para mantener una copia de todos los archivos y garantizar que se puede recuperar en caso de falla en alguno de los computadores. Considerando el ejemplo del sistema conformado por 8 MVs con 4 *snapshots* globales, sería necesario contar con un espacio de $8 \times 5{,}01\ GB = 40{,}08\ GB$ en un sistema de almacenamiento externo, los cuales deben ser transmitidos por la red. Además, al reanudar la ejecución, los archivos requeridos deben ser transmitidos del sistema externo al computador donde va a ser ejecutada la MV. El espacio en disco total para ofrecer una solución de *snapshot* global para el ejemplo que estamos analizando es de En total, $40{,}08\ GB + 4{,}14\ GB = 44{,}22\ GB$, la misma cantidad de espacio que debe ser transmitida por la red.

Sin embargo, al usar el nuevo modelo de aprovisionamiento de máquinas virtuales, no es necesario copiar el archivo con el disco principal en la MF destino, ya que estaría precargado. Esto representa una reducción significativa, ya que se trata del archivo más grande, en orden de los GB. Por lo tanto, de acuerdo con la Tabla 5.4, solo sería necesario transmitir 0.63 GB, aproximadamente un 87 % con respecto al tamaño original de la MV (5.01 GB).

Tabla 5.4: Archivos requeridos para reanudar la ejecución de una máquina virtual con 4 *snapshots* en otra máquina física — Nuevo modelo de aprovisionamiento de máquinas virtuales

Archivo	Espacio en disco
.vbox	10 KB
.vdi (snapshot 2)	125 MB
.vdi (snapshot 3)	125 MB
.vdi (snapshot 4)	125 MB
.sav (snapshot 4)	0.250 GB
Total	0.63 GB

Por otra parte, en un informe elaborado por la Administración de los Laboratorios del Departamento de Ingeniería de Sistemas y Computación de la Universidad de los Andes, manifestaron lo siguiente:

- Los computadores de las salas donde se ejecutan las MVs gestionadas por *UnaCloud* permanecen en garantía, dado que son computadores que se renuevan antes de su vencimiento.

- En los últimos cinco años se han realizado 11 reclamaciones por garantía a los proveedores, lo que equivale a menos del 2 % por computador al año.

Con esos resultados, podemos confirmar que las fallas que ocurren en un desktop cloud son fundamentalmente ocasionadas por la presencia del usuario

del computador al mismo tiempo que se ejecutan las máquinas virtuales administradas por el desktop cloud. Esto quiere decir que, por una parte, reanudar la ejecución de un sistema distribuido corriendo en las MVs de una DC, luego de una interrupción, puede realizarse a partir del útlimo *snapshot* global usando las mismas máquinas físicas, sin necesidad de asumir el costo de almacenamiento y transferencia que requiere un sistema de almacenamiento externo.

Capítulo 6

Conclusiones y trabajo futuro

Hemos desarrollado esta tesis doctoral con el fin de incorporar mecanismos de confiabilidad que permitan extender los servicios prestados por un sistema *desktop cloud* ofreciendo alguna garantía de terminación a la ejecución de sistemas distribuidos formados por aplicaciones que se comunican entre ellas.

Hemos estado haciendo y refinando un análisis de fallas en sistemas *desktop cloud* considerando *UnaCloud* y otros sistemas similares. Hemos propuesto una extensión a la cadena de amenazas usada en la literatura. Usando esta cadena extendida pudimos identificar las causas y consecuencias de las fallas, el actor o usuario causante y un conjunto inicial de estrategias de mitigación, algunas de ellas utilizadas en distintos sistemas *desktop cloud* y similares para contrarrestar los efectos de las anomalías e interrupciones para mejorar su confiabilidad.

Gracias a este análisis, encontramos que las fallas que afectan la confiabilidad de los sistemas *desktop cloud* ocurren principalmente en dos momentos: en el aprovisionamiento y en la ejecución de máquinas virtuales. El primero tiene limitaciones significativas de escalabilidad. Las imágenes usadas para la creación de máquinas virtuales son archivos grandes cuya transmisión es demorada y propensa a fallas. En el segundo, la presencia de los usuarios de las máquinas físicas al mismo tiempo que las máquinas virtuales se están ejecutando es crítica, debido a que pueden apagar o reiniciar el computador, desconectarlo de la red, o ejecutar aplicaciones exigentes en capacidad computacional, interrumpiendo la ejecución de las máquinas virtuales. Como resultado, el usuario del *desktop cloud* puede perder todo el trabajo realizado hasta el momento de la interrupción. Debido al tipo de aplicaciones que se pueden ejecutar en un *desktop cloud*, las interrupciones no solo afectan las máquinas virtuales en ejecución en los computadores comprometidos, sino que pueden afectar todo el sistema. Por ejemplo, en aplicaciones que tienen procesos que se comunican entre ellos, si uno deja de responder, puede arruinar todo el trabajo realizado.

