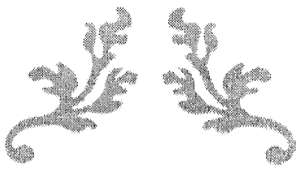
Bond, Román Alejandro

Simulación para el análisis de performance de un sistema de archivos distribuido en Cloud

2019

Instituto de Ingeniería y Agronomía

Ingeniería en Informática



PRÁCTICA PROFESIONAL SUPERVISADA

Título: Simulación para el análisis de performance de un sistema de archivos distribuido en Cloud.

Informe Final

ROMÁN ALEJANDRO BOND

LEGAJO: 3913

Universidad Nacional Arturo Jauretche

Instituto de Ingeniería y Agronomía

Ingeniería en Informática

Universidad Nacional ARTURO JAURETCHE *Instituto de Ingeniería y Agronomía* *Ingeniería en Informática*	**Práctica Profesional Supervisada (PPS)**

PRÁCTICA PROFESIONAL SUPERVISADA (PPS)
Simulación para el análisis de performance de un sistema de archivos distribuido en Cloud

DATOS DEL ESTUDIANTE

Apellido y Nombres: Bond, Román Alejandro

DNI: 37554558

Nº de Legajo: 3913

Correo electrónico: roman.alejandro.b@gmail.com

Cantidad de materias aprobadas al comienzo de la PPS: 44

DOCENTE SUPERVISOR

Apellido y Nombres: prof. Dr. Morales, Martín

Correo electrónico: martin.morales@unaj.edu.ar

DOCENTE TUTOR DEL TALLER DE APOYO A LA PRODUCCIÓN DE TEXTOS ACADÉMICOS DE LA UNAJ

Apellido y Nombres: prof. Dra. Ferrari, Mariela

Correo electrónico: mariela_c_ferrari@hotmail.com

DATOS DE LA ORGANIZACIÓN DONDE SE REALIZA LA PPS

Nombre o Razón Social: Universidad Nacional Arturo Jauretche

Dirección: Av. Calchaquí 6200, Florencio Varela, Buenos Aires

Teléfono: +54 11 4275 6100

Sector: Programa Tecnologías de la Información y la Comunicación (TIC) en aplicaciones de interés social, Instituto de Ingeniería y Agronomía

TUTOR DE LA ORGANIZACION

Apellido y Nombres: prof. Mg. Encinas, Diego Omar

Correo electrónico: dencinas@unaj.edu.ar

FIRMA DEL COORDINADOR DE LA CARRERA

Dr. Ing. MARTIN MORALES
Vicedirector
Instituto de Ingeniería y Agronomía
Universidad Nacional Arturo Jauretche

Firma Estudiante:	Firma Docente Supervisor:	Firma docente tutor TAPTA:	Firma tutor Organizacional:

Dr. Ing. MARTIN MORALES
Vicedirector
Instituto de Ingeniería y Agronomía
Universidad Nacional Arturo Jauretche

Contenido

Capítulo 1. Introducción

En 1961, John McCarthy propone el uso de la computación como un servicio público. Cada usuario pagaría únicamente por la capacidad de cómputo que utilizara, de manera muy similar a la que un usuario del servicio de telefonía público paga por minuto de llamada. La idea de McCarthy está estrechamente relacionada con las tecnologías *cloud computing*, que permiten acceder a procesamiento y servicios de software, bajo demanda, a través de Internet. Este fenómeno impuso un cambio de paradigma en cómputo y tratamiento de la información, haciendo que desarrolladores, investigadores, administradores de sistemas y empresas contraten servicios *cloud* para establecerse o para expandirse y seguir desarrollando sus actividades.

Para que *cloud computing* pasara de ser una idea a un hecho, se necesitaron diferentes avances tecnológicos. Algunos de ellos, como la virtualización, los modelos de comunicación de procesos remotos, las técnicas de modelado y simulación, aparecieron en los años 80. Otros aparecieron dentro de un período más reciente, por ejemplo, el almacenamiento de estado sólido, los *web services* y la fibra óptica.

En Septiembre de 1962, Gordon Geoffrey publica un artículo en *IBM Systems Journal*, titulado "Un simulador de sistemas de propósito general". En este trabajo, se define por primera vez un programa que permite crear simulaciones utilizando diagramas de bloques como lenguaje para describir el sistema. Este hecho fue el disparador para que, en los siguientes años, la simulación computacional se desarrollara como área de interés en informática y computación. Actualmente, la "Organización Europea para la Investigación Nuclear" (CERN) realiza

simulaciones de colisiones de partículas de gran cantidad de energía. Utilizan los resultados para obtener resultados teóricos para contrastarlos con los resultados del gran acelerador de partículas (*Large Hadron Collider* o LHC).

Por otro lado, en la década de los 70, se empezó a trabajar en la idea de que diferentes equipos conectados a una misma red puedan comunicar procesos entre sí. De esta forma, se elaboró un modelo de comunicación, llamado *Remote Procedure Call* (Llamadas a procedimientos remotos), que permitía ejecutar procedimientos o programas en un equipo remoto, de manera transparente, sin tener que preocuparse por la forma de comunicación o por el tipo de sistema que hubiera en el otro extremo. Esto dio origen al concepto de "sistemas distribuidos", que resulta central para el presente trabajo, como veremos desde el capítulo 2 en adelante.

Recién en 1981 se logró realizar una implementación del modelo de comunicación, en el centro de investigación de Xerox, en Palo Alto. Andrew Birrell y Bruce Nelson estuvieron a cargo de la implementación y la nombraron *Open Network Computing - Remote Procedure Call* (ONC RPC). Fue desarrollado principalmente para el entorno de desarrollo *Cedar* y para comunicarse con otros equipos que formaban parte de la red interna de investigación de Xerox. Buscaban comunicar y ejecutar procedimientos desde una máquina con bajos recursos a una máquina con mayores prestaciones (Dorado), de manera transparente y sencilla.

No solo se buscaba comunicar y ejecutar procesos de manera transparente: querían lograr que diferentes tipos de máquinas conectadas a la misma red pudieran acceder a archivos que se encuentran en un servidor remoto, como si se tratase de un archivo local. Es por esto, que, a partir del desarrollo de ONC RPC, en el año 1984, se implementó un sistema de archivos distribuido en red

denominado *SunNFS* a cargo de la empresa *Sun Microsystems*. Formalmente, idearon un protocolo de comunicación llamado *Network File System* que, inicialmente, fue implementado para el sistema operativo *Solaris SunOS*.

El presente trabajo se propone el desarrollo de una herramienta que permita realizar el análisis temporal y espacial de los sistemas de archivos utilizados en sistemas distribuidos. Para conseguirlo, se plantea el estudio y modelado del funcionamiento y comportamiento del sistema de archivos *Network File System*. Este es un protocolo que permite al usuario o administrador del sistema distribuido montar, o designar como accesible, la totalidad o una porción de un sistema de archivos local en un servidor. La porción del sistema de archivos montada puede ser accedida por múltiples clientes, con los privilegios que hayan sido asignados a cada archivo (únicamente lectura o, en algunos casos, lectura-escritura), como si estuvieran accediendo a su propio dispositivo de almacenamiento local.
Se implementará el modelo de todo el sistema por medio del *framework* de simulación para *cloud computing* CloudSim. Finalmente, se contrastarán los resultados del simulador con un escenario desplegado en un *cloud* público (*Amazon Web Services*).

Capítulo 2. Sistemas distribuidos

En este capítulo, se definirá el concepto de "sistema distribuido", sus características y objetivos. También se verá el papel central del *middleware* dentro de las arquitecturas distribuidas.

2.1. Definición

Un sistema distribuido es un conjunto de máquinas o elementos de cómputo independientes que, conectadas a una red de comunicación, forman un entorno de cómputo consistente e integrado.

Tanenbaum agrega que este conjunto o sistema de elementos "da al usuario la sensación de que se trata de un único sistema coherente" (Tanenbaum, 1996: 2)

Por otro lado, Coulouris, Dollimore y Kindberg van un poco más allá en la definición. Su aporte hace énfasis en cómo se deben comunicar los dispositivos de cómputo: "(estos sistemas) comunican y coordinan sus acciones únicamente mediante el paso de mensajes" (Coulouris, Dollimore y Kindberg, 2001: 2)

De esta definición, se observan, a priori, tres características fundamentales:

- Se trata de un conjunto de elementos independientes o autónomos, es decir, el sistema puede estar compuesto por diferentes tipos de máquinas y dispositivos. Por ejemplo, pueden coexistir nodos con gran poder de procesamiento (servidores), con nodos similares a un equipo de escritorio estándar de uso personal. El término independiente hace referencia a que cada máquina puede actuar de manera autónoma. Al momento de resolver un problema de cómputo, el administrador puede determinar qué máquinas utilizar, de acuerdo a la disponibilidad o las características, y cuáles no.
- Es un único sistema coherente, por lo tanto, todos los recursos disponibles en los diferentes dispositivos son vistos como los de una única máquina, sin

importar cuáles son las especificaciones de cada nodo. Por ejemplo, en el sitio *web* de *Virtual Institute for I/O* se publican anualmente la lista de mejores *clusters* y supercomputadores. Uno de los mejores del año 2018 es el sistema de *Oak Ridge National Laboratory*, que está conformado con 18,688 nodos. Posee una capacidad de almacenamiento de 278 *PetaBytes*, memoria RAM total de 3.511,66 *TeraBytes* y la posibilidad de realizar operaciones de punto flotante a 220,64 PFLOPS (*PetaFLOPS per second*).

- Estos sistemas comunican y coordinan sus acciones mediante paso de mensajes. En tal sentido, al momento de resolver un problema, los procesos necesitan comunicarse entre sí y, para esto, intercambian mensajes. El principal motivo de que la comunicación sea de esta manera se debe a la falta de un reloj global.

2.2. Objetivos

Además de las tres características mencionadas, los autores coinciden en una serie de objetivos o desafíos al momento de diseñar un sistema distribuido.

- Compartir recursos, es decir, garantizar el acceso a los recursos fácilmente, ya sea para usuarios o para las aplicaciones. En un principio, se deseaba compartir recursos por una razón económica. Resultaba menos costos contar con una solución de almacenamiento compartida que comprar y mantener una por cada usuario. Actualmente, con el desarrollo de las empresas 2.0 y el aumento del trabajo colaborativo a distancia, es necesario contar con recursos a los que los usuarios puedan acceder remotamente.
- Transparencia: esto significa ocultar a usuarios y programadores, el hecho de que los procesos y los recursos están en diferentes equipos y en

distintos lugares. Existen diferentes tipos de transparencias y diferentes autores agregan o quitan algunos tipos. Garantizar transparencia puede ser costoso en términos de cómputo, ya que puede implicar sacrificar performance. El término objetos hace referencia a recursos, datos y procedimientos. En este sentido, puede verse la siguiente tabla, que explica y enumera los diferentes tipos de transparencia que pueden ser aplicados a un determinado sistema distribuido.

Transparencia	Descripción
Acceso	Oculta cómo se representan y acceden los objetos.
Locación	Oculta dónde se encuentran los objetos.
Relocación	Similar a locación. Oculta donde se encuentra el objeto mientras está siendo usado.
Migración	Oculta cuando un objeto es desplazado de un lugar a otro.
Replicación	Oculta cuando un objeto es una réplica del original.
Concurrencia	Oculta el hecho de que un objeto pueda ser accedido por múltiples

	usuarios al mismo tiempo.
Fallo	Oculta el proceso de fallo y recuperación del objeto.

Tabla 1. *Tipos de transparencias.* (Van Steen y Tanenbaum, 2017: 8)

- Sistema abierto: consiste en garantizar que los recursos y elementos que ofrece el sistema sean fáciles de utilizar por usuarios finales u otros sistemas. Se tienen en cuenta los siguientes aspectos: en primer lugar, se deben respetar las interfaces (*key interfaces software*) ya definidas. En segundo lugar, se debe proveer un sistema de comunicación uniforme y es necesario publicar interfaces para que se pueda acceder a los diferentes recursos. Por último, cada sistema distribuido puede tener componentes heterogéneos. Esto posibilita que el sistema sea conformado por diferentes proveedores de hardware. Es por esto que cada pieza de *hardware* y *software* debe ser correctamente seleccionada de acuerdo a los estándares definidos.
- Escalabilidad: un sistema es escalable "cuando puede manejar el aumento de usuarios o el agregado de nuevos recursos, sin sufrir perdida de performance considerable y sin aumentar la dificultad de mantener el sistema" (Neuman, 1994: 1).Por otro lado, el mismo autor establece que la escalabilidad puede medirse en tres dimensiones. La primera es el tamaño, o sea, medir el crecimiento en términos de cantidad de usuarios y recursos. La segunda dimensión es la geográfica (distancia que hay entre los sistemas). Por último, la dimensión relacionada con lo administrativo es la que determina la cantidad de dominios de administración del sistema. (Ibíd:

3).

- Heterogeneidad: hace referencia a los diferentes tipos de *hardware*, *software* y diferentes implementaciones de protocolos y modelos de comunicación que forman parte del sistema distribuido. Se debe tener en cuenta que la manera de representar datos que tiene cada tipo de hardware es distinta. Por ejemplo, hay dos modos de ordenar los *bits* que conforman un dato de tipo *string*: desde el *bit* más significativo al menos significativo (*big endian*), o viceversa (*little endian*). Esto ocasiona problemas de comunicación, pues una máquina con determinado hardware interpreta el *string* de manera correcta y otra, de manera errónea. Es necesario definir una capa de software que provea abstracción al momento de programar. Se logra, entonces, ocultar la heterogeneidad que aplica para las redes, los protocolos, el hardware y el software. La capa de software que se encarga de eso se llama *middleware*. Es un componente esencial y fundamental en los sistemas distribuidos. En la sección 2.3 se abordará este tema con más detalle.
- Seguridad: uno de los objetivos de los sistemas distribuidos es compartir recursos, desde una transacción bancaria, hasta datos financieros de una empresa. ¿Qué sucede si ese recurso que se está compartiendo o accediendo en la red es interrumpido y alterado intencionalmente por alguien ajeno al sistema? Probablemente, esa transacción sea desviada hacia otra cuenta bancaria o los datos financieros terminen en manos de una empresa enemiga. Mantener la seguridad en este tipo de sistemas es esencial. La seguridad es la sensación que tienen las personas de sentirse seguro, a salvos o protegidos. Para garantizar esa sensación, se

implementan diferentes estrategias y políticas en los sistemas distribuidos. Algunas de ellas pueden ser utilizar *firewalls*, implementar redes con zonas desmilitarizadas, crear listas de accesos, utilizar protocolos de comunicación encriptados, entre otras.

- Manejo de errores: como cualquier sistema, las computadoras fallan. Ya sea *hardware* o *software*, alguno de estos elementos puede dejar de funcionar, interrumpiendo al equipo completo, en el peor de los casos. En los sistemas distribuidos, los fallos son parciales: si un componente del sistema falla, otros componentes seguirán funcionando correctamente, lo cual implica que este tipo de sistema es "tolerante a fallos". Para lidiar con errores, hay dos etapas bien establecidas. La primera etapa consiste en detectar el fallo. Por ejemplo, utilizando sumas de comprobación. La segunda etapa consiste en recuperarse del error. Por ejemplo, retransmitir un paquete de red que falló en llegar. Una de las técnicas más utilizadas para enfrentar problemas de fallos es la redundancia. Consiste en contar con otro recurso idéntico al recurso crítico. Se puede conectar cada equipo o elemento a dos redes distintas, separadas una de otra. Si falla una, la otra sigue intacta y el sistema sigue trabajando normalmente.
- Concurrencia: esta característica permite que los recursos disponibles en la red puedan ser utilizados por diferentes usuarios al mismo tiempo. Para que no ocurran errores de integridad y consistencia, las operaciones concurrentes sobre recursos y objetos deben sincronizarse entre sí.

2.3. Middleware

Anteriormente, se estableció que los componentes de un sistema distribuido necesitaban comunicarse entre sí. Para esto, era necesario contar con una capa

de *software*, llamada *middleware*, que oculte la heterogeneidad que aplica para las redes, los protocolos, el *hardware* y el *software*.

Desde otro punto de vista, esta capa encapsula las diferencias en la red para ofrecer recursos y servicios de forma transparente. Para los desarrolladores de aplicaciones, ofrece una herramienta (caja negra) que facilita el desarrollo de aplicaciones distribuidas. Para esto, el *middleware* brinda una API (*Application Programming Interface*) que el programador utiliza como herramienta o librería dentro su programa.

Desde el punto de vista de un administrador de un sistema distribuido, funciona como una caja transparente y facilitador de la comunicación. El administrador puede ver sus procesos y funcionamiento interno.

Generalmente, está definido por encima del sistema operativo, pero por debajo de la aplicación distribuida, tal como se explica en la figura 1.

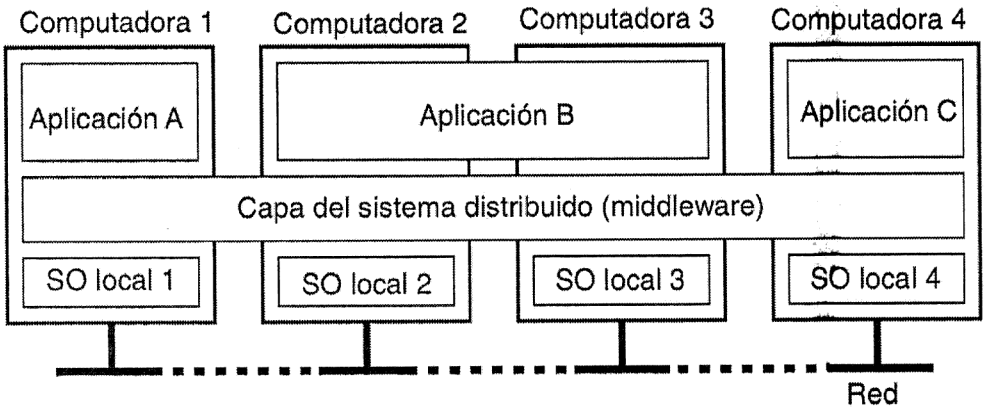


Figura 1. *Sistema distribuido organizado como middleware* (Tanenbaum, 1996: 3)

En la imagen, puede verse cuatro computadoras conectadas en red y tres aplicaciones. La aplicación "B" está distribuida entre las computadoras dos y tres. La capa de middleware se extiende sobre cada computadora brindado la misma

interfaz a cada aplicación, permitiendo que se comuniquen entre sí.

Un *middleware* está compuesto por dos elementos principales. El primero es el modelo de comunicación que determina y establece cómo se realiza. El segundo componente define cómo se representan o serializan los datos que se van a intercambiar, tal como se esquematiza en la figura 2.