Producto del análisis de fallas, de la revisión de la literatura relacionada y de la experiencia de casi diez años desarrollando, utilizando, evaluando y analizando *UnaCloud*, recopilamos las siguientes estrategias de mitigación que

pueden ser utilizadas para mejorar la confiabilidad, tanto en el aprovisionamiento como en la ejecución de máquinas virtuales:

1. Usar protocolos eficientes de transmisión de archivos
2. Usar mecanismos para reducir el tamaño de los archivos a transmitir
3. Usar mecanismos para reducir la necesidad de transmitir archivos
4. Usar mecanismos para gestionar eficientemente el espacio en disco
5. Usar algoritmos de ubicación que consideren el espacio en disco
6. Usar plantillas de imágenes virtuales preconfiguradas
7. Reiniciar la ejecución del sistema que corre en las máquinas virtuales
8. Guardar el estado del sistema que corre en las máquinas virtuales y reanudar la ejecución

De este conjunto de estrategias, implementamos las más relevantes, en el contexto de *UnaCloud*. Para la fase de aprovisionamiento seleccionamos las siguientes tres estrategias:

Usar plantillas de imágenes virtuales preconfiguradas. Ofrecer a los usuarios del *desktop cloud* imágenes virtuales preconfiguradas, creadas por usuarios expertos, las cuales superen las pruebas de calidad para asegurar que cumplen las condiciones establecidas por el *desktop cloud*.

Usar mecanismos para gestionar eficientemente el espacio en disco. Utilizar discos virtuales en modo de escritura *multiatatch* para guardar las imágenes virtuales preconfiguradas. De este modo, múltiples máquinas virtuales en la misma máquina física puedan ser conectadas al mismo disco virtual y ejecutadas al mismo tiempo.

Usar mecanismos para reducir la necesidad de transmitir archivos. Precargar, es decir, copiar de manera anticipada, los archivos con las imágenes virtuales en los discos duros de las máquinas físicas que hacen parte del *desktop cloud*. La copia de los archivos se puede realizar en el período de alistamiento de las salas de cómputo, durante el mantenimiento o en momentos de poco uso.

En la implementación de estas tres estrategias de mitigación desarrollamos el sistema *Multiattach Manager*, dado que el hipervisor no ofrece ninguna funcionalidad para aprovechar el modo de escritura *multiattach*. Los componentes desarrollados son los siguientes:

1. Precargar disco *multiattach*. Este componente se encarga de registrar el disco *multiattach* en el administrador de medios virtuales del hipervisor.
2. Crear una máquina virtual y conectarle un disco *multiattach*.
3. Clonar una máquina virtual conectada a un disco *multiattach*.

4. Replicar una máquina virtual conectada a un disco *multiattach* para correr en otra máquina física.

5. Migrar una máquina virtual conectada a un disco *multiattach* de una máquina física a otra.

6. Eliminar una máquina virtual conectada a un disco *multiattach*.

7. Eliminar el registro de un disco *multiattach* del administrador de medios virtuales del hipervisor.

Para la fase de ejecución de las máquinas virtuales seleccionamos la siguientes dos estrategias:

Guardar el estado del sistema que corre en las máquinas virtuales y reanudar la ejecución. En este caso utilizamos *checkpointing-restart* como mecanismo para enfrentar las interrupciones que se presentan en la ejecución de las máquinas virtuales. Hacemos referencia al protocolo de *snapshot* global que permite almacenar el estado, en un momento dado, de un sistema distribuido que corre en un conjunto de máquinas virtuales. Un *snapshot* global almacena en archivos los estados de las máquinas virtuales que hacen parte del sistema de manera que sea posible volver a ese estado y reanudar la ejecución de manera consistente, sin mensajes perdidos ni duplicados, y sin modificar la aplicación ni el hipervisor. Este desarrollo evita que se pierda el trabajo adelantado hasta el momento en que se toma el *snapshot* global.

En la implementación del protocolo de *snapshot* global desarrollamos el sistema *Global Snapshot* compuesto por dos componentes: (1) *GSLauncher*, encargado de lanzar las instancias de la aplicación en cada máquina física donde corre una máquina virtual que hace parte de un sistema distribuido, y (2) *MetadataServer*, un componente encargado de manejar la estructura de los nodos del sistema.

Se puede reanudar la ejecución de una máquina virtual a partir de un *snapshot*, utilizando la misma máquina física u otra. En la misma máquina física, la recuperación es trivial, dado que los archivos necesarios están almacenados donde el hipervisor los creó. Sin embargo, para reanudar en otra máquina física, es necesario poner a disposición del hipervisor los archivos correspondientes a la máquina virtual. Dado que no se necesitan todos los archivos que forman una máquina virtual para reanudar su ejecución, desarrollamos un componente encargado de analizar los archivos que se generan al obtener múltiples *snapshots* en cada máquina virtual. Este sistema, denominado *Global Snapshot File Tracker*, determina el conjunto de archivos requeridos. Al utilizar esta herramienta, podemos manejar de manera eficiente el espacio en disco y la red, evitando la transferencia de los archivos que no se necesitan.

Adicionalmente, desarrollamos *Hypervisor Remote Control*, un sistema de administración remota de máquinas virtuales flexible y fácil de usar. Este sistema se utiliza desde la línea de comandos y se puede incluir dentro de *scripts* para agilizar la realización de pruebas y facilitar la repetición en caso de ser necesario.

6.1. Lecciones aprendidas

El monitoreo es esencial y complejo. Como resultado del análisis de fallas se identificaron las anomalías e interrupciones que dan origen a las fallas en el aprovisionamiento y la ejecución de máquinas virtuales. Es esencial contar con un sistema de monitoreo que registre la ocurrencia de esas anomalías e interrupciones para tener información oportuna y precisa del comportamiento de las máquinas físicas y sus usuarios. Desafortunadamente, no tenemos un sistema de monitoreo en el que podamos confiar.

La red y el almacenamiento son cuellos de botella. Las salas de cómputo de las universidades se renuevan con alguna frecuencia. Generalmente los nuevos computadores aumentan la cantidad de memoria RAM, el número de núcleos y la velocidad de procesamiento. Sin embargo, con el disco duro y la red no sucede lo mismo. Los computadores nuevos siguen llegando con un solo disco duro y la velocidad de la red está en los mismos niveles de servicio desde hace unos años.

El análisis de fallas es un proceso iterativo. Todos los sistemas computacionales evolucionan con el tiempo. El análisis de fallas mediante la cadena extendida de amenazas no termina con una solución determinista. Diferentes personas pueden hacer un análisis al mismo tiempo y llegar a diferentes resultados. Asimismo, si el análisis es realizado en un tiempo diferente, el resultado puede cambiar. En nuestro caso, hemos estado haciendo y refinando un análisis de fallas en sistemas *desktop cloud* haciendo énfasis en *UnaCloud*. Sabíamos que *UnaCloud* es una solución frágil que falla con frecuencia, pero no por qué. En un principio, encontramos problemas en la ejecución de las máquinas virtuales. Sin embargo, con el tiempo encontramos que teníamos muchos errores cuando intentábamos aprovisionar cierta cantidad de máquinas virtuales y, por lo tanto, no alcanzaban a llegar a la ejecución.

Pasar del entorno de desarrollo al entorno de producción es complejo. El entorno de pruebas que teníamos a disposición contaba, en apariencia, con condiciones similares al entorno de producción en el que se ejecutan las máquinas virtuales administradas por el *desktop cloud*. Sin embargo, al realizar pruebas en el entorno de producción, encontramos restricciones de seguridad que obstaculizaron la realización de estas pruebas y aumentaron la complejidad del desarrollo de los componentes de software de nuestra solución.

6.2. Acerca de *UnaCloud* como caso de estudio

UnaCloud es un sistema que se caracteriza por prestar un servicio de "mejor esfuerzo", es decir, un servicio donde no hay garantías de que las máquinas virtuales van a finalizar su ejecución de manera exitosa.

Desde un principio, utilizar *UnaCloud* fue muy positivo. Contar con un sistema funcional, nos permitió identificar sus fallas más fácilmente. Al compararlo con otros sistemas similares, pudimos contrastar sus características y

refinar nuestro análisis de fallas, el cual arrojó importantes resultados que fueron tenidos en cuenta en la implementación de soluciones.

Al inicio, concentramos los esfuerzos por resolver el problema de la interrupción de la ejecución de las máquinas virtuales. Con el tiempo nos dimos cuenta de la importancia de enfrentar los problemas de aprovisionamiento.

La falta de un sistema de monitoreo que nos ofreciera información acerca de las máquinas físicas y las máquinas virtuales es una limitación enorme, no solo para el desarrollo de esta tesis doctoral, sino para la operación del sistema.

El acceso oportunista a los recursos de las salas de cómputo, sin los privilegios suficientes, por ejemplo, para copiar archivos o para instalar nuestras aplicaciones fue un obstáculo que retrasó el desarrollo del trabajo. Sin embargo, eso también dio origen a nuevos desarrollos que al final nos obligó a tener una solución más completa, la cual sin duda mejorará las condiciones de la nueva versión de *UnaCloud*, y podremos ofrecer servicios confiables en nuestra plataforma.