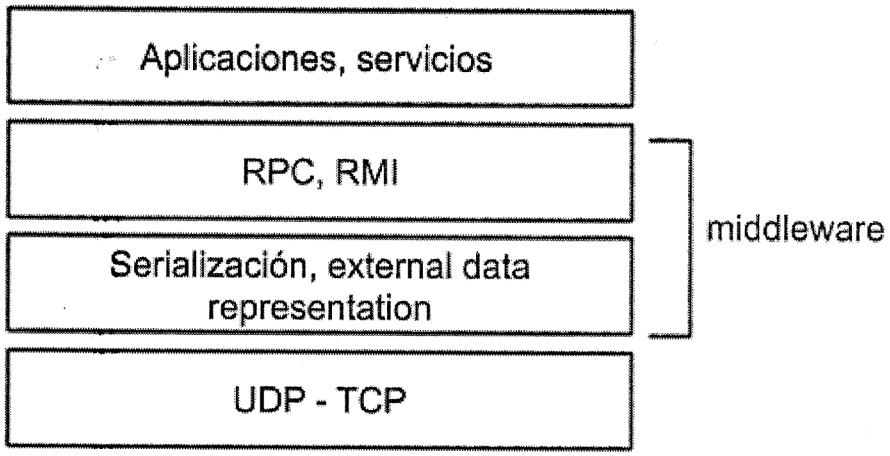


Figura 2.*Capas de middleware* (imagen propia)

Existen varios tipos de modelos de comunicación con diferentes implementaciones. Dos de los modelos más conocidos y utilizados son: *Remote Procedure Call* (RPC), implementado por primera vez en 1981 por *Sun Microsystems*. Permite ejecutar o realizar llamados a procedimientos que se encuentran en un dispositivo remoto conectado a una red. El segundo modelo es *Remote Invocation Method* (RMI) y fue implementado para el lenguaje de programación Java. Permite a un objeto, que reside en una *Java Virtual Machine* (JVM), invocar un método que se encuentra en otra JVM.

2.4. Tipos de sistemas distribuidos

Tanenbaum (1996) propone tres tipos de sistemas distribuidos:

- Sistemas de cómputo distribuido de alto rendimiento: de este grupo se desprenden dos grandes categorías. *Clustering* o cómputo en clúster, en donde, generalmente, todos los nodos son de características homogéneas que están conectadas a una red de área local de alta velocidad (en los mejores casos, *Infiniband*). Este grupo es utilizado para programación paralela en donde un solo programa, de cálculo intensivo, corre paralelamente en todos los nodos del clúster para resolver el problema. En la figura 3, se observa la configuración básica de un clúster: generalmente, consta de un nodo *master* que ejecuta el *middleware* y los programas necesarios para la administración de un conjunto de nodos de cómputo.

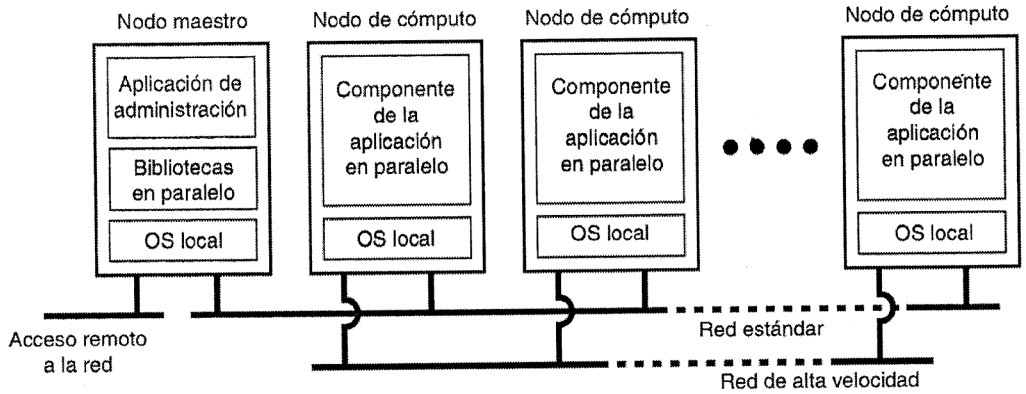


Figura 3. *Ejemplo de clúster* (Tanenbaum, 1996: 17)

Por otro lado, existen sistemas de cómputo *grid* que, a diferencia de un clúster, son totalmente heterogéneos en todos los aspectos: hardware, red, sistemas operativos, entre otros. Para reunir los recursos, diferentes organismos y organizaciones se conforman con lo que es conocido como organización virtual. Cada uno de los miembros es dueño del procesamiento o recursos de la organización virtual.

Firma Estudiante:	Firma Docente Supervisor:	Firma docente tutor TAPTA:	Firma tutor Organizacional:

En la figura 4, se observa la arquitectura en capas (Foster, 2001: 207). El principal objetivo de esta arquitectura es otorgar privilegios a los usuarios y aplicaciones que forman parte de una organización. A continuación, se detalla la función de cada una de las capas que componen esta arquitectura. La capa de fabricación provee interfaces para los recursos. También brinda una serie de herramientas para saber el estado del recurso y para administrarlo. La capa de conectividad brinda modelos y mecanismos de comunicación para poder establecer comunicación y autenticación entre los recursos. La capa de recursos administra un solo recurso y permite su configuración y acceso. Esta se vale de la capa de conectividad para establecer una conexión al sistema. También utiliza la capa de fabricación para acceder a las interfaces. La siguiente capa es la colectiva y se encarga de acceder a múltiples recursos. Posee mecanismos para indexar y buscar otros recursos. Por último, la capa de aplicación está constituida por aplicaciones que permiten la administración y uso de los recursos del sistema *grid*, tal como muestra la siguiente figura.

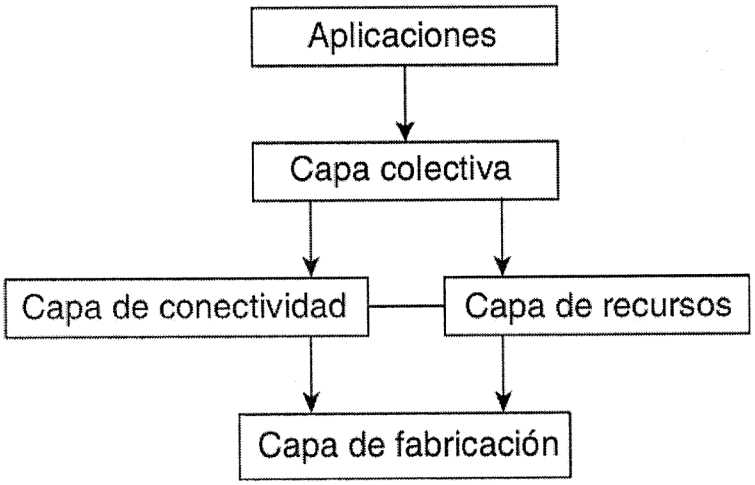


Figura 4. *Arquitectura grid computing propuesta* (Foster, 2001: 207)

- Sistemas distribuidos de información: de este tipo de sistemas se desprenden, a su vez, dos tipos de sistemas. Por un lado, están los sistemas de procesamiento de transacciones, que utilizan transacciones anidadas compuestas por subtransacciones pequeñas. La transacción de más alto nivel puede dividirse en subprocesos que se ejecutan en paralelo y en diferentes equipos. El *middleware* encargado de comunicar los procesos y subprocesos en este tipo de sistemas es conocido como "monitor de procesamiento de transacciones". Este *middleware* permite el acceso a múltiples bases de datos.

  Por otro lado, existen los sistemas de integración de aplicaciones empresariales. Estos sistemas utilizan modelos de comunicación que permiten la interacción entre diferentes aplicaciones (*middleware*). Los modelos más utilizados son RPC y RMI. No obstante, aparece un tercer modelo conocido como "*middleware* orientado a mensajes" (*message*

oriented middleware).En este caso, el *middleware* facilita, únicamente, la comunicación entre aplicaciones distribuidas. Para esto envía mensajes de una aplicación a otra mediante una cola como paso intermedio. Entonces, la aplicación cliente envía un mensaje, se almacena en la cola esperando a que la aplicación servidor verifique si existen mensajes pendientes.

- Sistemas distribuidos embebidos: los sistemas más notables de este conjunto son las redes de sensores. Cada red de sensores está conformada por miles de dispositivos pequeños. Cada nodo contiene uno o más sensores. Generalmente, estos sistemas son utilizados para monitoreo, por ejemplo, en agricultura o en transporte.

Capítulo 3. Sistema de archivos

La finalidad de este capítulo es, por un lado, introducir los conceptos de sistemas de archivos y su implementación en sistemas UNIX. Por otro lado, se analizarán los conceptos relacionados con los sistemas de archivos distribuidos: definición, tipos de modelos (de acceso y de transferencia de archivos) y ventajas del modelo utilizado por los sistemas de archivos distribuidos (de acceso de archivos).

3.1 Definición

En todo momento, los procesos están creando, recuperando, manipulando y almacenando datos. El sistema operativo, además de encargarse de la correcta gestión de los procesos, posee un mecanismo que permite realizar todas las tareas relacionadas con la manipulación de datos.

El mecanismo que permite al sistema operativo controlar el modo en que los datos son almacenados y recuperados se denomina "sistema de archivos". Si no existiese este sistema, en el disco duro existiría una gran cantidad de información sin relación, de la cual no se sabe dónde comienza y termina.

Existen varios tipos de sistemas de archivos, cada cual con su estructura, seguridad y nivel de rendimiento. En la figura 5, se observa la estructura de un sistema de archivos en UNIX y su interacción con el sistema operativo:

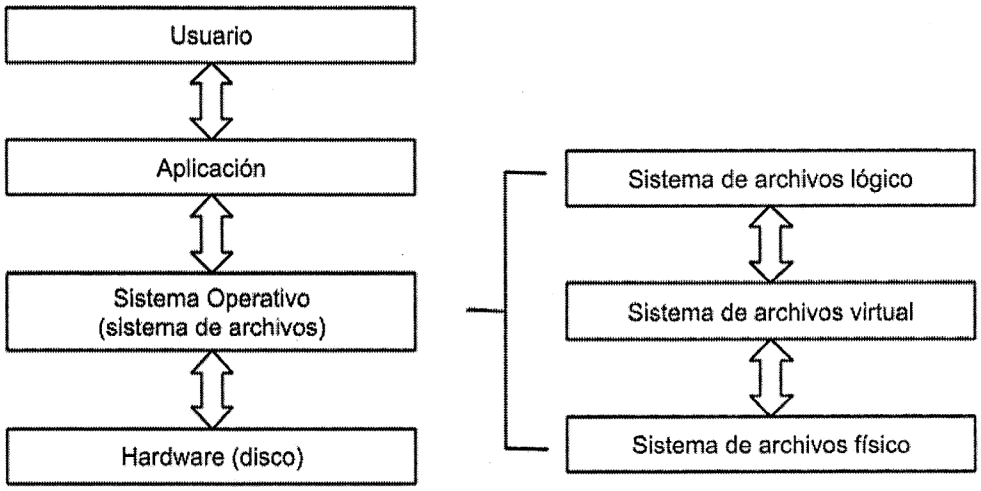


Figura 5. *Estructura de un sistema de archivos en SO. Interacción con el usuario* (imagen propia)

El sistema de archivos lógico es responsable de la interacción con la aplicación del usuario. Provee la interfaz de programación de aplicaciones (API) para las operaciones de archivos (*open*, *close*, *read*, etc.) y envía la operación solicitada a la capa inferior para que sea procesada.

En sistemas UNIX, se busca siempre integrar varios tipos de sistemas de archivos en una única estructura. Es decir, se trata de posibilitar que varios sistemas de archivos interactúen dentro del mismo sistema operativo. La capa que permite esto es la del sistema de archivos virtual o *Virtual File System* (VFS). La idea clave de esta capa es abstraer la parte del sistema de archivos que es común para todos los sistemas de archivos y poner ese código en una capa separada, tal como se ve en la figura 6.

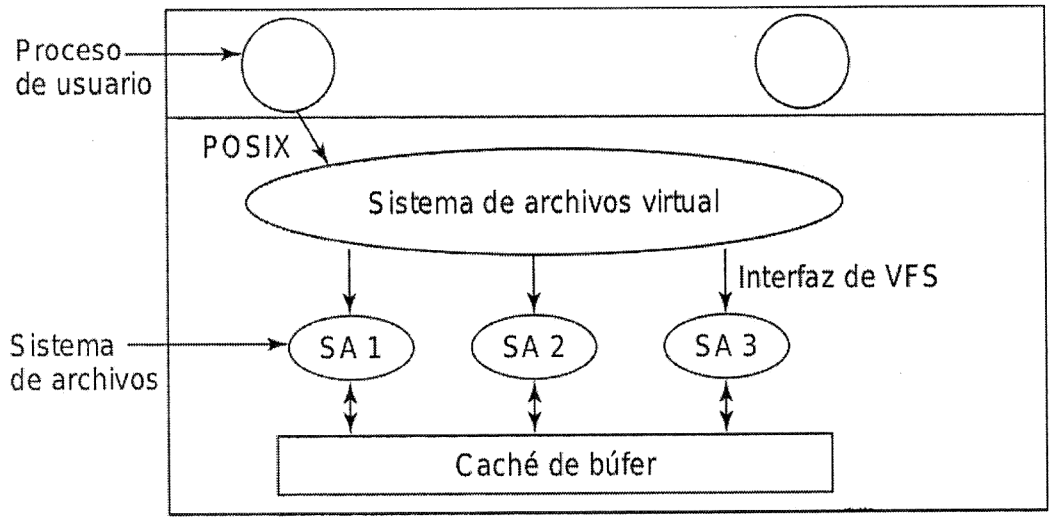


Figura 6. *Interacción del VFS con diferentes sistemas de archivos*
(Tanenbaum, 1992: 289)

Por último, la tercera capa es el sistema de archivos físico. Se encarga de las tareas físicas relacionadas con el hardware y permite la lectura y la escritura de bloques físicos. Esta capa interactúa con los controladores de los dispositivos o con el canal de E/S para revisar el dispositivo de almacenamiento.

3.2 Sistema de archivos distribuidos

Con la aparición de las primeras redes de computadoras, la necesidad de compartir recursos, específicamente, archivos, fue creciendo cada vez más. En la década de los 80, aparecen los primeros protocolos y programas que permitían transferir un archivo de una red/computadora a otra (FTP o UUCP). Contar con discos para cada usuario era relativamente caro. Además, el sistema de archivos local solo podía ser accedido por un conjunto de procesos locales. Entonces, nace una nueva necesidad: la de acceder a archivos mediante la red sin realizar la transferencia de estos.

Así aparecen los primeros sistemas de archivos distribuidos, por ejemplo, *Andrew File System*, *Network File System* (NFS) y *AT&T's Remote File Sharing*. El sistema que perduró en el tiempo fue NFS.

Los sistemas de archivos distribuidos utilizan la arquitectura cliente/servidor. En estos, el servidor contiene el sistema de archivos al que los clientes acceden a través de protocolos de comunicación. De esta manera, permite el acceso a archivos. Cada servidor de archivos brinda una vista estandarizada o común de su sistema de archivos local. Entonces, los clientes pueden acceder de modo transparente a los archivos, dando la sensación de que el archivo está en la máquina del cliente.

Los sistemas de archivos distribuidos utilizan el modelo de acceso remoto, que elimina la necesidad de mover un archivo entero por la red. Como puede verse en la figura 7, la clave está en que el archivo siempre está en el servidor y el trabajo se realice "*in-place*". Esto supone una serie de ventajas frente a modelos de transferencia de archivos. Callaghan (1999) establece las siguientes ventajas:

- Si el cliente solo desea una porción del archivo, se envía únicamente esa porción.
- Brinda la sensación de que el archivo remoto se encuentra localmente. El usuario no sabe cuál es la ubicación exacta del archivo como tampoco sabe dónde se encuentra el servidor.
- Los archivos siempre se encuentran en la última versión, actualizados a la fecha.
- Requiere un mínimo espacio de disco en los nodos clientes.
- Se utilizan tiempos de esperas mínimos. Una vez que el archivo se encuentra en el servidor, estará disponible para todos los clientes.

- Ofrece mecanismos de bloqueo de archivos a los clientes. Esto puede generar problemas de inconsistencia.

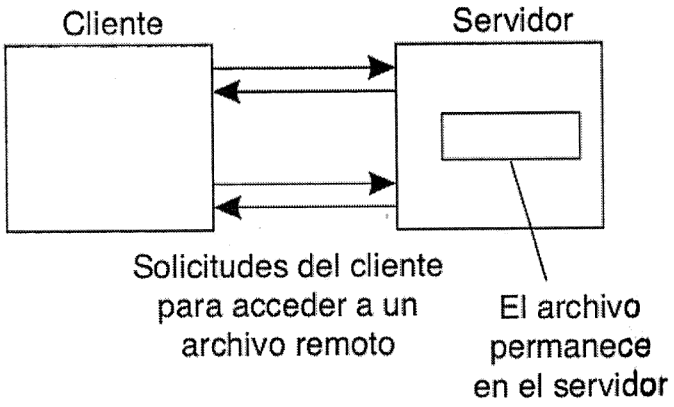


Figura 7. *Modelo de acceso remoto* (Tanenbaum, 1992: 592)

Capítulo 4. Network File System

En este capítulo, se mostrarán los conceptos relacionados con NFS: funcionamiento, arquitectura y middleware que se utilizan como sistema de archivos distribuido. A su vez, se caracterizarán los protocolos extras que hacen al funcionamiento de NFS: RPC, XDR, *Lock Manager* y *Mount*.

4.1 Definición

Un NFS es un protocolo de red que permite a archivos alojados en un servidor ser accedidos por múltiples clientes a través de la red. Este protocolo está basado en la arquitectura cliente/servidor.

El primer prototipo implementado fue realizado por *Sun Microsystems* en el año 1981, junto con el modelo de comunicación RPC. La segunda versión nace en 1984, realizada por la misma empresa. El protocolo es especificado detalladamente para la IETF (*Internet Engineering Task Force*) en el documento RFC 1904 (*Request For Comments*). A partir de este suceso, NFS es un protocolo estándar.

Es importante realizar una distinción entre protocolo e implementación. NFS es un protocolo de red. La implementación de este protocolo da como resultado un sistema de archivos distribuido robusto que utiliza el modelo de acceso a archivo remoto (cf. sección 3.2). De esta manera, este sistema de archivos ofrece acceso a archivos a través de una red heterogénea y de manera transparente para los usuarios.

Sandberg (1985) establece las características principales que debe tener un NFS:

- Independencia del tipo de sistema operativo y del hardware: el protocolo debe ser independiente del sistema operativo. De esta forma, varios tipos de clientes (Windows, UNIX-*like*, etc.) pueden acceder a los archivos. El

protocolo debe ser sencillo.

- Recuperación ante fallos: es importante que los clientes puedan recuperarse frente a fallos del servidor. Si hay fallos en el servidor, el cliente no debería notar cambios en el funcionamiento o flujo de trabajo normal y habitual.
- Acceso transparente: de acuerdo con esto, es necesario proveer un sistema que permita el acceso a archivos remotos del mismo modo en el que se accede a un archivo local.
- Mantener la semántica UNIX en el cliente: para cumplir con el anterior objetivo, es necesario mantener la semántica de los sistemas de archivos UNIX en los clientes.
- Performance aceptable: al respecto, Sandberg explica que: "los usuarios no van a querer utilizar NFS si no es más rápido y fácil de utilizar que las demás herramientas de red, como las utilidades que brinda la implementación de RPC (Sandberg, 1985: p.)". Por eso, se propuso que NFS fuera más rápido que el protocolo *Sun Network Disk Protocol* o, al menos, un 80% más rápido que un disco local.