6.3. Acerca de la solución

Las implementaciones realizadas, bajo el supuesto de que se pueden usar los discos en modo de escritura *multiattach* ofrecen una solución eficiente en el uso del espacio físico y libera importantes cantidades de bytes que tendrían que ser enviados por la red si se fuera a utilizar la solución actual basada en discos normales.

El *snapshot* global ha sido probado y los resultados permiten pensar que es una solución para almacenar el estado de las máquinas virtuales de manera preventiva para que, en caso de una interrupción, el sistema pueda ser recuperado en el estado anterior y, de este modo, evitar que el trabajo adelantado se pierda.

Además, teniendo en cuenta que el hardware tiene una probabilidad de falla muy pequeña, podemos usar un sistema de pausa y siga, reanudando la ejecución en la misma máquina física donde se estaba ejecutando una máquina virtual, en lugar de migrar, lo cual es más demorado.

6.4. Impacto del trabajo realizado

El trabajo realizado establece cambios significativos en *UnaCloud* ya que, por una parte, fue nuestro caso de estudio y fuente de inspiración. Por otra parte, tendremos una nueva versión que tendrá en cuenta todas las ideas contempladas en esta tesis doctoral. Los problemas de aprovisionamiento que tuvimos en el pasado, quedarán resueltos y podremos asegurar despliegues exitosos de conjuntos significativos de máquinas virtuales. Además, tendremos un mecanismo de *snapshot* global incorporado a *UnaCloud* que nos permitirá almacenar el estado de un sistema distribuido que se ejecuta en máquinas virtuales para mejorar la confiabilidad de *UnaCloud*.

Las herramientas desarrolladas para gestionar los discos virtuales en modo de escritura *multiattach* pueden ser utilizados por fuera de un *desktop cloud*. El software desarrollado permite obtener máquinas virtuales completamente operativas en segundos con aplicaciones de uso común fácilmente aplicables en colegios y universidades.

La implementación del sistema de *snapshot* global puede ser utilizada directamente en cualquier *desktop cloud* si el hipervisor es *Oracle VirtualBox*. Sin embargo, si el hipervisor es diferente, algunas modificaciones deben ser realizadas, siempre y cuando el hipervisor proporcione la interfaz de comandos equivalente, o soporte la tecnología utilizada. Por ejemplo, la ejecución de comandos en una máquina virtual, o los discos *multiattach*.

6.5. Trabajo futuro

En esta sección presentamos algunas ideas de trabajo futuro con respecto a las contribuciones de esta tesis doctoral.

Ciclo de vida para las imágenes virtuales. Para que el nuevo modelo de aprovisionamiento haga parte de la solución de un sistema *desktop cloud*, es necesario establecer un ciclo de vida para las imágenes virtuales, incluyendo procesos claramente definidos para su creación y mantenimiento de imágenes virtuales. Además, se requiere un sistema de control de versiones para garantizar la compatibilidad de las máquinas virtuales que se crean a partir de esas imágenes.

Sistema de monitoreo. *UnaCloud* necesita el desarrollo de un sistema de monitoreo confiable, que haga parte integral de su tablero de control, el cual, además de registrar métricas básicas acerca de cada máquina física, registre los eventos de ocurrencia de todas las funciones que ofrece el sistema, así como la ocurrencia de las anomalías, interrupciones, errores y fallas. El sistema de monitoreo debe aportar información precisa y oportuna que pueda ser utilizada en la toma de decisiones, tanto en el aprovisionamiento como de ejecución de máquinas virtuales.

Nuevos algoritmos de ubicación. Es necesario proponer y desarrollar nuevos algoritmos de ubicación de máquinas virtuales que mejoren la confiabilidad de las plataformas *desktop cloud*. Por ejemplo, utilizar la cantidad de espacio disponible en el disco antes de hacer la selección de la máquina física que tendrá alojada la máquina virtual.

Nueva versión de *UnaCloud*. Dentro del análisis realizado en esta tesis hemos identificado oportunidades de mejora para *UnaCloud*, nuestra implementación de *desktop cloud*. Hemos propuesto algunas estrategias e implementado algunos componentes de software los cuales, sin duda, van a ser parte fundamental en nuestro objetivo de extender las capacidades de servicio que prestan esta clase de sistemas. Sin embargo, para la nueva versión de *UnaCloud* está todo por hacer, incluyendo la integración del nuevo modelo de aprovisionamiento, el protocolo de *snapshot* global, el ciclo de vida para las imágenes virtuales, el sistema de monitoreo y nuevos algoritmos de ubicación.

Experimentos para medir la escalabilidad. Las pruebas realizadas en el marco de esta tesis doctoral mostraron resultados prometedores en la escalabilidad del aprovisionamiento de máquinas virtuales. En cuanto al protocolo de *snapshot* global, todavía hace falta realizar experimentos de mayor envergadura, para determinar su capacidad de escalabilidad.

Generalización de la solución. Las pruebas de concepto, el protocolo de *snapshot* global, el sistema de control remoto y las herramientas desarrolladas para gestionar los discos en modo de escritura *multiattach* y las máquinas virtuales que los utilizan, fueron realizados con base en el hipervisor *Oracle VirtualBox* y el sistema operativo Windows en las máquinas físicas, debido a que son las herramientas utilizadas en la versión actual de nuestro *desktop cloud*. Los componentes de software mencionados pueden ser extendidos a otros hipervisores como *VMware Workstation* a través de la API VIX, y *KVM* a través de la línea de comandos.

Tecnologías emergentes. Como fue mencionado en la sección 1.1, los *desktop clouds* ofrecen un subconjunto de servicios de infraestructura (IaaS), permitiendo a sus usuarios el despliegue de máquinas virtuales sobre una infraestructura oportunista para ejecutar aplicaciones. Esta tesis doctoral está enfocada en la incorporación de mecanismos de confiabilidad y tolerancia a fallos para sistemas *desktop cloud*, con el fin de extender las capacidades de servicio que prestan estos sistemas. En este contexto, se pueden utilizar máquinas virtuales para ejecutar aplicaciones de cualquier tipo, incluyendo la infraestructura necesaria para prestar servicios de computación en la nube, tales como contenedores, la computación sin servidor (*serverless*) o la distribución de aplicaciones en streaming. Sin embargo, para dar un paso hacia adelante hacia nuevos servicios, por ejemplo, plataforma como servicio, o los servicios administrados tales como contenedores y computación sin servidor, es necesario que un *desktop cloud* sea confiable en el aprovisionamiento y la ejecución de máquinas virtuales.

Bibliografía

[1] M. W. Mutka and M. Livny, *"Profiling Workstation's Available Capacity for Remote Execution"*. University of Wisconsin-Madison. Computer Sciences Department, 1987.

[2] K. D. Ryu, *"Exploiting Idle Cycles in Networks of Workstations"*. PhD thesis, University of Maryland, College Park, 2001.

[3] P. Domingues, P. Marques, and L. Silva, "*Resource Usage of Windows Computer Laboratories*," in *Parallel Processing, 2005. ICPP 2005 Workshops. International Conference Workshops on*, pp. 469–476, IEEE, 2005.

[4] C. E. Gómez, C. O. Díaz, C. A. Forero, E. Rosales, and H. Castro, "*Determining the Real Capacity of a Desktop Cloud*," in *Latin American High Performance Computing Conference*, pp. 62–72, Springer, 2015.

[5] A. A. Oviedo, "*UnaCloud MSA: Plataforma Basada en UnaCloud para la Generación y Análisis de Alineamientos Múltiples de Secuencias*," Master's thesis, Universidad de los Andes, 2011.

[6] N. Gómez and J. G. Tamura, "*Detección y Predicción de Fallas a partir de Métricas de Performance en Sistemas Desktop Cloud*," Master's thesis, Universidad de los Andes, 2018.

[7] L. F. G. Sarmenta, *"Volunteer computing"*. PhD thesis, Massachusetts Institute of Technology, 2001.

[8] D. Kondo, *"Scheduling Task Parallel Applications for Rapid Turnaround on Desktop Grids"*. PhD thesis, University of California, San Diego, 2005.

[9] A. Alwabel, R. J. Walters, and G. B. Wills, "*A View at Desktop Clouds*," in *International Workshop on Emerging Software as a Service and Analytics (ESaaSA 2014)*, pp. 55–61, 2014.

[10] E. Rosales, H. Castro, and M. Villamizar, "*UnaCloud: Opportunistic Cloud Computing Infrastructure as a Service*," *Cloud Computing*, pp. 187–194, 2011.

[11] H. E. Castro, E. E. Rosales, and M. J. Villamizar, "*Desktop Grids and Volunteer Computing Systems*," in *Computational and Data Grids: Principles, Applications and Design*, pp. 1–30, IGI Global, 2012.

[12] G. A. McGilvary, *"Ad hoc cloud computing"*. PhD thesis, The University of Edinburgh, 2014.

[13] V. D. Cunsolo, S. Distefano, A. Puliafito, and M. Scarpa, "*Volunteer Computing and Desktop Cloud: The cloud@home Paradigm*," in *Network Computing and Applications, 2009. NCA 2009. Eighth IEEE International Symposium on*, pp. 134–139, IEEE, 2009.