4.2 Middleware de NFS

NFS requiere otros protocolos para su funcionamiento, como, por ejemplo, *Remote Procedure Call* (RPC) y *External Data Representation* (XDR). Estos protocolos son los más importantes de NFS. A su vez, existen dos protocolos más: *Lock Manager* y *Mount*. En las siguientes secciones, se profundizará más en estos protocolos. A fin de comprender mejor dichos protocolos, presentamos a continuación el diagrama que los muestra, en la figura 8.

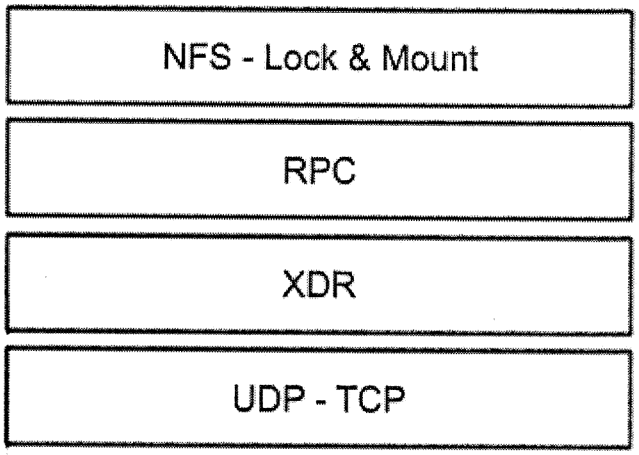


Figura 8. *Pila de protocolos NFS. Relación con middleware* (imagen propia)

Si se compara la Figura 2 (estructura de un *middleware*), presentada previamente, con la Figura 8, se observa que la capa de aplicación corresponde a la implementación NFS (cliente o servidor) con los protocolos *Lock Manager* y *Mount*. La capa de comunicación de la Figura 2 es idéntica a la de la Figura 8. Lo más significativo de esta comparación es analizar los componentes de un middleware. Anteriormente, se estableció que un *middleware* estaba compuesto por un modelo de comunicación y por un mecanismo que permitía representar la información. Comparando ambas figuras, se observa que el *middleware* utilizado por NFS está compuesto por RCP como modelo de comunicación y XDR, para representar la información.

Esto brinda una primera aproximación a la forma en que se realiza la comunicación entre clientes y servidores

4.3 Arquitectura del protocolo

NFS utiliza la arquitectura cliente/servidor. Esto quiere decir que hay dos

implementaciones que se realizan del protocolo. Por un lado, se tiene una implementación del protocolo para el cliente, llamado "cliente NFS". Este accede a los archivos remotos montando el directorio del servidor que contiene toda la estructura de archivos. Por otro lado, se tiene una implementación del servidor llamado "servidor NFS". Este, además de contener el sistema de archivos al que acceden los clientes, posee un proceso o demonio (para sistemas UNIX) que está continuamente en ejecución, denominado "nfsd". En la figura 9, que presentamos a continuación, se observa cómo interactúan el cliente y el servidor con los diferentes protocolos que hacen a NFS.

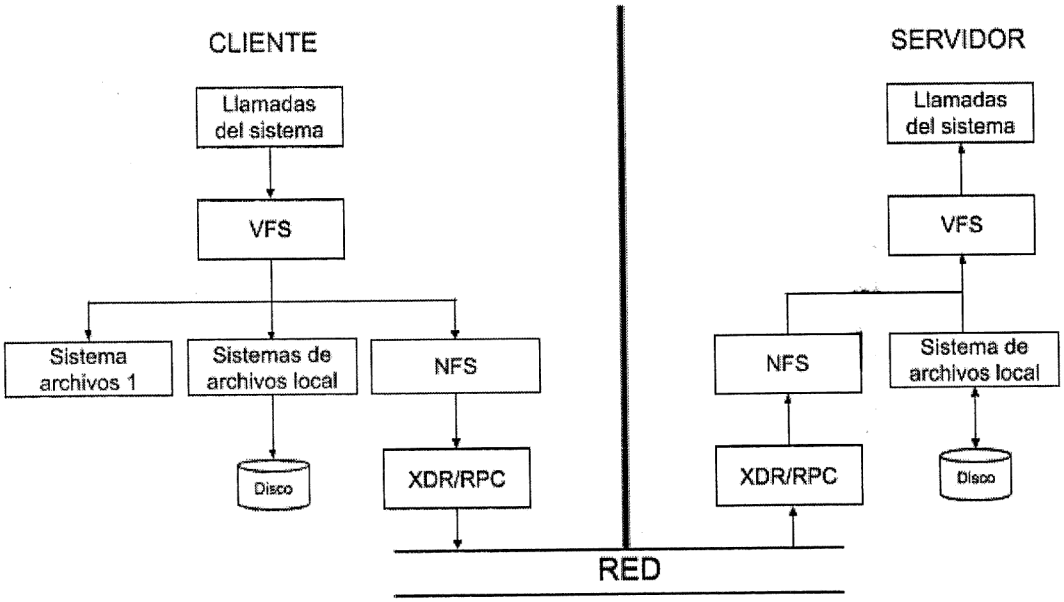


Figura 9. *Arquitectura protocolo NFS* (Sandberg, 1985: 12)

Otro aspecto importante a mencionar es que NFS es un protocolo sin estado (*stateless*). Esto quiere decir que el servidor no necesita mantener ningún tipo de información sobre el estado de los clientes o sobre el estado de las llamadas a

procedimientos, para funcionar debidamente.

4.4. Procedimientos NFS

Para poder realizar operaciones sobre archivos y directorios, NFS implementa una serie de procedimientos bien definidos. Cada procedimiento tiene un nombre, un identificador único y una serie de parámetros necesarios para que el servidor pueda ejecutar la tarea correctamente. Una vez finalizada la ejecución de la tarea, se devuelve algún valor de retorno.

El formato de un procedimiento es el siguiente: PROCEDIMIENTO(parámetro_1, parámetro_2, parámetro_3). Por ejemplo, para realizar la lectura de una archivo, es necesario invocar el procedimiento *READ* con sus parámetros: READ(file_handle, offset, count). A continuación, se listan los parámetros del procedimiento:

- *file handle* es una estructura, que proviene del servidor y tiene la referencia del archivo que va a utilizar el cliente. En este caso, la referencia del archivo que se va a leer.
- *offset*: desde que byte se comienza a leer el archivo.
- *count*: la cantidad de bytes a leer.

Como resultado, retorna:

- *attributes*: atributos propios del archivo a leer
- *data*: los datos leídos.

En el anexo A, se encuentra una lista completa de los procedimientos implementados por la versión 3 de NFS.

En las siguientes secciones, se profundizarán los protocolos RPC, XDR, *Lock Manager* y *Mount*. Esto tiene como propósito mostrar el funcionamiento de NFS.

4.5 Remote Procedure Call (RPC)

El modelo de comunicación de RPC permite la comunicación entre un proceso cliente y uno servidor, generalmente localizados en máquinas diferentes. El flujo de comunicación funciona del siguiente modo: el proceso cliente realiza un llamado a un procedimiento que se encuentra definido en el proceso servidor. El proceso cliente se suspende, tomando lugar la ejecución del proceso servidor. Este último recibe un mensaje con toda la información necesaria para ejecutar el procedimiento. Una vez que el proceso servidor termina su ejecución, devuelve una respuesta al proceso cliente. Este flujo de comunicación es transparente para el usuario.

Este método implica un procedimiento, un mecanismo para la conversión de parámetros (*stub*) y una entidad de transporte para la conexión de red entre el cliente y el servidor. Se puede ver la esquematización de estos elementos en la figura 10.

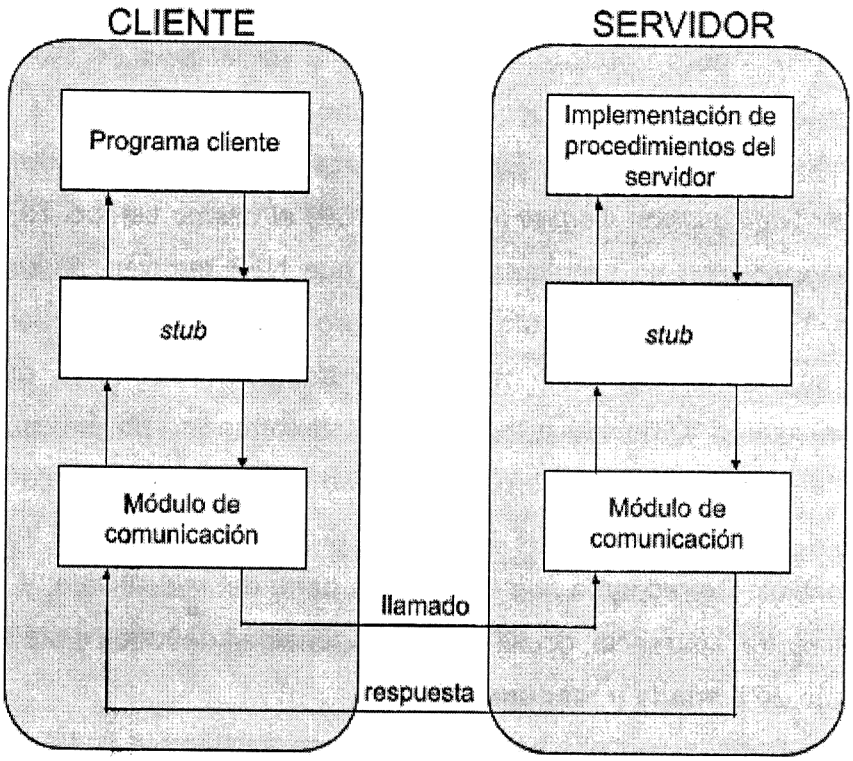


Figura 10. *Comunicación entre cliente y servidor RPC* (imagen propia)

De esta manera, la comunicación queda definida del siguiente modo:

A. El cliente se comunica con el servidor realizando una llamada a su *stub* asociado en su espacio de direcciones.
B. El *stub* del cliente empaqueta los parámetros en un mensaje e invoca a la entidad de transporte para enviar el mensaje al proceso servidor.
C. La entidad de transporte del servidor deriva el mensaje a su *stub* donde se obtienen los parámetros asociados y se realiza la llamada al procedimiento solicitado.

Estas etapas se llevan a cabo en el orden inverso al finalizar la ejecución del

procedimiento, retornando el control al cliente y finalizando el ciclo de la comunicación.RPC es un protocolo independiente de la capa de transporte: no necesita ningún tipo de información sobre cómo se intercambian los mensajes entre los procesos para funcionar. Es posible trabajar con *User Datagram Protocol* (UDP) y con *Transmission Control Protocol* (TCP) al mismo tiempo. RPC trabaja con los dos protocolos en simultáneo, por lo que NFS también. Si se tiene en cuenta que TCP es un protocolo con estado (*stateful*) y UDP sin estado (*stateless*), se podría decir que hay una contradicción con NFS, que es un protocolo sin estado y, debido a lo definido anteriormente, en principio, podría trabajar únicamente con UDP. Pero, en realidad, RPC se encarga de ocultar el hecho de que TCP es un protocolo de estado, lo que permite a NFS trabajar con ambos protocolos. Se recuerda que RPC forma parte del *middleware*, y, como tal, debe garantizar transparencia, ocultando heterogeneidad de redes y protocolos.

4.5.1 Mensaje de llamada y respuesta

El mensaje de llamada contiene parámetros que se encuentran codificados por el protocolo XDR. La cabecera del mensaje requiere solamente ocho campos. En la figura 11, se observa la estructura completa del mensaje de llamada.

El primer atributo del mensaje es el identificador de la transacción (XID). Es un valor único, típicamente, un entero de 32 *bits,* que identifica tanto la llamada como la respuesta. A modo de ejemplo, cuando un cliente realiza una llamada, RPC adhiere un identificador. El servidor, en caso de tener una respuesta, va a responder con el mismo XID para que el cliente pueda procesar la solicitud.

El segundo atributo, que se deja en cero intencionalmente, sirve para identificar la cabecera del mensaje de llamada. El tercer campo, RPC *version*, determina el formato que va a tener el mensaje, que varía de acuerdo a la versión de RPC y

NFS. Luego sigue el número de programa (para NFS es 100003), la versión del protocolo utilizada por ese programa (3 o 4) y, por último, el número de procedimiento (para la versión 3 de NFS, corresponde un 6).

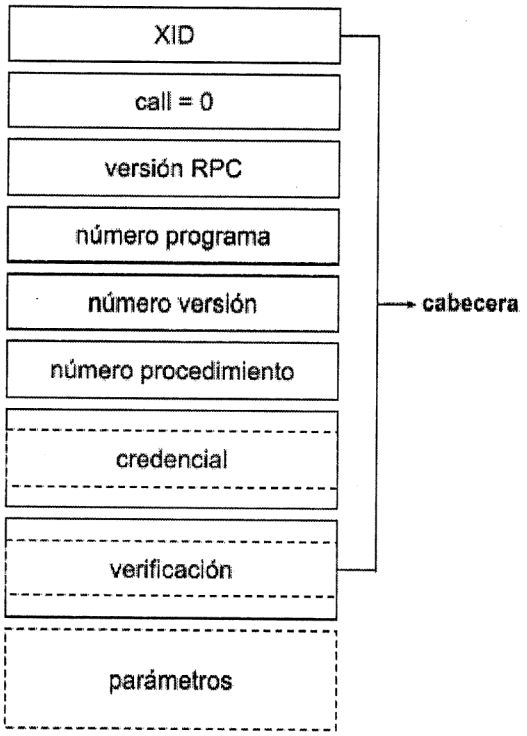


Figura 11. *Estructura de mensaje de llamada* (Callaghan, 1999: 56)

El campo credencial y verificación son de longitud variable y se utilizan para efectuar la autenticación. El campo credencial brinda la autenticación al cliente, con el servidor, y el campo verificación valida las credenciales.

Al final del mensaje, se encuentran los parámetros necesarios para que se ejecute el llamado.

En cuanto al mensaje de respuesta, se distinguen dos tipos: el mensaje de respuesta con errores (Figura 12.a) y el mensaje de respuesta con resultados positivos (Figura 12.b).

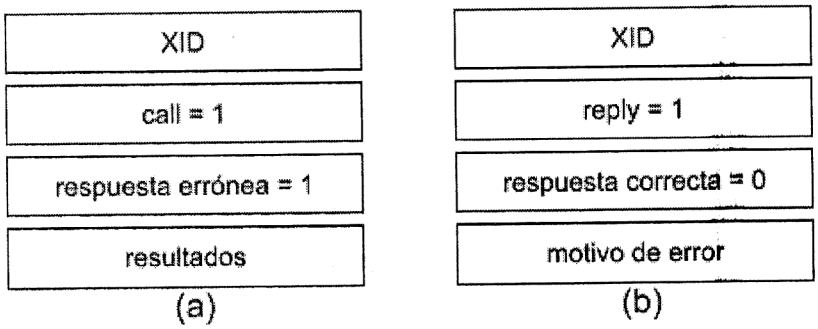


Figura 12. *Estructura y tipos de mensajes de respuesta* (imagen propia)

Los dos primeros campos de ambos mensajes de respuestas son iguales: el primer valor corresponde al identificador de la transacción y el segundo valor indica el tipo de mensaje. Como se trata de un mensaje de respuesta, el valor es uno.

El tercer campo varía de acuerdo al tipo de respuesta. Si la respuesta fue errónea corresponde el valor 1. Si la respuesta fue satisfactoria, corresponde el valor 0. Por último, el campo de datos, en el caso de errores, retorna el motivo del error. En caso satisfactorio, el último campo contiene los datos de la respuesta como por ejemplo, los datos leídos de un procedimiento *READ*.

El intercambio de mensajes RPC entre cliente y servidor es asíncrono. Esto quiere decir que el proceso que realiza la llamada no queda bloqueado esperando una respuesta, por lo que se puede continuar con otras tareas, en el mismo intervalo temporal.

4.6 External Data Representation (XDR)

Los datos que se intercambian en los mensajes RPC tienen que ser representados en un formato que sea extensible tanto para el cliente como el servidor.

XDR es un estándar para la descripción y codificación de datos binarios en un formato común e interoperable. Es útil para transferir datos entre diferentes arquitecturas de computadoras, sistemas operativos y diferentes lenguajes de programación. Como estándar, está definido en el RFC 4506 del año 2006. Todos los protocolos basados en el modelo de comunicación RPC, tales como NFS, utilizan XDR para codificar datos.

XDR asume que cada *byte* u octeto es portable y que cada tipo de datos (enteros, *strings*, punto flotante, etc.) puede ser representado como una secuencia de bytes portables. Para esto, utiliza un modo canónico de representación. Es decir, cada tipo de dato tiene una única representación en formato XDR; por ejemplo, para ordenar los bytes de un entero utiliza el método *big endian* (el *byte* más significativo primero) o, para representar valores punto flotante, utiliza el estándar de la IEEE 754.

Para representar todos los elementos, define un *block size* que tiene que ser múltiplo de 4 *bytes* (32 *bits*) de datos, como se representa en la figura 13.

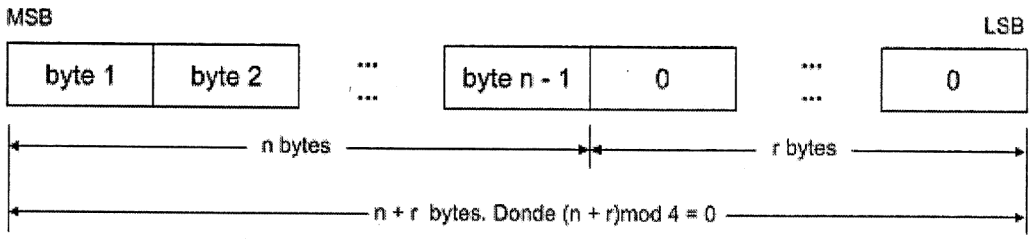


Figura 13. *Block size básico* (Srinivasan, 1995: 2)

XDR tiene las siguientes consideraciones:

- Los *bytes* son enumerados de 0 hasta n - 1.
- Los *bytes* son leídos o escritos teniendo en cuenta que el *byte* m siempre precede al *byte* m - 1.
- Si los n bytes necesarios para contener los datos no son múltiplo de 4, se agrega al final una cantidad r de ceros, para lograr que sea múltiplo de 4 bytes.

Se pueden representar los siguientes tipos de datos:

- Enteros.
- Enumeración.
- Booleanos.
- *Hyper* enteros y *Unsigned Hyper* enteros.
- Punto flotante.
- Punto flotante de doble precisión.
- Punto flotante de cuádruple precisión.
- Datos *opaque* de tamaño fijo y variable.
- String.
- Arreglos de tamaño fijo y variable.
- *Structure*.
- *Discriminated union*.
- Constantes.
- Tipos definidos.
- Void.
- Datos opcionales (variante de *union*).