[14] G. Fedak, *"Contributions to Desktop Grid Computing"*. PhD thesis, Ecole Normale Supérieure de Lyon, 2015.

[15] OpenStack, "*OpenStack*." `https://www.openstack.org`, 2018.

[16] Amazon, "*Amazon Web Services*." `https://aws.amazon.com/es/`, 2019. Online; accessed 15-July-2019.

[17] Microsoft, "*Microsoft Azure*." `https://azure.microsoft.com/es-es/`, 2018. Online; accessed 23-July-2018.

[18] Universidad de los Andes, Grupo COMIT, "*UnaCloud: Opportunistic cloud computing platform*." `https://sistemasproyectos.uniandes.edu.co/iniciativas/unacloud/`, 2018. Online; accessed 23-July-2018.

[19] CERN, "*CernVM*." `https://cernvm.cern.ch/`, 2018. Online; accessed 12-June-2019.

[20] B. Segal, C. Aguado Sanchez, P. Buncic, J. Rantala, P. Mato, J. Blomer, D. G. Quintas, D. J. Weir, Y. Yao, and A. Harutyunyan, "*LHC cloud computing with CernVM*," *PoS*, p. 004, 2010.

[21] C. E. Gómez, J. Chavarriaga, D. C. Bonilla, and H. E. Castro, "*Global Snapshot File Tracker*," in *Applied Informatics* (H. Florez, C. Diaz, and J. Chavarriaga, eds.), (Cham), pp. 90–104, Springer International Publishing, 2018.

[22] B. Nicolae and F. Cappello, "*BlobCR: Virtual Disk Based Checkpoint-Restart for HPC Applications on IaaS Clouds*," *Journal of Parallel and Distributed Computing*, vol. 73, no. 5, pp. 698–711, 2013.

[23] dos Santos, Maicon Ança and Du Bois, André Rauber and Cavalheiro, Gerson Geraldo Homrich, "*A User-Level Scheduling Framework for BoT Applications on Private Clouds*," in *Computer Architecture and High Performance Computing (SBAC-PAD), 2017 29th International Symposium on*, pp. 81–88, IEEE, 2017.

[24] A. Avižienis, "*A Visit to the Jungle of Terminology*," in *Dependable Systems and Networks Workshop (DSN-W), 2017 47th Annual IEEE/IFIP International Conference on*, pp. 149–152, IEEE, 2017.

[25] A. Kangarlou-Haghighi, *"Improving the Reliability and Performance of Virtual Cloud Infrastructures"*. PhD thesis, Purdue University, 2011.

[26] E. Bóhorquez, E. Rosales, and H. Castro, "*Running MPI Applications over an Opportunistic Infrastructure*," in *2015 Ninth International Conference on Complex, Intelligent, and Software Intensive Systems*, pp. 446–453, IEEE, 2015.

[27] A. Marosi, J. Kovács, and P. Kacsuk, "*Towards a Volunteer Cloud System*," *Future Generation Computer Systems*, vol. 29, no. 6, pp. 1442–1451, 2013.

[28] T. Desell, M. Magdon-Ismail, B. Szymanski, C. Varela, H. Newberg, and N. Cole, "*Robust Asynchronous Optimization for Volunteer Computing Grids*," in *e-Science, 2009. e-Science'09. Fifth IEEE International Conference on*, pp. 263–270, IEEE, 2009.

[29] A. Kangarlou, D. Xu, and P. Eugster, "*VNsnap: Taking Snapshots of Virtual Networked Environments with Minimal Downtime*," tech. rep., Purdue University Center for Education and Research Information Assurance and Security, 2008.

[30] Mersenne Research, Inc., "*The Great Internet Mersenne Prime Search*." `https://www.mersenne.org/`, 1996. Online; accessed 19-June-2019.

[31] J. F. Shoch and J. A. Hupp, "*The "Worm" Programs—Early Experience with a Distributed Computation*," *Communications of the ACM*, vol. 25, no. 3, pp. 172–180, 1982.

[32] M. J. Litzkow, M. Livny, and M. W. Mutka, "*Condor - A Hunter of Idle Workstations*," in *Distributed Computing Systems, 1988., 8th International Conference on*, pp. 104–111, IEEE, 1988.

[33] E. Rosales, G. Sotelo, A. de la Vega, C. O. Díaz, C. E. Gómez, and H. Castro, "*Harvesting idle CPU resources for desktop grid computing while limiting the slowdown generated to end-users*," *Cluster Computing*, vol. 18, no. 4, pp. 1331–1350, 2015.