Formalmente, RPC no usa directamente XDR: emplea como IDL (*Interface Definition Language*) una ampliación del lenguaje XDR que permite la definición de

procedimientos. Este lenguaje es conocido como RPCL (RPC *Language*) y es idéntico al lenguaje XDR, excepto por el hecho de que agrega definición al programa y su versión. Un IDL es un lenguaje utilizado para describir la interfaz de un componente de software. Este no es de ejecución sino declarativo, es decir, dice lo que existe, pero no cómo existe. (Montanaro 2014).

4.7 Network Lock Manager Protocol

Antes de entrar en detalle respecto del protocolo *Network Lock Manager* (NLM), primero se explicará qué es *File Locking* (bloqueo de archivos). Para esto, se planteará el siguiente interrogante: ¿qué sucede cuando más de un proceso intenta acceder a un mismo archivo para realizar operaciones de entrada y salida (escrituras y lecturas típicamente)? Cuando esto sucede ocurren problemas de sincronización e inconsistencia. Es por esto que se implementa un bloqueo de archivos en los sistemas operativos. Este mecanismo permite a un proceso bloquear un archivo entero o solo una pequeña región de *bytes* pertenecientes a este.

Existen dos mecanismos de bloqueo:

- *Advisory blocking*: este tipo de bloqueos es manejado cooperativamente entre los procesos involucrados, y depende del programador de cada uno de los programas en cuestión el solicitar y respetar dicho bloqueo. El procedimiento es similar a utilizar semáforos. Es decir, implementar una estructura de datos abstracta que, mediante operaciones atómicas, permitan a otro proceso el acceso a diferentes partes de un programa o recurso. Generalmente, las operaciones son *wait, init* y *signal.* Si el semáforo es de tipo binario, los valores que puede tomar son cero y uno. En este caso, solo un proceso puede acceder a un recurso a la vez. Los

semáforos que definen más de dos valores son llamados "multivariables" y pueden ser utilizados por varios procesos y varios recursos.

- *Mandatory blocking*: Una vez que un proceso adquiere un bloqueo mandatorio, el sistema operativo se encargará de imponer las restricciones correspondientes de acceso a todos los demás procesos, independientemente de si estos fueron programados para considerar la existencia de dicho bloqueo o no.

El protocolo *Network Lock Manager* (NLM) implementa el bloqueo de archivos (de advertencia), en un ambiente distribuido. Depende de otro protocolo, llamado *Network Status Monitor* (NSM) que está incluido en RPC. Permite notificar a clientes y servidores cuando se pierde el estado de bloqueo durante errores y problemas.

NLM asume un modelo de bloqueo que permite al proceso cliente bloquear una región de un archivo determinado por el *offset* y la cantidad de *bytes*.

La región bloqueada puede ser controlada por dos tipos de bloqueo. Uno de ellos es el bloque compartido, que permite a otros clientes leer contenido de un archivo bloqueado, pero no modificarlo. Es útil cuando múltiples clientes desean acceder al contenido de un archivo para leerlo. A su vez, permite a varios clientes bloquear otras regiones disponibles del archivo. En cambio, el bloqueo exclusivo impide a otros procesos leer y escribir en la región bloqueada.

Según Callaghan (1999), un protocolo de bloqueo de archivos debe proveer soluciones ante dos escenarios típicos:

- Pérdida de estado del servidor: cuando el servidor otorga el bloqueo a un cliente, tiene que mantener información del bloqueo: el propietario del bloqueo, el *offset*, la longitud del bloqueo y el tipo de bloqueo. Este registro

se almacena en la memoria del servidor (por razones de performance). Si el servidor se reinicia o presenta fallos inesperados, el cliente tiene que ser notificado de que el servidor se está recuperando (tolerancia a fallos), con el fin de restablecer el bloqueo.

- Pérdida de estado del cliente: si ocurre algún tipo de error en el cliente, la información del bloqueo que mantiene el servidor sigue impidiendo a otros clientes tener acceso a un bloque sobre el mismo archivo. Es importante que el servidor conozca el estado del cliente para prevenir este tipo de error.

NLM se vale del protocolo *Network Status Monitor* (NSM) para conocer y notificar estados entre el cliente y el servidor. Existen cinco tipos de procedimientos para operaciones de bloqueo de archivos:

- *TEST*
- *LOCK*
- *CANCEL*
- *GRANTED*
- *UNLOCK*

Al ser asíncronos, cada uno tiene una implementación de llamada y otra de respuesta. Anteriormente, se había dicho que NFS era un protocolo sin estados y se había discutido sobre qué sucedía si se utiliza un TCP (protocolo con estados). La conclusión que se obtuvo fue que RPC, en su rol de *middleware*, ocultaba ese comportamiento con estado. Pero ¿qué sucede al utilizar NLM? Para este caso, NFS pasa a ser un protocolo con estados. La principal ventaja de NLM es que los administradores de NFS pueden habilitarlo o deshabilitarlo. Simplemente se detiene el demonio "nfslockd".

4.8. Protocolo Mount

Este protocolo permite al servidor brindar acceso remoto de su sistema de archivos a un conjunto específico de clientes y provee al sistema operativo herramientas para acceder a rutas de archivos en servidor, obtener *file handles*, validar la identidad del usuario y verificar permisos de acceso. Los clientes utilizan este protocolo para obtener el primer *file handle*, lo cual posibilita acceder al sistema de archivos remoto.

Mount es un protocolo que no está integrado a NFS, funciona aparte. Mientras que el proceso servidor de NFS es a nivel *kernel*, el proceso correspondiente a *Mount* es a nivel usuario (*daemon*).

La implementación de este protocolo implica utilizar servidores con estados. Esto se debe a que el servidor tiene que mantener una lista de llamados relacionados con MOUNT de los clientes. No es necesario ni crítico mantener la lista de clientes. Se utiliza, específicamente, para comunicar a los clientes si ocurre un error en el servidor.

A continuación, se listan los procedimientos relacionados con el protocolo:

- NULL: no realiza ningún tipo de trabajo. Permite a los servicios de RPC testear los tiempos de respuesta del servidor. Es utilizado para verificar si el servidor responde.
- MNT: dado un *pathname*, devuelve el *filehandle* asociado. En sistemas UNIX, este procedimiento es usado a través del comando *mount*, por ejemplo, mount -t nfs server1:/home/shareddir.
- DUMP: retorna una lista de todos los clientes y los sistemas de archivos exportados.
- UMNT: quita un sistema de archivos montado del cliente y de la lista del

servidor.

- UMNTALL: es similar a UMNT, con la diferencia de que quita todas las entradas asociadas al cliente.
- EXPORT: devuelve una lista de los sistemas de archivos exportados con información detallada.
- EXPORTALL: Este procedimiento realiza lo mismo que EXPORT, por eso, fue eliminado en la versión tres.

Por otro lado, el protocolo MOUNT asume que el servidor mantiene una tabla que contiene la lista de clientes que montaron el sistema de archivos del servidor. Esto hace que MOUNT sea un protocolo que tiene que mantener el estado. En la figura 14, se observa el funcionamiento de la tabla con la lista de clientes:

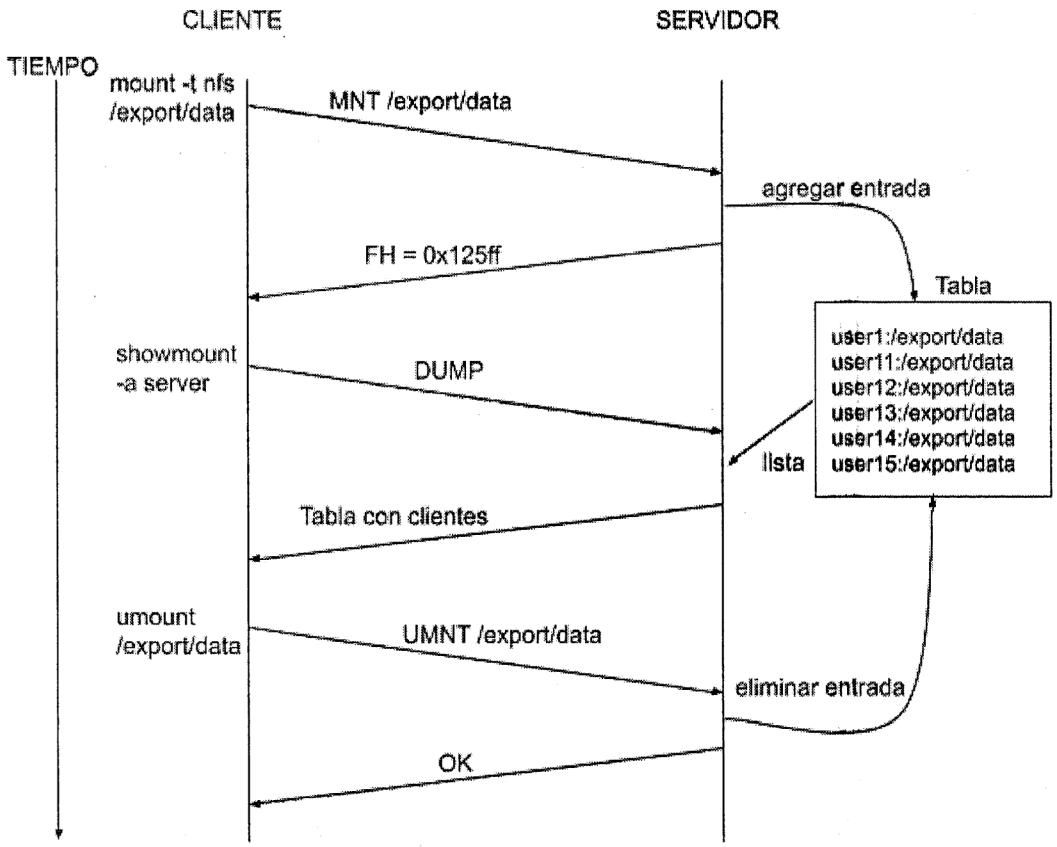


Figura 14. *Interacción cliente/servidor con protocolo MOUNT. Tabla clientes* (imagen propia)

Firma Estudiante:	Firma Docente Supervisor:	Firma docente tutor TAPTA:	Firma tutor Organizacional:

Capítulo 5. Cloud Computing

Existen múltiples definiciones de *cloud computing*, con distintos enfoques. Vaquero recopila veintidós definiciones, de diferentes autores, en una tabla (Vaquero, 2008: 52). Por otro lado, el *National Institute of Standards and Technology* (NIST), propone una definición estándar.

Dar una definición de *cloud computing* resulta complejo. Es por esto que, a partir de tres definiciones, se construirá una definición apropiada para el contexto del presente trabajo.

- Buyya propone que "*cloud computing* es un tipo de sistema paralelo y distribuido que consiste en una red de computadoras interconectadas y virtualizadas, se proveen dinámicamente presentándose como un único recurso basado en el acuerdo de nivel de servicio (*service level agreement* o SLA) que se negocia entre el cliente y el proveedor del servicio" (Buyya 2008: 2).
- A partir de la mencionada tabla con definiciones, Vaquero delimita el concepto de la siguiente manera: "*cloud computing* es un gran conjunto (*pool*) de recursos (*hardware*, plataformas de desarrollo y servicios) fácilmente utilizables y accesibles. Estos recursos pueden ser reconfigurados dinámicamente para ajustarse a la carga de trabajo permitiendo la utilización óptima de los mismos. Este *pool* de recursos son utilizados bajo el modelo 'pago por uso' ofrecido por el proveedor en el acuerdo de nivel de servicio" (Vaquero, 2008: 51).
- NIST establece que "es un modelo que permite acceso, bajo demanda, conveniente y en todo momento, a una red de recursos compartidos de cómputo configurables que pueden ser rápidamente utilizados con un

esfuerzo mínimo de administración o interacción con el proveedor (...)" (NIST, 2011: 2).

Teniendo en cuenta las delimitaciones previas, aquí definiremos *cloud computing* como un tipo de sistema distribuido, del tipo cómputo de altas prestaciones (*High Performance Computing*), que está conformado por una serie de recursos (físicos y virtualizados) que pueden ser accedidos de forma inmediata y en todo momento, de acuerdo al SLA. Estos recursos se ajustan y reconfiguran fácilmente de acuerdo a la carga de trabajo que estén procesando. Para el usuario, ese conjunto de recursos funciona como uno y su acceso y utilización son totalmente transparentes.

En la anterior definición, se omite el hecho de que los usuarios pagan por uso, de acuerdo al SLA. Esto se debe a que, formalmente no todos las infraestructuras *cloud* cobran por su uso. Este podría ser el caso hipotético de un laboratorio de simulación, que despliegue su propia infraestructura *cloud*, para contar con todo el software y recursos al momento de crear y ejecutar simulaciones complejas. En este caso, nadie pagaría por el uso de los recursos.

5.1 Arquitectura

En 2010, Zhang y Boutaba proponen una arquitectura ampliamente aceptada por diferentes autores. En la figura 15, se observa la estructura de cuatro capas, cuyos componentes se explicarán a continuación:

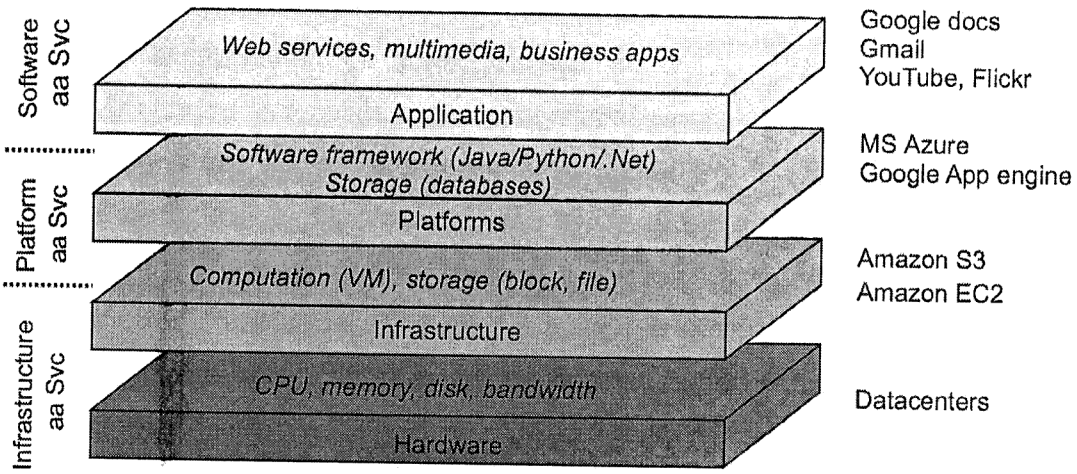


Figura 15. *Arquitectura cloud computing. Niveles de servicios* (Zhang y Boutaba, 2010: 9)

- Capa de hardware: esta capa es responsable de administrar los recursos físicos: servidores, *routers*, *switches*, sistemas de refrigeración y alimentación. Generalmente, esta capa se implementa en data centers que agrupan en forma de *stack* una cantidad considerable de servidores y otros recursos físicos. Algunas de las cuestiones de implementación de esta capa que hay que tener en cuenta son la tolerancia a fallos, la configuración de hardware, el manejo de gran volumen de tráfico de red, entre otras.
- Capa de infraestructura: también conocida como la capa de virtualización. Permite crear un conjunto de recursos de almacenamiento y cómputo, fraccionando los recursos físicos mediante tecnología de virtualización como Xen, KVM (*Kernel-based Virtual Machine*) y VMware. Esta capa es un componente esencial dentro de la arquitectura, ya que permite una

asignación dinámica de recursos que solo es posible mediante virtualización.

- Capa de plataforma: ofrece un nivel de abstracción alto para facilitar la el desarrollo e implementación de aplicaciones que corren en la nube mediante API's. Otro de los propósitos es minimizar la barrera en el momento de realizar el despliegue de una aplicación en una máquina virtual o contenedor. Por ejemplo, *Google App Engine* que opera sobre la capa de plataforma y provee una API para implementar aplicaciones web o móvil que utilizan lógica de negocio, base de datos y almacenamiento.
- Capa de aplicación: provee a los usuarios aplicaciones que corren en la infraestructura *cloud* y que, generalmente, son accedidas mediante interfaces *web*. A diferencia de las aplicaciones de escritorio tradicionales, estas aplicaciones pueden ajustarse al ritmo de trabajo automáticamente, a la disponibilidad, e incluso a los costos que el usuario maneja. Este es el caso de *Google Docs*: un conjunto de herramientas ofimáticas a las que se puede acceder desde un navegador web.

A partir de esto, se definen tres servicios que puede brindar una infraestructura *cloud*:

- SaaS: *Software as a Service,* asociado a la capa de aplicación.
- PaaS: *Platform as a Service,* enlazado a la capa de plataforma.
- IaaS: *Infraestructure as a Service,* relacionado con las capas de hardware e infraestructura.

5.2 Características

NIST define una serie de características esenciales de este tipo de sistemas:

- Servicio *on-demand* o autoservicio: un usuario puede realizar una petición

para que los diferentes recursos que usa ajusten su capacidad (ancho de banda, cantidad de memoria RAM, entro otros) sin necesidad de interacción con los administradores.

- Acceso amplio a la red: es posible realizar el acceso a la red de recursos a través de cualquier dispositivo (*tablets*, *notebooks*, *smartphones*, etc). Esto fomenta el acceso mediante dispositivos heterogéneos.
- Conjunto de recursos: los recursos se agrupan y coordinan entre sí para poder cumplir con la demanda de múltiples usuarios. Para los administradores de la infraestructura, existe una falta del conocimiento exacto de dónde se encuentran los recursos que están, por ejemplo, brindando procesamiento a determinado usuario. Como mucho, se conoce la ubicación y el nombre de los *datacenters.*
- Elasticidad: las capacidades de cómputo pueden ser elásticamente ajustadas y asignadas para escalar rápidamente o para cumplir con la demanda de recursos del usuario.
- Servicio cuantificado: los sistemas *cloud* pueden controlar y optimizar automáticamente el uso de recursos, realizando mediciones sobre los estos.

5.3 Tipos de despliegue

- Privados: la infraestructura está disponible de manera exclusiva para una única organización, que se encarga de realizar el despliegue, administrar y operar la infraestructura completa.
- Comunidades: en este caso, la infraestructura está a cargo de una comunidad exclusiva de diferentes usuarios de organizaciones que comparten los mismos objetivos, misión, visión y políticas. Puede estar

administrada por una o más organizaciones en simultáneo, o por terceros o una combinación de ambos.

- Público: en este caso, el uso está disponible para el público general. Pero es organizado, administrado y operado por una organización académica o gubernamental, con fines de lucro.
- Híbridos: está compuesta por dos o más de los anteriores tipos de nubes mencionados.

5.4 Virtualización

Cloud computing se vale de esta tecnología para hacer funcionar su infraestructura y poder ofrecer todas las funcionalidades mencionadas anteriormente.

Esta tecnología permite que una sola computadora contenga varias máquinas virtuales, cada una ejecutando un sistema operativo distinto. La principal ventaja de este método es que una falla en una máquina virtual no ocasiona que las demás fallen. El componente esencial para que un sistema operativo pueda ofrecer esta función se llama "hipervisor" o "monitor de máquinas virtuales" (MMV). El sistema operativo que se ejecuta encima del hipervisor se denomina "sistema operativo invitado" y el sistema operativo que se ejecuta en el hardware se denomina "sistema operativo anfitrión".

Típicamente, existen dos tipos de monitores:

- Hipervisores de tipo 1: Este tipo se caracteriza porque se instala directamente sobre el equipo, haciendo las funciones tanto de sistema operativo como de virtualización. La figura 16.a grafica esto. Algunos ejemplos son VMware ESXI Y XEN. Este tipo de hipervisores se ejecuta en modo *kernel*, por lo que la máquina virtual se ejecuta como un proceso de

usuario en modo de usuario. Si el sistema operativo invitado ejecuta una instrucción sensible, se produce una interrupción en el *kernel*. Si la instrucción la produjo el sistema operativo invitado, se prepara para ejecutar la instrucción. Si la instrucción la produjo un usuario de la máquina virtual, emula lo que haría el hardware real para procesar la instrucción.

- Hipervisores de tipo 2: este tipo se caracteriza porque debe ser instalado en un equipo que cuente con un sistema operativo previo. La figura 16.b. muestra este tipo de sistema. Algunos ejemplos son Vmware *player* y QEMU. Una máquina virtual (con su sistema operativo invitado) se ejecuta como un proceso en el host.

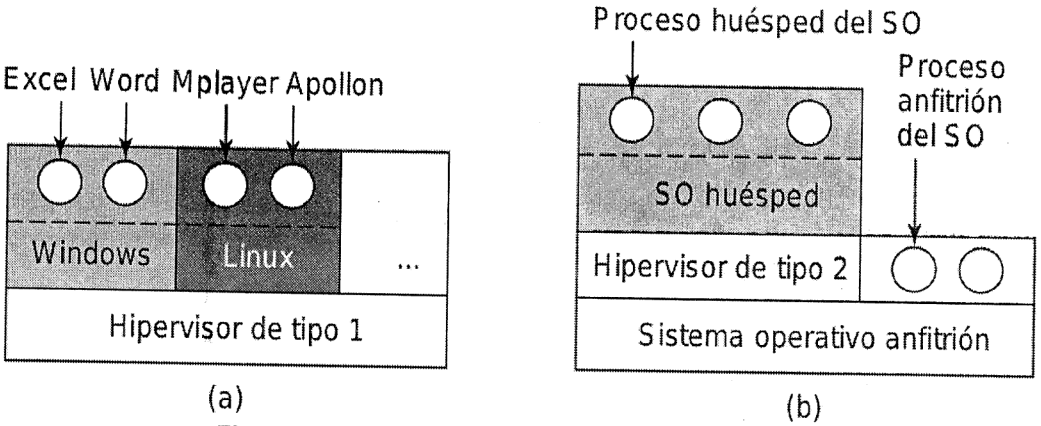


Figura 16. *Tipos de hipervisores* (Tanenbaum, 1992: 70)

Firma Estudiante:	Firma Docente Supervisor:	Firma docente tutor TAPTA:	Firma tutor Organizacional:

Capítulo 6. Simulación

Simular es una técnica que permite imitar y analizar las operaciones de varios tipos de procesos y comportamientos del mundo real en el tiempo. Robert Shannon propone el concepto de simulación "como el proceso de diseñar un modelo de un sistema real con el fin de realizar experimentos. El propósito es entender el comportamiento del sistema y evaluar varias estrategias sobre cómo opera dicho sistema" (Shannon, 1998: 7).

Los procesos o comportamientos de interés son conocidos como sistema. Nótese que se incluye la palabra interés, ya que no se puede modelar cada aspecto de la realidad al máximo nivel de detalle. Es por eso que es necesario realizar un análisis de lo que se quiere estudiar y simular, previo a comprobar que sea viable.

Los procesos y comportamientos de interés del mundo real que se quieren estudiar constituyen un sistema. Cuando se realiza un estudio sobre este sistema, se obtienen suposiciones acerca del funcionamiento. Estas suposiciones forman lo que se conoce como "modelo del sistema".

Según Sixto Ríos un modelo es "un objeto, concepto o conjunto de relaciones que se utiliza para representar y estudiar de forma simple y comprensible una porción de la realidad empírica" (Ríos, 1995: 23). Por otro lado, un sistema es un conjunto de entidades que se relacionan entre sí. Cada entidad y relación posee atributos. Los sistemas tienen un entorno con el que pueden intercambiar información (sistema abierto) o bien, pueden no hacerlo (sistema cerrado). Por otro lado, todos los sistemas tienen un objetivo.

Es posible construir un modelo del sistema utilizando métodos matemáticos como el álgebra, cálculo o probabilidades. Se dice que esta es una aproximación analítica. Cuando los sistemas son muy complejos, se utilizan simulaciones

computacionales para evaluar ese modelo. Los datos que se obtienen de la simulación son evaluados a fin de estimar las características reales del modelo.
Simular permite trabajar con el sistema sin la necesidad de interactuar, en principio, con el sistema real. Esto es útil cuando no se cuenta con el sistema real o es muy riesgoso interactuar con el mismo.

6.1 Ventajas y desventajas

Robert Shannon (1998) define una serie de ventajas que se obtienen al simular:

- Se puede testear nuevos escenarios sin la necesidad de contar con una gran cantidad de recursos.
- Permite explorar la implementación de nuevas reglas, políticas, organización estructural y flujos de información y comunicación.
- Es posible identificar cuellos de botellas en el sistema.
- También posibilita probar nuevas hipótesis.
- Se puede pasar las barreras del tiempo. Es posible realizar simulaciones que muestran el posible comportamiento de un sistema dentro de diez años, por ejemplo.
- Permite analizar posibles casos que no se tenían pensados o suponer escenarios de condiciones extremas.

Por otro lado, el mismo especialista propone una serie de desventajas:

- El modelado es una técnica que requiere práctica y experiencia. En muchos casos, requiere que sea aprobado por un experto. Shannon dice que modelar es un arte. La calidad del simulador depende de la calidad del modelado.
- Generar información de entrada para los simuladores puede ser tedioso. En algunos casos, si no se introduce información confiable, los datos de salida

pueden resultar erróneos.

- Los modelos de simulación no brindan soluciones por si solos. Tienen que ser utilizados como herramientas para evaluar el análisis de un sistema.

6.2 Proceso de modelado y simulación.

La gran mayoría de especialistas propone una serie de pasos para garantizar el éxito en el desarrollo de un simulador. En principio, el enfoque utilizado, resulta rudimentario y muy procedural, pero esto no quita que se pueda trasladar este mismo procedimiento a metodologías ágiles. Simplemente, se utiliza este procedimiento porque sienta las bases del proceso de modelado y simulación.

A continuación, se lista la serie de pasos necesaria y fundamental para llevar a cabo el proceso de simulación, de acuerdo con Shannon (1998):

A. Definición del problema: definir el objeto de estudio y su propósito.
B. Definición del sistema: definir el alcance y los límites del sistema a estudiar. Investigar cómo es el funcionamiento del sistema.
C. Formulación del modelo conceptual: desarrollar un modelo gráfico donde se reflejen las entidades del sistema, los parámetros de entradas, las relaciones y las variables del sistema.
D. Diseño experimental preliminar: determinar el tipo de pruebas a realizar sobre el modelo, teniendo en cuenta las variables de entrada.
E. Preparación de los datos de entrada: identificar y obtener los datos de entrada para el simulador.
F. Modelo computacional: formular el modelo conceptual en algún lenguaje de simulación o *framework*, por ejemplo, Netlogo, Repast, Arena, etc.
G. Verificación y validación: confirmar que el modelo funcione correctamente. En caso de no ser así, se reformula el modelo conceptual.

H. Diseño del experimento final: determinar cómo se ejecutará cada una de las ejecuciones de prueba especificadas en el diseño experimental.

I. Experimentación: probar el simulador en todos los casos posibles y asegurarse de que sea coherente o tenga la misma tendencia que el sistema real.

J. Análisis e Interpretación: analizar los datos provistos por el simulador. Obtener conclusiones.

K. Documentación: documentar modelos, sistemas, pruebas y resultados.

6.3 Tipos de sistemas

A continuación, se caracterizan dos de los tipos más importantes de sistemas:

- Discretos: en este tipo de sistema, las variables de estado cambian instantáneamente en diferentes puntos del tiempo; por ejemplo, se supone un banco. La variable de estado asociado puede ser la cantidad de clientes en el banco. Este valor cambia únicamente cuando un cliente entra al banco o cuando termina de realizar sus operaciones. En los diferentes puntos del tiempo, suceden eventos. Estos son definidos como una ocurrencia instantánea que puede cambiar el estado del sistema. Un aspecto importante de este tipo de sistemas, y, por lo tanto, de sus simuladores, es la presencia de un reloj de simulación que no posee unidades. Generalmente, se manejan mediante *ticks* y *subticks*.
- Continuos: en este tipo de sistemas, las variables cambian continuamente respecto del tiempo. Por ejemplo, un avión moviéndose en el aire es un sistema continuo: la velocidad y su posición cambian continuamente en el tiempo.

6.4 CloudSim

CloudSim es un *framework,* escrito en Java, de simulación generalizado y extensible, que permite el modelado, la simulación y la experimentación de diferentes infraestructuras y servicios de aplicaciones de *cloud computing*. Un ejemplo de utilización es la simulación de muchos centros de datos (Buyya, 2010). Permite construir simulaciones basadas en eventos discretos.

CloudSim basa su funcionamiento en eventos y entidades específicas que se representan como clases Java, que pueden ser heredadas o variadas para simular experimentos. Estas clases representan centros de datos, *hosts* físicos, máquinas virtuales, servicios a ejecutar en los centros de datos, servicios en la nube de usuarios, redes internas, centro de datos, consumo de energía de los *hosts* físicos y elementos de los centros de datos. Mediante eventos, se relacionan las diferentes entidades para enviar información y ejecutar tareas específicas de cada clase. Cada evento tiene un *tag* o constante numérica que lo identifica y es administrado por la cola de futuros eventos. Una vez que el evento se procesa, se agrega a la cola de eventos procesados.

CloudSim establece que el entorno de simulación es un sistema de *Data Centers* con la administración de un *broker* que asigna los recursos disponibles. El objetivo es procesar todas las instancias de *cloudlets* y eventos que se definen al inicio de la simulación. Al respecto, puede verse la figura 17:

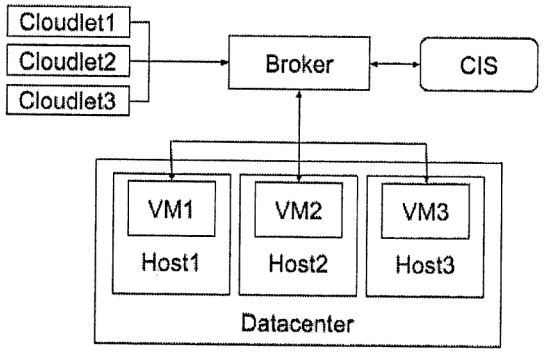


Figura 17. *Interacción entre entidades* (imagen propia)

El motor de simulación de *CloudSim* es la clase CloudSim, donde se definen cómo es el flujo de simulación y la interacción entre entidades. A su vez, este se encarga de iniciar y finalizar la simulación, una vez que los eventos y *cloudlets* fueron procesados y se han mostrado los datos de salida. La figura 18 presenta este esquema:

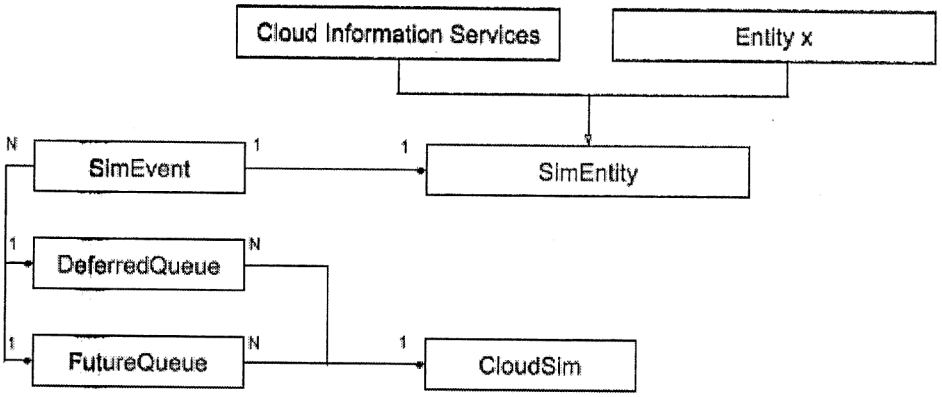


Figura 18. *Simplificación del core de CloudSim* (Buyya, 2010: 39)

6.4.1 Entidades y clases principales

A continuación, se listan las entidades básicas para el funcionamiento de una correcta simulación. Es necesario tener en cuenta que cada entidad posee un

nombre y un identificador (único).

Los *cloudlets* representan el funcionamiento de una aplicación basada en la nube. De esta forma, se logra abstraer el concepto de aplicación compleja en términos de procesamiento. Cada *cloudlet* contiene los requisitos de memoria necesaria, de forma que el *broker* asigne a una máquina virtual capaz de procesarlo, según su algoritmo de asignación. No hereda esto de *SimEntity.*

Un *broker* (o *Data center broker*) consulta al CIS (*Cloud Information Service*) y es el responsable de negociar entre el *cloudlet* y el proveedor de servicios *cloud* para la asignación de recursos.

CIS es una entidad que provee funciones de registro, indexado y búsqueda de recursos. Soporta dos tipo de primitivas: la primer es *publish*, que permite a las entidades registrarse a sí mismo en CIS. La segunda es *search*, que permite buscar entidades, recursos e información sobre otras entidades.

Vm (*Virtual Machine*) es la representación de una máquina virtual. No hereda de SimEntity.

Data Center es una entidad que simula la infraestructura o capa de hardware de los proveedores *cloud.*

Todas las entidades heredan de la clase *SimEntity.* Aquí es donde se define formalmente cómo procesa eventos cada entidad.

6.4.2 Metodología para agregar nuevas entidades

Como *framework* no solo posibilita construir simulaciones sino que permite agregar entidades nuevas, el principal desafío de agregar entidades y nuevas funcionalidades al *core* del *framework* está en no alterar o modificar el flujo natural de la simulación. Para esto, se realiza un análisis de las clases "*SimEntity*" y "*SimEvent*".

Al momento de definir una entidad, es necesario que esta herede de la clase *SimEntity*. Al hacer esto, se tienen que implementar tres métodos:

- startEntity(): indica como inicializa la entidad. Se pueden inicializar variables y mostrar mensajes de inicio.
- processEvent (SimEven event): recibe como parámetro un evento y se utiliza el *tag* de cada evento (evento.getTag()) para determinar qué debe realizar la entidad.
- shutdownEntity(): se debe especificar qué sucede cuando todas los *cloudlets* y eventos fueron procesados. Generalmente, se vacían colas y se muestra un mensaje.

En la figura 19, se observa cómo es el ciclo de ejecución de una entidad.

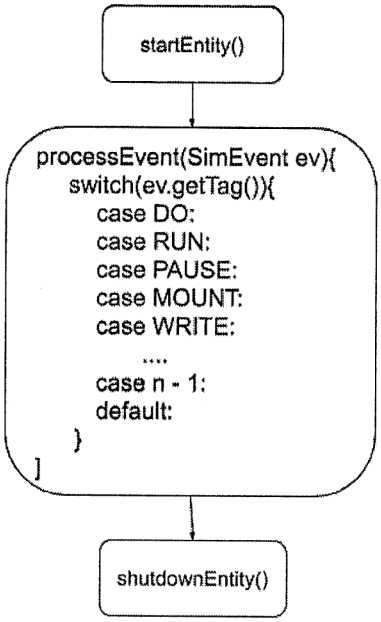


Figura 19. *Ciclo de ejecución de una entidad* (imagen propia)

Otro aspecto a tener en cuenta son los *tags*. Cada evento queda definido por una constante numérica. Para no interferir con nombres propios del *framework* y evitar utilizar un mismo valor de constante, se puede crear un nuevo archivo con constantes propias o definir cada constante en cada entidad con un número mayor a trescientos.

Al momento de mostrar mensajes en pantallas, no se puede utilizar el clásico método que provee Java para imprimir en pantalla. *CloudSim* provee un método llamado "Log.printLine()". Todo lo que se muestre con este método forma parte de la simulación.

Por último, para llamar a objetos y poder acceder a sus métodos dentro del entorno de simulación de *CloudSim*, es necesario obtener el identificador e invocarlo mediante el método "getEntity(int id)" de la clase principal *CloudSim*. De no ser así, los cambios que se quiera realizar sobre los objetos no se van a efectuar. Formalmente, se efectúan, pero por fuera del entorno de simulación.

Para enviar eventos de una entidad a otra, se utiliza el método *send* de la clase principal *CloudSim*. Este método recibe, en su forma completa, los siguientes parámetros:

- El identificador de la entidad que envía el evento.
- El identificar de la entidad que recibe el evento.
- El *tag* único del evento, por ejemplo, (MOUNT_DIR).
- El tiempo que tarda en enviarse el evento. Para esto, se puede utilizar el método clock() de la clase *CloudSim* o el método "eventTIme()" de la clase "SimEvent".
- Los datos: a través de este parámetro, se pueden enviar variables y objetos de cualquier tipo. Para poder acceder a los datos que se envían, es

necesario utilizar el método "getData()" de la clase "SimEvent". Además, se debe utilizar el procedimiento de *casting* para poder cambiar los datos de tipo *Object* a los específicos de cada clase.

Capítulo 7. Benchmark

Un *benchmark* es el acto de ejecutar un programa informático o un conjunto de programas con el objetivo de evaluar el rendimiento de un sistema, normalmente, llevando a cabo un número de pruebas estándar sobre él. Provee uno o varios métodos para comparar el rendimiento de varios subsistemas a través de distintas arquitecturas. Generalmente, suele utilizarse este término para definir los programas diseñados para llevar a cabo la técnica de *benchmark*.

Las evaluaciones o pruebas suelen involucrar características del hardware de una computadora; por ejemplo, el rendimiento de la operación en punto flotante de una CPU; aunque, también existen *benchmark* que son aplicables a sistemas de software.

Para poder realizar estas pruebas, cada *benchmark* genera un tipo particular de carga de trabajo en un componente o sistema. A partir de esto, se distinguen dos tipos de *benchmarks*:

- Sintéticos: crean programas especiales para imponer la carga de trabajo en un componente determinado, como, por ejemplo, un disco duro. La ventaja de esta categoría es permitir evaluar un único componente y no todo el sistema. Un ejemplo de este tipo de herramientas es IOR (*Interleaved or Random*).
- De aplicación: ejecutan programas reales sobre el sistema. Este tipo de *benchmark* ofrecen medidas confiables del rendimiento de todo el sistema. Un ejemplo de esta categoría es BTIO.

7.1 IOR (*Interleaved or Random*)

IOR forma parte de los ASCI *Purple Benchmarks*, desarrollados por LLNL (en español, Laboratorio Nacional Lawrence Livermore) para evaluar el rendimiento de

E/S. Permite medir las operaciones de lectura y escritura secuenciales y no incluye patrones de acceso aleatorios.