[34] D. P. Anderson, "*BOINC: A System for Public-Resource Computing and Storage*," in *proceedings of the 5th IEEE/ACM International Workshop on Grid Computing*, pp. 4–10, IEEE Computer Society, 2004.

[35] S. Choi, R. Buyya, H. Kim, E. Byun, J. Gil, and C. Park, "*A Taxonomy of Desktop Grids and its Mapping to State-of-the-Art Systems*," *Grid Computing and Distributed Systems Laboratory, The University of Melbourne, Tech. Rep.*, 2008.

[36] M. N. Durrani and J. A. Shamsi, "*Volunteer Computing: Requirements, Challenges, and Solutions*," *Journal of Network and Computer Applications*, vol. 39, pp. 369–380, 2014.

[37] P. Mell and T. Grance, "*The NIST Definition of Cloud Computing*," Tech. Rep. SP 800-145, National Institute of Standards & Technology, Gaithersburg, MD, United States, 2011.

[38] D. K. Barry and D. Dick, "*Cloud Computing*," in *Web Services, Service-Oriented Architectures, and Cloud Computing (Second Edition)* (D. K. Barry and D. Dick, eds.), pp. 35 – 44, Morgan Kaufmann, second edition ed., 2013.

[39] A. Alwabel, R. Walters, and G. Wills, "*Towards a Volunteer Cloud Architecture*," in *European Workshop on Performance Engineering*, pp. 248–251, Springer, 2012.

[40] C. E. Gómez, H. E. Castro, and C. A. Varela, "*Global Snapshot of a Distributed System Running on Virtual Machines*," in *2017 29th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, pp. 169–176, IEEE, 2017.

[41] D. Prasad, J. McDermid, and I. Wand, "*Dependability Terminology: Similarities and Differences*," in *10th Annual Conference on Computer Assurance, COMPASS'95.*, pp. 213–221, IEEE, 1995.

[42] J.-C. Laprie, "*Dependability of Computer Systems: Concepts, Limits, Improvements*," in *Software Reliability Engineering, 1995. Proceedings., Sixth International Symposium on*, pp. 2–11, IEEE, 1995.

[43] A. Avizienis, J. C. Laprie, B. Randell, and C. Landwehr, "*Basic Concepts and Taxonomy of Dependable and Secure Computing*," *IEEE Transactions on Dependable and Secure Computing*, vol. 1, no. 1, pp. 11–33, 2004.

[44] E. Jonsson, "*An Integrated Framework for Security and Dependability*," in *The 1998 Workshop on New Security Paradigms, NSPW '98*, pp. 22–29, 1998.

[45] E. Jonsson, "*Towards an Integrated Conceptual Model of Security and Dependability*," in *Availability, Reliability and Security, 2006. ARES 2006. The First International Conference on*, pp. 8–pp, IEEE, 2006.

[46] J. C. Laprie, *"Dependability: Basic Concepts and Terminology"*. Springer, 1992.

[47] Wikipedia, "*Fault*." `https://en.wikipedia.org/wiki/Fault`, 2018. Online; accessed 18-Jan-2019.

[48] Oracle, "*Oracle VirtualBox*." `https://www.virtualbox.org`, 2019. Online; accessed 19-June-2019.

[49] C. E. Gómez, J. Chavarriaga, and H. E. Castro, "Fault characterization and mitigation strategies in desktop cloud systems," in *Latin American High Performance Computing Conference*, (Cham), pp. 322–335, Springer International Publishing, 2019.

[50] T. M. Mengistu, A. M. Alahmadi, Y. Alsenani, A. Albuali, and D. Che, "cucloud: Volunteer computing as a service (vcaas) system," in *International Conference on Cloud Computing*, pp. 251–264, Springer, 2018.

[51] A. Chandra and J. B. Weissman, "Nebulas: Using distributed voluntary resources to build clouds.," in *HotCloud*, 2009.

[52] M. Ryden, A. Chandra, and J. Weissman, "Nebula: Data intensive computing over widely distributed voluntary resources," *Dept. of Computer Science and Eng., Univ. of Minnesota tech. report TR*, pp. 13–007, 2013.

[53] M. Amoretti, A. Grazioli, and F. Zanichelli, "An autonomic approach for p2p/cloud collaborative environments," *Peer-to-Peer Networking and Applications*, vol. 9, no. 6, pp. 1226–1241, 2016.

[54] G. A. McGilvary, A. Barker, and M. Atkinson, "*Ad Hoc Cloud Computing: From Concept to Realization*," in *2015 IEEE 8th International Conference on Cloud Computing*, pp. 1063–1068, June 2015.