El tamaño del *buffer* de memoria y del archivo, y los tipos de archivo (compartido o único para cada proceso) pueden ser configurados mediante parámetros de línea de comandos. Además, se distingue de otros *benchmarks* en que provee distintas implementaciones para el mismo algoritmo, utilizando diversas interfaces de programación paralela, incluyendo POSIX, MPI-IO, HDF5 y NETCDF. La desventaja de IOR es que utiliza arreglos simples y unidimensionales y no incluye accesos a datos complejos y multidimensionales.

Un archivo compartido hace referencia a que todos los procesadores escriben en el mismo archivo en disco, pero en una "sección" distinta de él. Como las secciones son contiguas, no existe ninguna posibilidad de sobrescribir datos de un procesador vecino. Es necesario un sistema de archivos compartido al que todos los procesadores puedan acceder, como, por ejemplo, NFS. En cambio, en un archivo por proceso, cada procesador lee y escribe datos sobre su propio archivo en disco. Este mecanismo no requiere coordinación entre los procesadores; cada uno efectúa sus operaciones de E/S independientemente de todos los demás.

En la figura 20, se puede apreciar la estructura de un archivo compartido en IOR: esta está organizada como una secuencia de segmentos que representan los datos de la aplicación de una única variable de datos simulada, como, por ejemplo, la variable presión para la etapa de tiempo 1, 2, 3, etc.

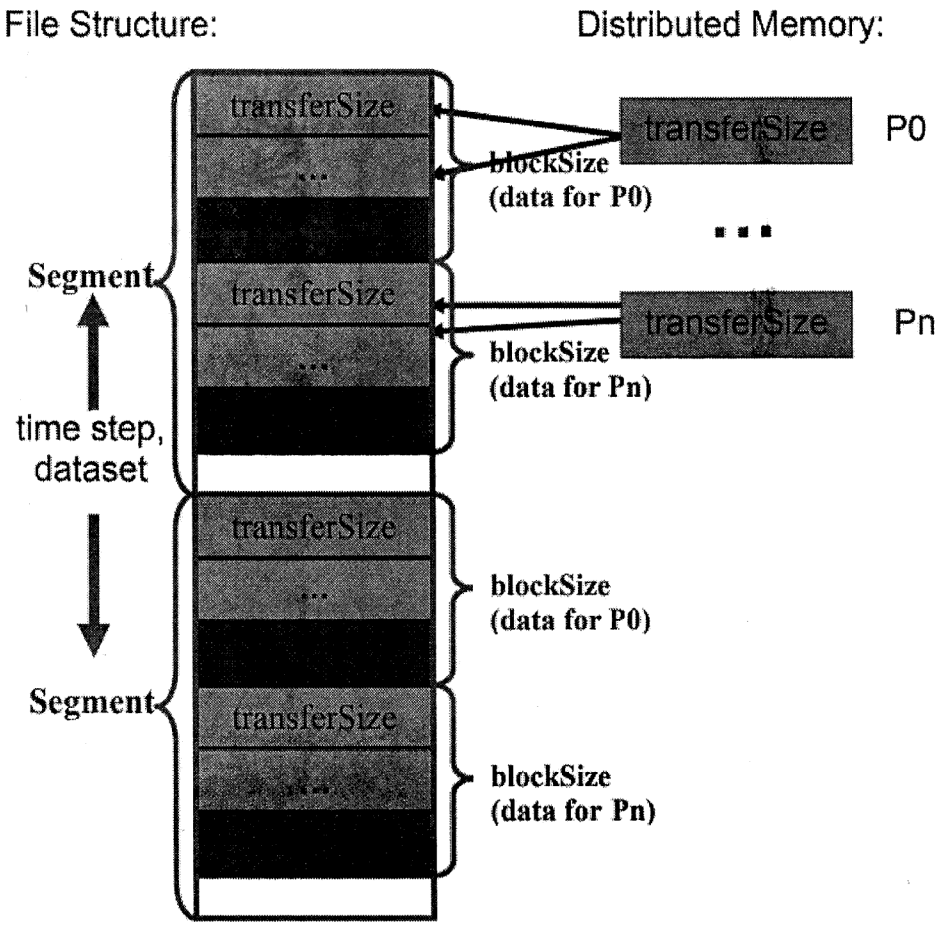


Figura 20. *Estructura de un archivo compartido* (Shan y Shalf, 2007: 5)

Cada segmento está dividido igualmente entre los procesos en unidades llamadas "bloques". El proceso con rango 0 (P0) recibe el primer bloque, el proceso con rango 1 (P1), el segundo, y así sucesivamente. Cada bloque es subdividido nuevamente en múltiples unidades de transferencia llamadas "*TransferSize*". Estos fragmentos corresponden directamente al tamaño de la transacción de E/S, que se define como la cantidad de datos transferidos de la memoria del

procesador al archivo, por cada llamada a una función de E/S. Para el caso de un archivo por proceso, la estructura de archivo es casi idéntica a la de la Figura 20. La única diferencia está en que cada proceso escribirá y leerá datos hacia y desde su propio archivo (por ejemplo, cada bloque estará empaquetado contiguamente en archivos separados), según postulan Shan y Shalf (2007).

Capítulo 8. Modelado

Para realizar el modelado de todo el sistema, se utilizaron máquinas de estado finito y diagramas UML. Para NFS, como protocolo, se utilizaron diagramas UML. Particularmente, para los protocolos XDR y RPC, se utilizaron máquinas de estado finito y, a su vez, diagramas UML (para definir interacción con NFS).

Por otro lado, se modeló el comportamiento de un sistema de archivos básico junto con la capa de abstracción provista por *Virtual File System*. Para esto se utilizaron, nuevamente, diagramas UML.

En último lugar, se modelo parte del comportamiento del *benchmark* sintético IOR. Se tuvieron en cuenta las operaciones leer y escribir, para el caso de un archivo compartido.

En el Anexo B, al final de este trabajo, se encuentra las definiciones y formalidades de las dos técnicas de modelado utilizadas.

8.1 Sistema de archivos

En la figura 21, se observa el correspondiente modelado del sistema de archivos básico. La abstracción utilizada para representar un sistema de archivos fue la estructura de tipo árbol con sus nodos; particularmente, la implementación de árbol con listas enlazadas. Por otro lado, se observa que la clase "SistemaArchivos" hereda de "SimEntity".

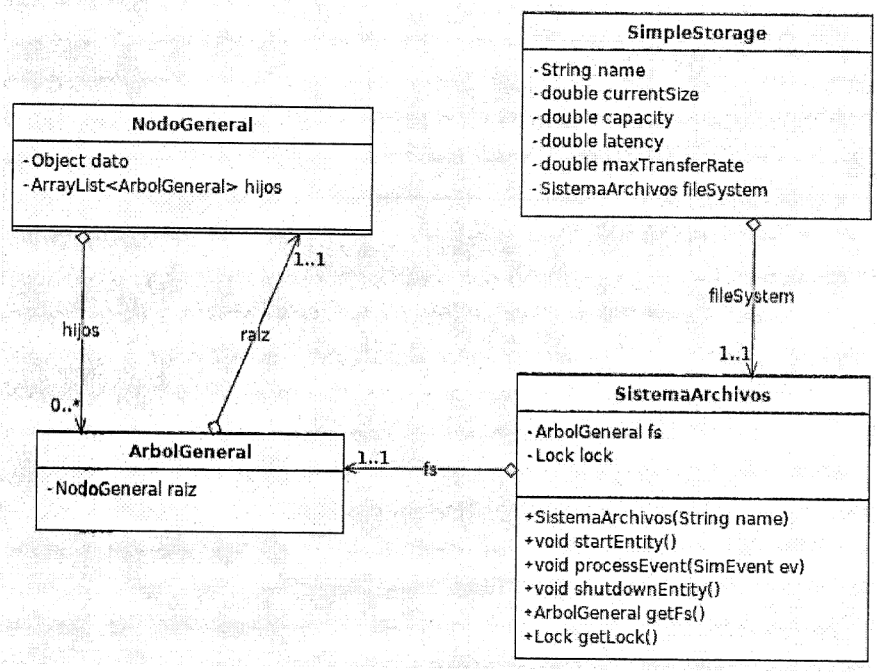


Figura 21. *Sistema de archivos UML* (imagen propia)

Se ocultan los métodos de las clases que no son entidades del *framework CloudSim* para que resulte más fácil de comprender el funcionamiento y para que el diagrama quede prolijo.

Este modelo representa un sistema de archivo físico. Por eso, existe una relación con la clase "SimpleStorage", que representa el comportamiento de un disco.

Además, se incorporó la noción de archivo y directorio. En la figura 22, se observa cómo es la estructura de cada componente. La clase "ArbolGeneral" puede agregar archivo y directorio de manera organizada. Algunos de los aspectos que se representan de los archivos son el nombre, su hora de creación, el tiempo que llevó crearlo, el tamaño y el contenido.

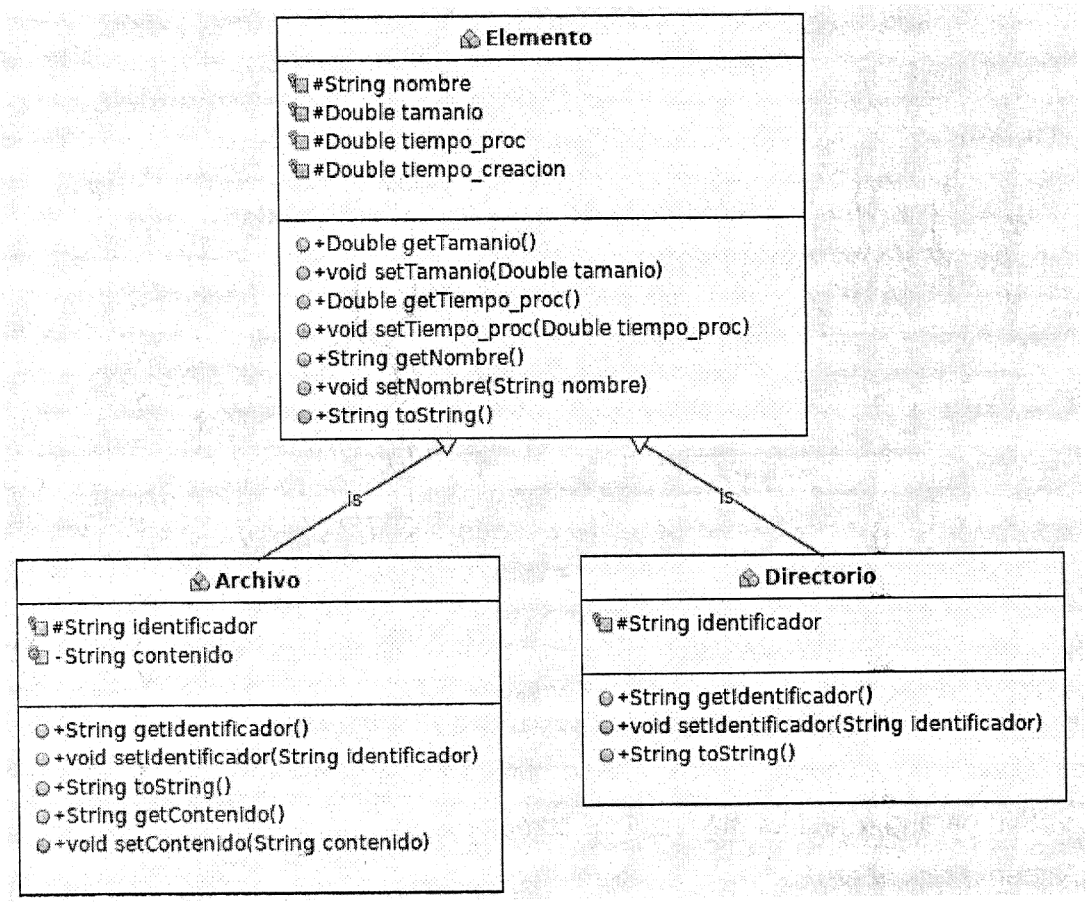


Figura 22. *Caracterización de archivo y directorio* (imagen propia)

8.2 Virtual File System

De manera muy similar, se implementa la capa de abstracción que permite trabajar con múltiples sistemas de archivos. También se utiliza una representación de una estructura de árbol. Cada sistema de archivo que es montado tiene una referencia en VFS y, luego, en el sistema de archivos físico.

Se centró la implementación de VFS en la máquina virtual cliente. Esto se debe a

que esta máquina es la que monta el sistema de archivos NFS. Al igual que la clase "SistemaArchivos", "VirtualFileSystem" hereda de "SimEntity".

En la figura 23, se puede observar el correspondiente modelo UML y cómo es la interacción con la máquina virtual correspondiente al cliente NFS.

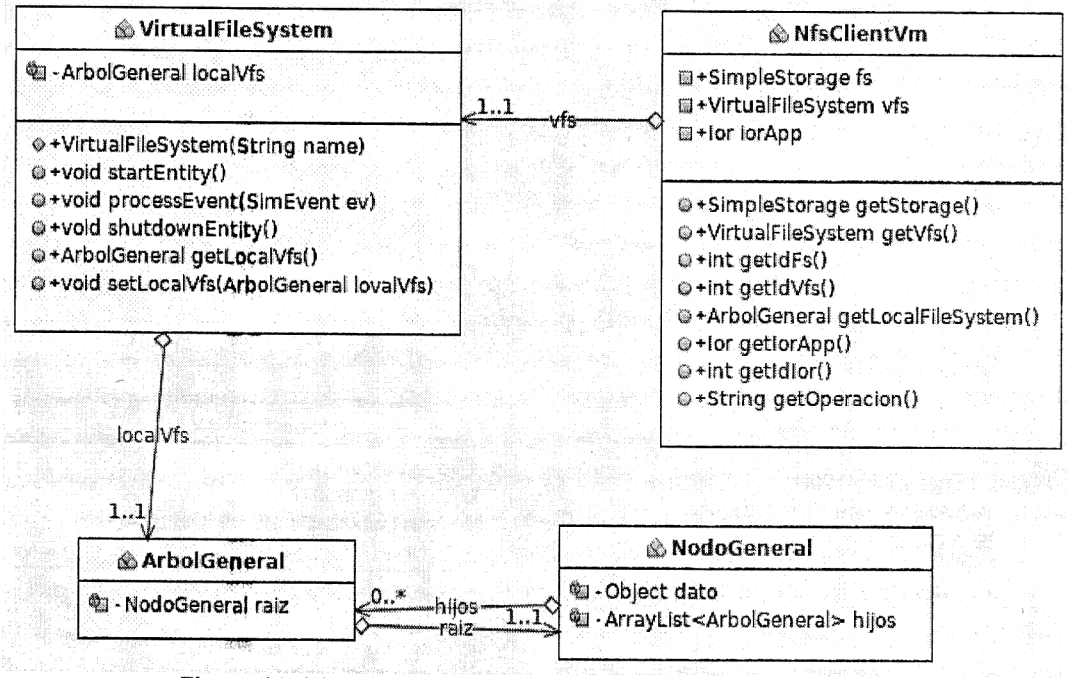


Figura 23. *Virtual File System. Relación con cliente* (imagen propia)

8.3 Máquinas virtuales

CloudSim ofrece una implementación de máquina virtual. Pero, para no alterar la estructura original del *framework* y con el fin de caracterizar NFS, se implementaron dos versiones de máquina virtual: un cliente NFS y otra correspondiente al servidor NFS. Ambas máquinas heredan y sobrescriben métodos de la clase "Vm". En la figura 24, se puede observar el mecanismo de herencia implementado. Cabe destacar que las máquinas virtuales en *CloudSim*

no son entidades: simplemente, son contenedores de otros objetos.

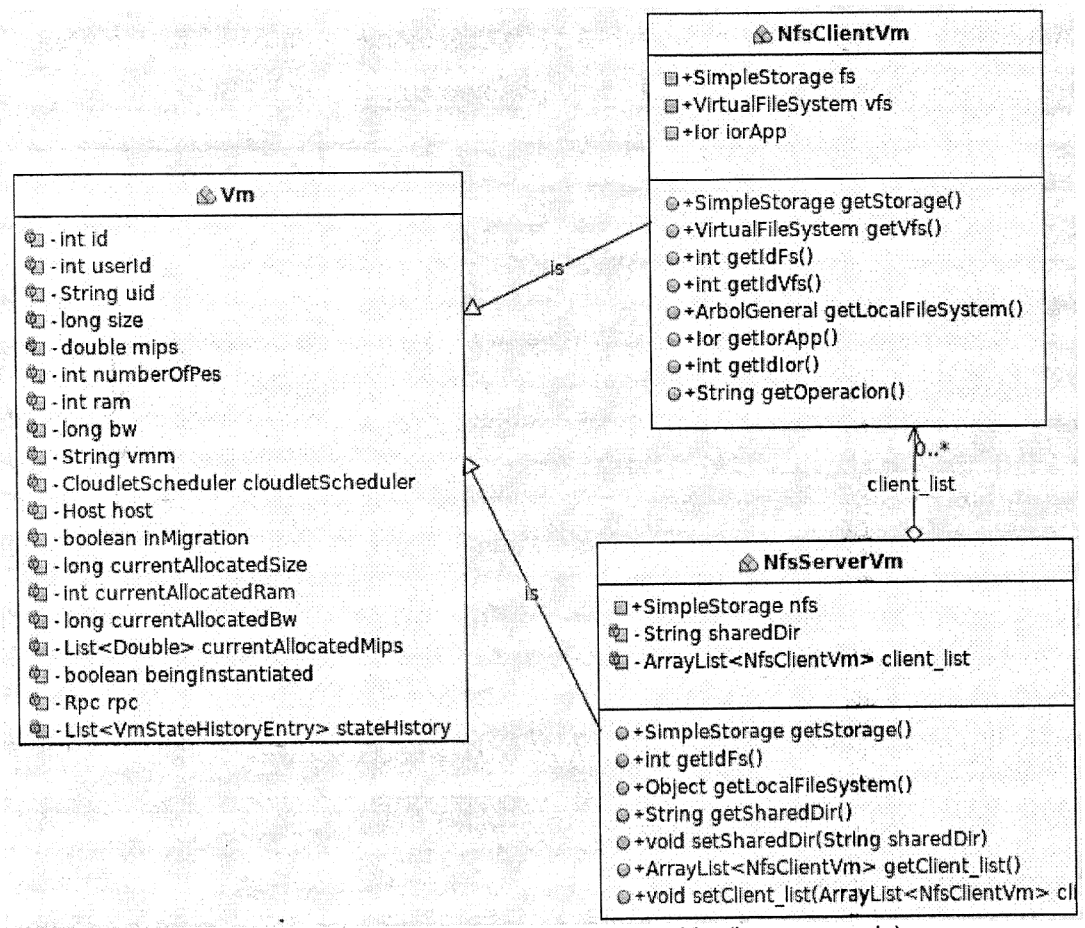


Figura 24. *Máquinas virtuales cliente y servidor* (imagen propia)

El modelo de la clase "NfsServerVm" posee una lista de todos los clientes que montaron el sistema de archivos del servidor. En principio, la idea de implementar esta lista sirve para modelar el comportamiento de la tabla de clientes creada por el protocolo MOUNT. En segundo lugar, sirve a fines de simulación: facilita la comunicación e interacción entre servidor y clientes.

Firma Estudiante:	Firma Docente Supervisor:	Firma docente tutor TAPTA:	Firma tutor Organizacional:

En este punto de modelado, comienza a caracterizarse NFS. Al representar las máquinas virtuales "cliente" y "servidor NFS", se consigue modelar parte del comportamiento del protocolo. En particular, un cliente NFS, por un lado, y, por otro, un servidor, que corre el proceso "nfsd". Además, cada máquina tiene una relación a RPC, XDR y *Lock Manager*. MOUNT es propio del cliente.

8.4 RPC y XDR

Para este protocolo, se realizó el modelado con máquinas de estado de Moore. En la figura 25, se observa el comportamiento de RPC servidor.

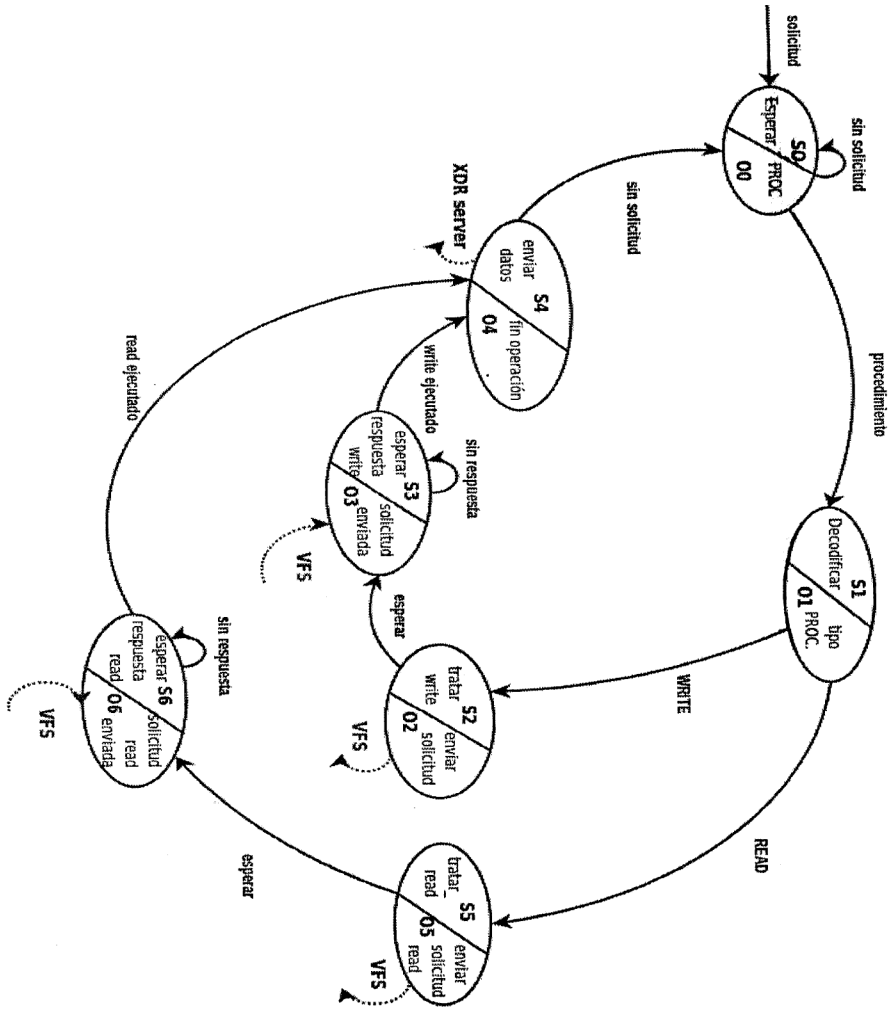


Figura 25. *Máquina de estados finito de RPC servidor* (imagen propia)

Un aspecto importante a destacar es que "esperar" no hace referencia a la tarea explícita de esperar una respuesta de una entidad. En este caso, "esperar" hace referencia a realizar otras actividades de procesamiento, propias de RPC, hasta

Firma Estudiante:	Firma Docente Supervisor:	Firma docente tutor TAPTA:	Firma tutor Organizacional:

que llegue un mensaje de VFS.

Para el caso de RPC cliente, el diagrama resulta similar. La principal diferencia está en el llamado a otras entidades: mientras el servidor realiza el llamado a VFS y a XDR servidor, el cliente realiza llamado a XDR cliente y a la entidad IOR (aplicación). No se incluye la figura de RPC cliente para no ser repetitivo (lo mismo sucede con XDR cliente)[1].

XDR servidor y cliente están implementados a nivel código, dentro de RPC, a través del método "processEvent(SimEvent ev)". Este método utiliza las etiquetas o *tags* de *CloudSim*. Cada etiqueta es un estado XDR o RPC. No obstante, se muestra a continuación la correspondiente máquina de estados, en la figura 26.

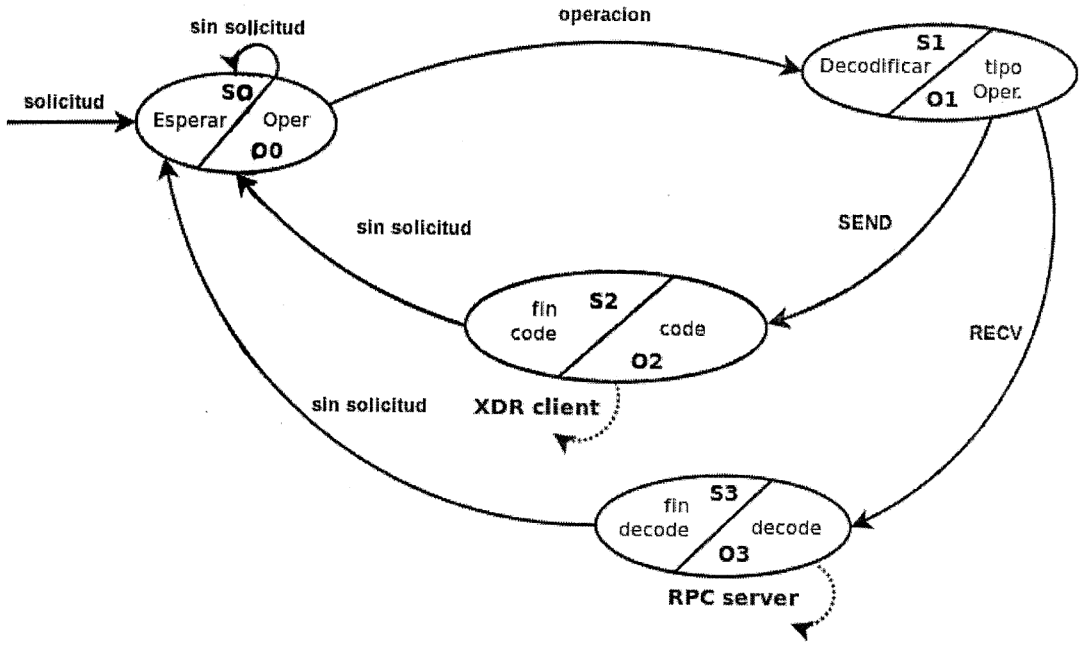


Figura 26. *Máquina de estados finito de XDR servidor* (imagen propia)

[1] Al respecto, puede consultarse el anexo C, en el presente trabajo.

Firma Estudiante:	Firma Docente Supervisor:	Firma docente tutor TAPTA:	Firma tutor Organizacional:

En este caso, sucede lo mismo que con el modelo de RPC. XDR cliente y servidor son muy similares, pero con funciones espejadas. En XDR server, cuando hay una solicitud SEND, se hace un envío de datos a XDR cliente y el estado está relacionada con decodificar. En el servidor, sucede lo inverso: cuando hay una solicitud SEND, se codifican los datos y se envían al servidor.

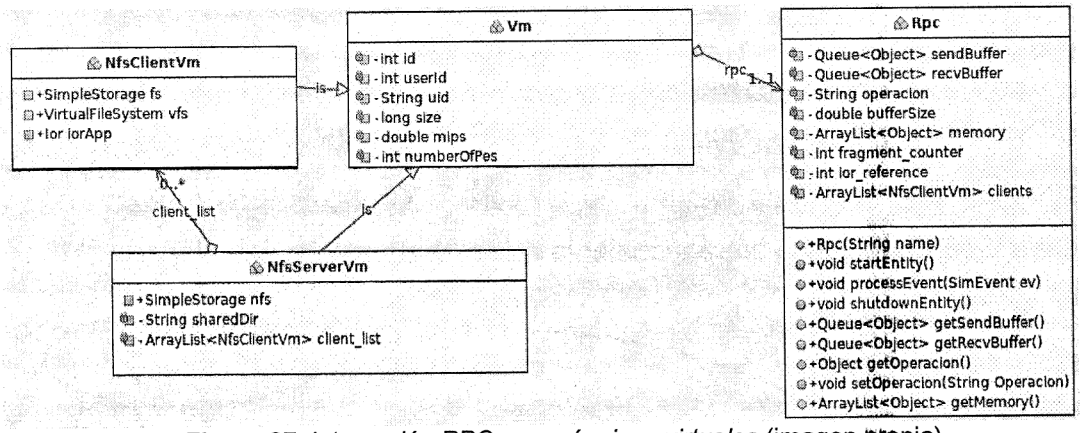


Figura 27. *Interacción RPC con máquinas virtuales* (imagen propia)

Por otro lado, teniendo en cuenta que se debe implementar en el paradigma orientado a objetos, en la figura 27, se observa cómo interactúan RPC y XDR con el cliente y servidor modelado.

Otro aspecto a recalcar de la clase RPC es que cuenta con una cola de solicitudes. Específicamente, una cola para las operaciones recibir (RECV) y otra para las operaciones enviar (SEND) de XDR, tanto para el cliente como el servidor.

8.5 Lock Manager

De este protocolo, se modelaron únicamente dos procedimientos: LOCK y UNLOCK. En la figura 28, se muestra cómo es la estructura de su clase.

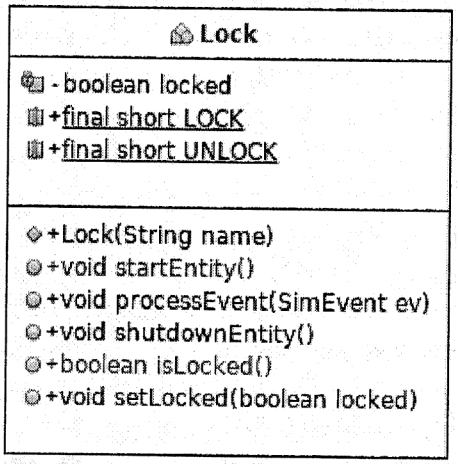


Figura 28. *Modelo del protocolo Network Lock Manager* (imagen propia)

Este protocolo no se relaciona con otras entidades del simulador mediante los tipos de asociaciones de objetos conocidas (agregación y composición). Simplemente, interviene mediante eventos de simulación en el programa principal. Cuando una operación de lectura o escritura sucede, se envía un evento a la entidad "SistemaArchivos" para bloquear el archivo en el que se está realizando la operación. Una vez que la operación termina, se envía un evento de respuesta a la entidad "Lock" para que libere el archivo.

8.6 Mount

Para Mount, se modelaron dos procedimientos: MOUNT y EXPORT. Inicialmente, este protocolo no se iba a implementar en el simulador. Entonces, se implementaron los modelos obtenidos hasta ese momento.

Estudiando y analizando nuevamente su comportamiento, se deduce que es un protocolo clave, al menos, al menos, en cuanto a las funcionalidades que ofrecen

los dos procedimientos seleccionados. Es por esto que, en el inicio, no se modeló y que, luego, se implementó directamente.

La implementación está separada dentro de dos entidades: el procedimiento MOUNT forma parte de la entidad "VirtualFileSystem" y el procedimiento EXPORT forma parte de la entidad "SistemaArchivos".

8.7 IOR

Para poder obtener métricas comparables a las de un sistema real, se modela el comportamiento del *benchmark* IOR para un archivo compartido. En la figura 29, se observa el correspondiente diagrama UML.

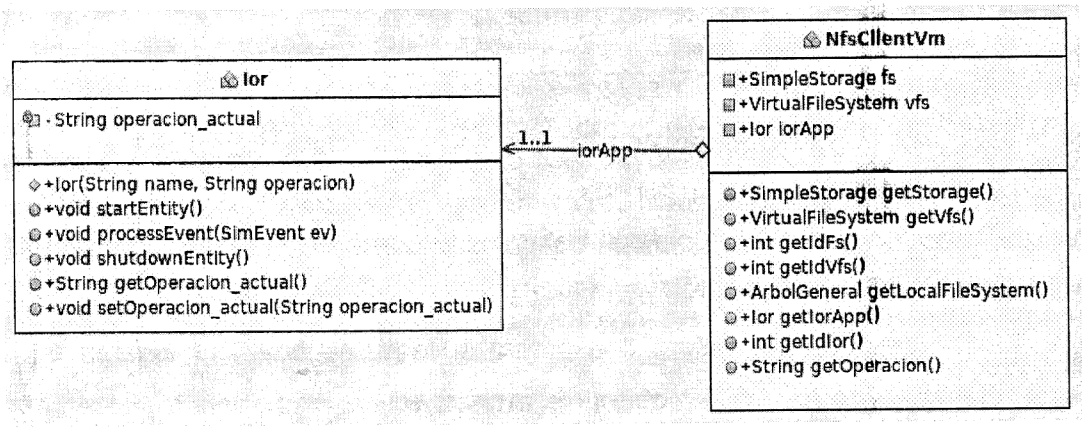


Figura 29. *IOR benchmark* (imagen propia)

De IOR, se modelaron los parámetros básicos:

- Tipo de operación: *write* y *read*.
- BlockSize.
- SegmentCount
- TransferSize.

Otro aspecto a destacar en IOR es que, al ser una aplicación que se ejecuta en un

cliente o conjunto de ellos, se relaciona únicamente con la clase "NfsClientVm". Además, se obtuvo la relación matemática con la que IOR devuelve sus resultados. Esto se realizó estudiando el código fuente del *benchmark*. La relación está dada por los siguientes componentes:

$$AnchoDeBanda = \frac{aggregateFileSize}{TiempodeOperación}$$

$$= \frac{BlockSize * SegmentCount * numberOfprocesors}{TiempodeOperación}$$

Figura 30. *Relación ancho de banda utilizada por IOR* (imagen propia)

8.8 Nuevas entidades

A modo de resumen, en el figura 31, se muestran las entidades nuevas que se agregaron con la intención de modelar y caracterizar NFS y sus protocolos, junto con un sistema de archivos y la capa de abstracción VFS, así como el *benchmark* IOR.

En el gráfico, se observa cómo heredan las nuevas entidades con la clase "SimEntity". Esto posibilita que las entidades puedan intercambiar eventos y formen parte del entorno de ejecución del simulador.

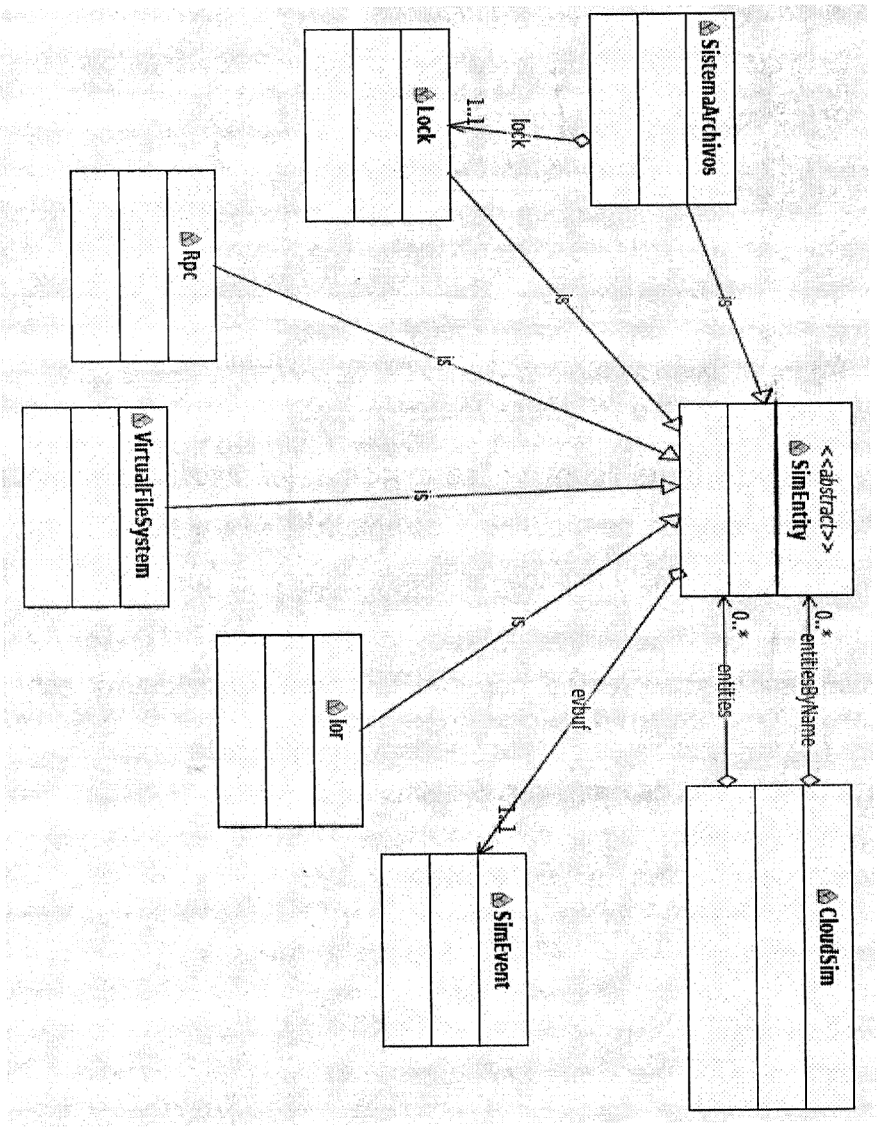


Figura 31. *Relación nuevas entidades con core de CloudSim* (imagen propia)

Firma Estudiante:	Firma Docente Supervisor:	Firma docente tutor TAPTA:	Firma tutor Organizacional:

Capítulo 9. Implementación

Para poder realizar el desarrollo, se utilizaron las siguientes herramientas:

- *CloudSim* versión 3.
- *Netbeans 8.2*, como entorno de desarrollo integrado.
- *OpenJDK Java 8 full runtime environment*, como entorno de ejecución del lenguaje de programación JAVA.
- *OpenJDK Java 8 development kit*, que brinda herramientas de compilación y ejecución de programas JAVA.

El proceso de instalación y configuración de cada herramienta se describe en el Anexo D, al final del presente trabajo.

Para pasar de un modelo formal al código JAVA, se crea un nuevo proyecto Java, con dependencias y referencias al *framework CloudSim.* Cada proyecto puede contar con uno o varios paquetes. En este caso particular, se crearon cuatro paquetes.