[55] D. Berzano, J. Blomer, P. Buncic, I. Charalampidis, G. Ganis, and R. Meusel, "*Status and Roadmap of CernVM*," *Journal of Physics: Conference Series*, vol. 664, no. 2, p. 022018, 2015.

[56] BOINC, "*Compute for Science*." `https://boinc.berkeley.edu/`, 2019. Online; accessed 13-June-2019.

[57] G. A. McGilvary, A. Barker, A. Lloyd, and M. Atkinson, "*V-BOINC: The Virtualization of BOINC*," in *2013 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing*, pp. 285–293, IEEE, 2013.

[58] BOINC, "*Running Apps in VirtualBox Virtual Machines*." `https://boinc.berkeley.edu/trac/wiki/VboxApps`, 2019. Online; accessed 26-April-2019.

[59] G. Fedak, C. Germain, V. Neri, and F. Cappello, "*Xtremweb: A Generic Global Computing System*," in *Proceedings First IEEE/ACM International Symposium on Cluster Computing and the Grid*, pp. 582–587, IEEE, 2001.

[60] G. Kirby, A. Dearle, A. Macdonald, and A. Fernandes, "An approach to ad hoc cloud computing," *arXiv preprint arXiv:1002.4738*, 2010.

[61] CERN, "*CERN Accelerating science*." `https://home.cern`, 2019. Online; accessed 13-June-2019.

[62] S. Prathiba and S. Sowvarnica, "*Survey of Failures and Fault Tolerance in Cloud*," in *2017 2nd International Conference on Computing and Communications Technologies (ICCCT)*, pp. 169–172, IEEE, 2017.

[63] C. Colman-Meixner, C. Develder, M. Tornatore, and B. Mukherjee, "*A survey on Resiliency Techniques in Cloud Computing Infrastructures and Applications*," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 3, pp. 2244–2281, 2016.

[64] J. Chavarriaga, C. Forero-González, J. Padilla-Agudelo, A. Muñoz, R. Cáliz-Ospino, and H. Castro, "*Scaling the Deployment of Virtual Machines in UnaCloud*," in *High Performance Computing* (E. Mocskos and S. Nesmachnow, eds.), (Cham), pp. 399–413, Springer International Publishing, 2018.

[65] KVM, "*Kernel Virtual Machine*." `https://www.linux-kvm.org`, 2019. Online; accessed 19-June-2019.

[66] vmWare, "*Workstation Pro*." `https://www.vmware.com/products/workstation-pro.html`, 2019. Online; accessed 19-June-2019.

[67] Google, "*Compute Engine*." `https://cloud.google.com/compute/`, 2019. Online; accessed 15-July-2019.

[68] H. Ong, N. Saragol, K. Chanchio, and C. Leangsuksun, "*VCCP: A Transparent, Coordinated Checkpointing System for Virtualization-based Cluster Computing*," in *Cluster Computing and Workshops, 2009. CLUSTER'09. IEEE International Conference on*, pp. 1–10, IEEE, 2009.

[69] H. Agarwal and A. Sharma, "*A Comprehensive Survey of Fault Tolerance Techniques in Cloud Computing*," in *Computing and Network Communications (CoCoNet), 2015 International Conference on*, pp. 408–413, IEEE, 2015.

[70] A. Kangarlou, D. Xu, P. Ruth, and P. Eugster, "*Taking Snapshots of Virtual Networked Environments*," in *Virtualization Technology in Distributed Computing (VTDC), 2007 Second International Workshop on*, pp. 1–8, IEEE, 2007.

[71] Oracle Corporation, *Oracle VM VirtualBox: User Manual Version 5.0.20*. Oracle Corporation, 2016.

[72] K. M. Chandy and L. Lamport, "*Distributed snapshots: Determining global states of distributed systems*," *ACM Transactions on Computer Systems (TOCS)*, vol. 3, no. 1, pp. 63–75, 1985.

[73] F. Mattern, "*Efficient Algorithms for Distributed Snapshots and Global Virtual Time Approximation*," *Journal of Parallel and Distributed Computing*, vol. 18, no. 4, pp. 423–434, 1993.

[74] A. D. Kshemkalyani and M. Singhal, *Distributed Computing: Principles, Algorithms, and Systems*. Cambridge University Press, 2011.

[75] T. H. Lai and T. H. Yang, "*On Distributed Snapshots*," *Information Processing Letters*, vol. 25, no. 3, pp. 153–158, 1987.

[76] X. Jiang and D. Xu, "*Violin: Virtual Internetworking on Overlay Infrastructure*," in *International Symposium on Parallel and Distributed Processing and Applications*, pp. 937–946, Springer, 2004.

[77] J. F. Kurose and K. W. Ross, *Computer Networking: A Top-Down Approach*. Addison-Wesley, 2013.