En primer lugar, el paquete "Estructuras" contiene las clases "ArbolGeneral" y "NodoGeneral" que se utilizaron para crear los diferentes sistemas de archivos del proyecto. En segundo lugar, el paquete "Examples" contiene los programas principales que ejecuta el usuario. Está compuesto por las clases "NfsCloudSimExample1" y "NfsCloudSimExample2". Cada una corresponde a la ejecución de un *write* y un *read*, respectivamente, con los resultados correspondientes."Examples" hace referencia al paquete "org.cloudbus.cloudsim.examples", que contiene los ejemplos básicos provistos por el *framework.* Los últimos dos paquetes contienen las entidades y clases fundamentales para que la aplicación funcione: por un lado, el paquete "synthetic"

contiene las clases "Archivo", "Directorio" y la implementación de las máquinas virtuales cliente y servidor NFS; por otro lado, el paquete "org.cloudbus.cloudsim.core" contiene la implementación de cada entidad nueva que se agrega. Se distinguen los siguientes archivos ".java":

- "Ior".
- "Lock".
- "Rpc".
- "SistemaArchivos".
- "VirtualFileSystem".
- "SimpleStorage".

En la sección 6.4.2, se especificó cómo agregar nuevas entidades al *framework*. La parte más importante era sobrescribir el método "processEvent (SimEvent event)". A continuación, se muestra, a modo de ejemplo, el contenido de este método que corresponde a la entidad Ior:

```
@Override
public void processEvent(SimEvent ev) {
switch (ev.getTag()) {
case SimEvent.SEND:
CloudSim.resumeSimulation();
break;
case SimEvent.CREATE:
CloudSim.resumeSimulation();
break;
case SimEvent.ENULL:
```

```
CloudSim.resumeSimulation();
break;
case SimEvent.HOLD_DONE:
CloudSim.resumeSimulation();
break;
case lor.OPERACION:
setOperacion_actual((String) ((Object[]) ev.getData())[0]);
this.setBlockSize((Double) ((Object[]) ev.getData())[1]);
this.setTransferSize((Double) ((Object[]) ev.getData())[2]);
this.setSegmentCount((int) ((Object[]) ev.getData())[3]);
this.setTipo_regresión((String) ((Object[]) ev.getData())[4]);
Log.printLine("Operacion: " + getOperacion_actual());
CloudSim.resumeSimulation();
break;
case RESULTS:
Double total_time = this.time();
this.output(total_time);
CloudSim.resumeSimulation();
break;
default:
Log.printLine("Error");
break; }}
```

Figura 32. *Ejemplo de métodos "processEvent"* (imagen propia)

Las constantes SEND, CREATE, ENULL y HOLD_DONE son propias de la clase padre "SimEntity" y tienen ya un comportamiento definido. Para los fines de este

proyecto, no fue necesario a sobreescribir esas operaciones.

Luego, de las cuatro operaciones descriptas, siguen otras dos:

- OPERACION: recibe un evento con los datos de la operación a realizar y los parámetros del *benchmark* IOR.
- RESULTS: este evento muestra los resultados finales de ejecutar el *benchmark*. Se vale del método "output" para realizar todos los cálculos y mostrar los resultados.

9.1 Obtención de tiempos

Otro aspecto fundamental de la implementación es la incorporación de un modelo estadístico obtenido del entorno virtual ejecutado en *Amazon Web Services*. En el Anexo E, se detalla cómo es el despliegue de este escenario.

Lo significativo de este entorno es la cantidad de nodos con los que se trabaja y los tamaños de archivos que se manejan. El primer escenario es con cinco nodos, un servidor y cuatro clientes. El segundo escenario es con diez nodos, un servidor y nueve clientes. Por otra parte, se trabajó con tamaños de archivos, generados por el *benchmark*, de uno, dos, tres y cuatro *gigabytes*.

El objetivo es obtener los tiempos máximos de lectura y escritura para introducirlos en el simulador. En la tabla 2, se observan los tiempos para cinco y diez nodos, respectivamente.

	Tiempo (segundos)			
	5 Nodos		10 Nodos	
	write	*read*	*write*	*read*

1G	20,54	16,08	24,53	19,38
	21,96	15,83	22,73	17,12
	22,06	15,87	24,49	20,56
	23,9	15,83	26,88	20,25
	19,28	16,54	23,79	20,99
Promedio	21,548	16,03	24,484	19,66
2G	46,6	32,67	46,24	32,56
	44,65	34,53	46,49	33,03
	47,19	32,93	47,7	32,61
	48,16	32,59	53,38	38,66
	48,01	33,26	46,04	37,38
Promedio	46,922	33,196	47,97	34,848
3G	68,14	49,37	71,25	58,92

Firma Estudiante:	Firma Docente Supervisor:	Firma docente tutor TAPTA:	Firma tutor Organizacional:

	70,93	49,79	72,95	59,64
	71,27	51,75	71,53	57,99
	71,26	49,81	70,35	60,36
	72,33	50,23	79,97	58,51
Promedio	70,786	50,19	73,21	59,084
4G	95,96	68,18	93,33	67,43
	92,77	66,25	96,15	66,19
	93,51	66,14	93,36	66,14
	95,14	69	90,31	66,12
	95,9	68,34	93,5	66,13
Promedio	94,656	67,582	93,33	66,402

Tabla 2. *Tiempos para 5 y 10 nodos correspondientes a diferentes tamaños de archivo*
(tabla propia)

En las figuras 33 y 34, se observan los mismos resultados de la tabla 2, de manera gráfica. Claramente, se observa que las operaciones de escritura requieren mayor tiempo que las de lectura.

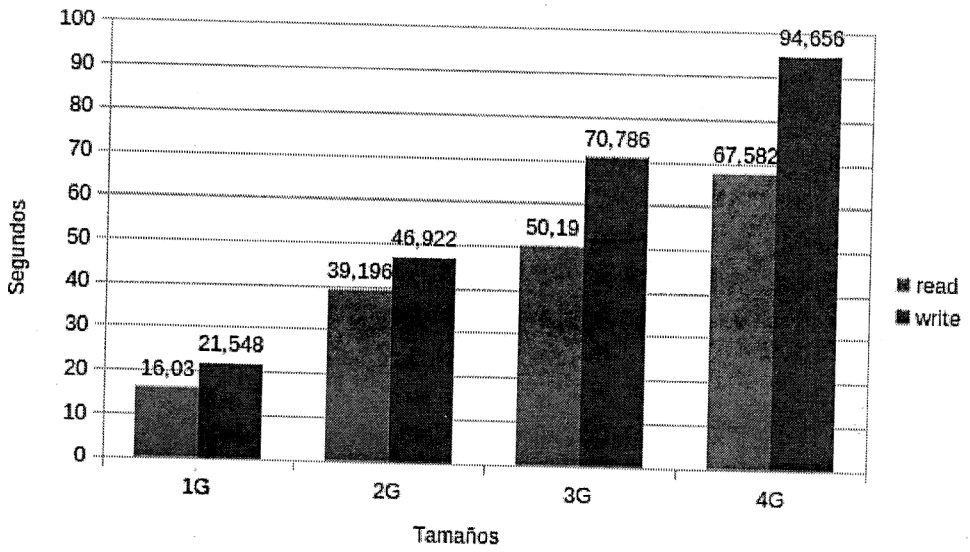


Figura 33. *Tiempos promedios de read y write para 5 nodos* (imagen propia)

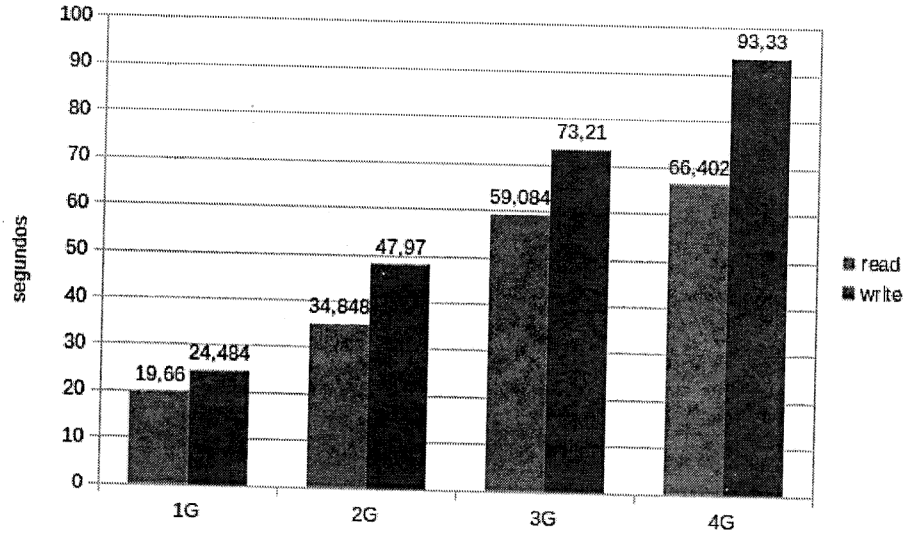


Figura 34. *Tiempos promedios read y write para 10 nodos* (imagen propia)

Firma Estudiante:	Firma Docente Supervisor:	Firma docente tutor TAPTA:	Firma tutor Organizacional:

Si se comparan los tiempos del escenario con cinco y diez nodos por operación, se observa que la variación es muy poca: de tres a cuatro segundos, en la mayoría de los casos. En principio, se podría decir que esta variación tiene que ver con la cantidad de nodos. Sin embargo, esta propuesta es inadecuada, ya que la variación tiene que ver con la elasticidad y disponibilidad de recursos de la nube de *Amazon*, en la capa gratuita. Si se observan los tiempos promedios de dos *gigabytes* (para un *read*) y de cuatro gigabytes (para un *write*) con diez nodos, son mayores a los correspondientes con cinco nodos. Este aspecto no es un detalle menor, pues tiene que ver con una de las características principales de *cloud computing*: la elasticidad.

9.2 Modelo estadístico

Para obtener un modelo matemático que cumpla con los tiempos obtenidos en la sección anterior, se realiza un análisis de regresión, particularmente, lineal y exponencial. El usuario puede optar por realizar ejecuciones con una u otra técnica, de acuerdo a sus necesidades.

El análisis de regresión es una técnica estadística para el modelado y la investigación de la relación entre dos o más variables. De esta forma, es posible obtener, matemáticamente, una función que represente el comportamiento o relación entre las variables. En este caso particular, las variables son el tamaño de archivo (independiente) y el tiempo (dependiente).

En las figuras 35 y 36, se observa el diagrama de dispersión y la función resultante al realizar el análisis de regresión lineal para un *read* y *write*, respectivamente.

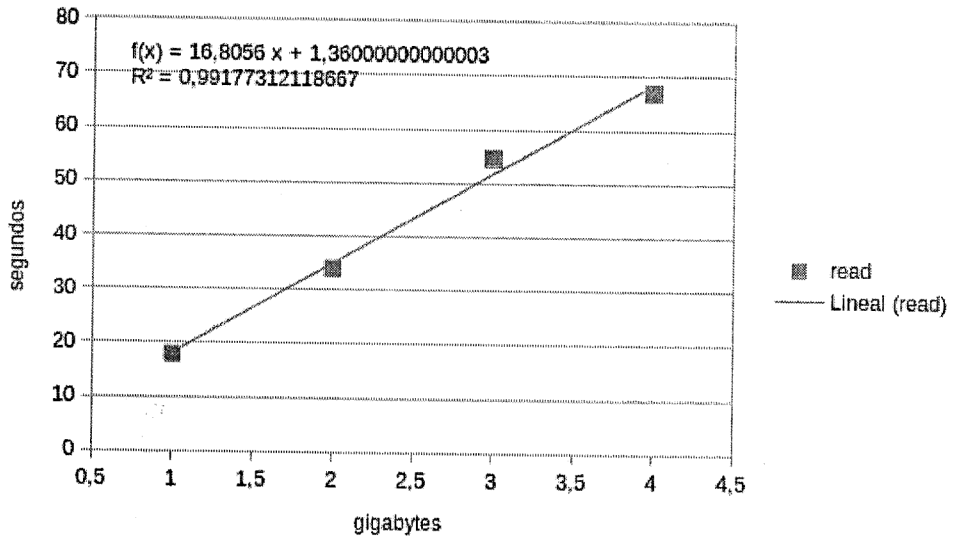


Figura 35. *Función lineal y coeficiente de determinación para un read* (imagen propia)

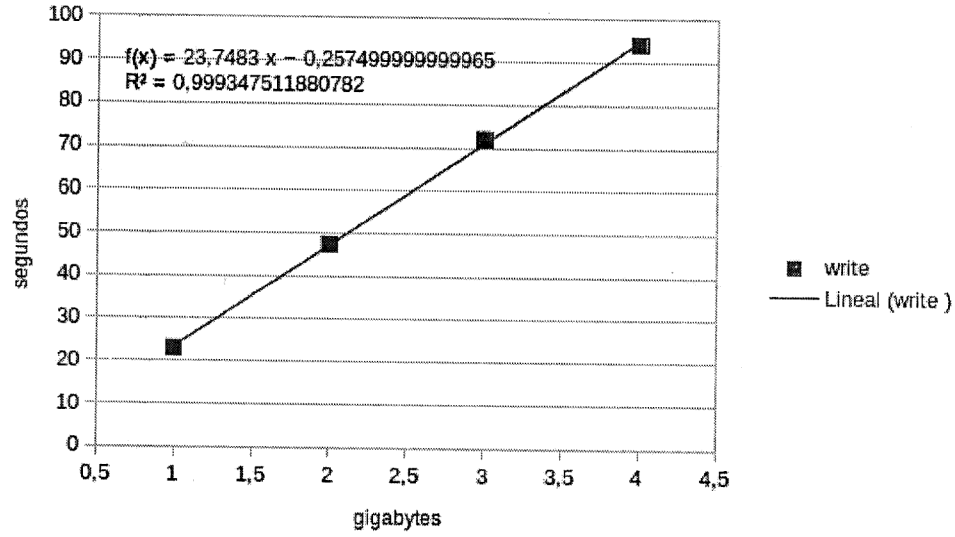


Figura 36. *Función lineal y coeficiente de determinación para un write* (imagen propia)

Firma Estudiante:	Firma Docente Supervisor:	Firma docente tutor TAPTA:	Firma tutor Organizacional:

El valor R^2 corresponde al coeficiente de determinación. Este valor es estadístico y representa la proporción de variación explicada por la regresión. Es una medida relativa del grado de asociación entre la variable independiente y la dependiente. Típicamente, toma valores entre cero y uno. Cuanto más cercano a uno sea el valor del coeficiente, mayor es el nivel de representación del modelo. En cambio, cuanto más cercano a cero sea el valor, menor es el nivel de representación del modelo.

En las figuras 37 y 38, se observa el diagrama de dispersión y la función resultante al realizar el análisis de regresión exponencial para un *read* y *write* respectivamente.

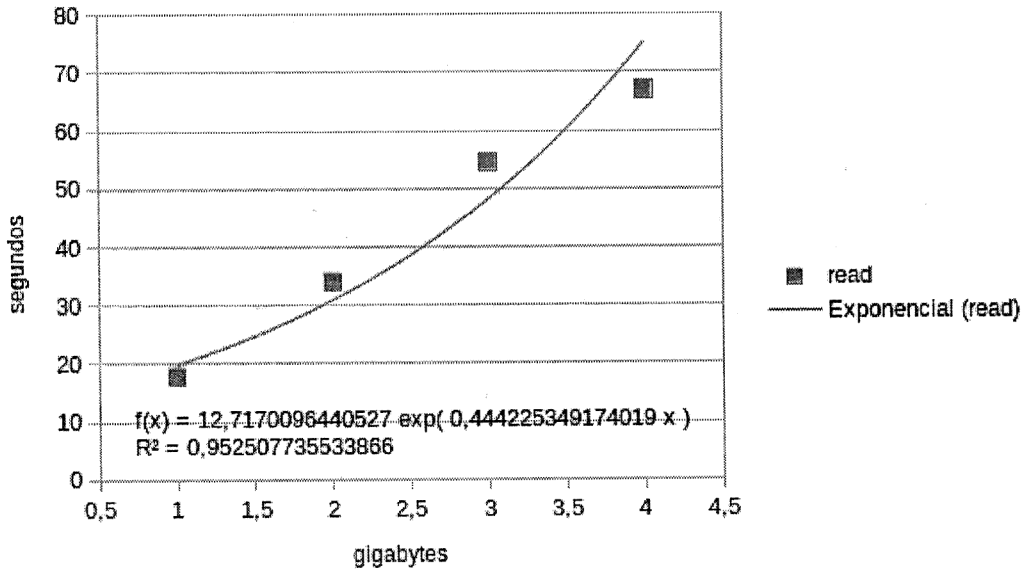


Figura 37. *Función exponencial y coeficiente de determinación para un read* (imagen propia)

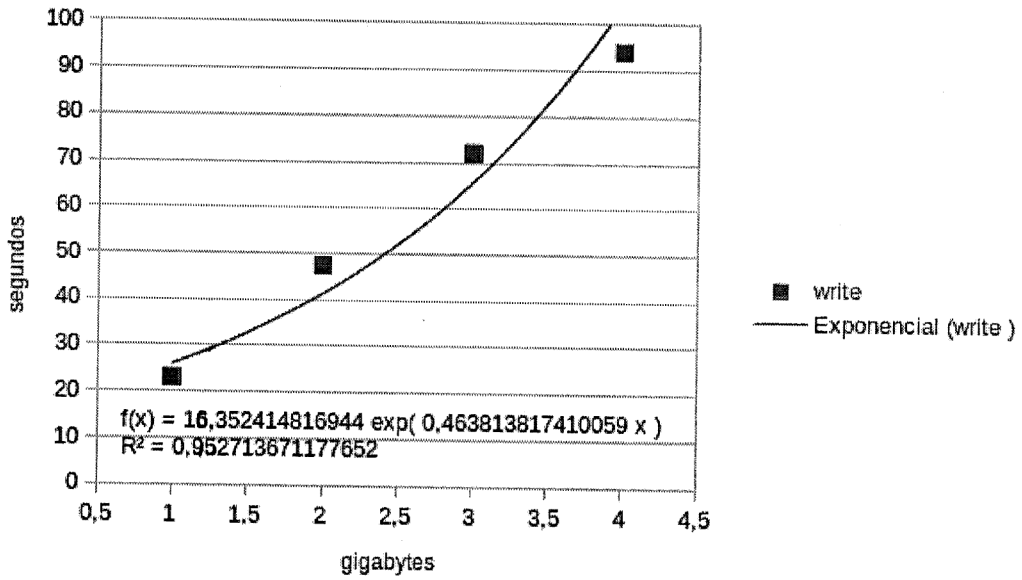


Figura 38. *Función exponencial y coeficiente de determinación para un write* (imagen propia)

Puede notarse que los coeficientes de determinación correspondientes a la regresión lineal son mayores a los coeficientes de la regresión exponencial.

En la tabla 3, se observa cómo quedan las expresiones finales, que se embeben en el simulador. La variable "x" corresponde al tamaño de archivo.

	Exponencial	Lineal
READ	$f(x) = 16.717 * e^{0,4442*x}$	$f(x) = 16,8056 * x + 1,63$
WRITE	$f(x) = 16.3524 * e^{0,4638*x}$	$f(x) = 23,7843 * x - 0,2575$

Tabla 3. *Funciones por operación y tipo de aproximación* (tabla propia)

A continuación, se muestran el método "linealRegresion", que se encuentra

definido en la clase "Ior".

```
public double linealRegresion(double fileSize) {
double time = 0.00;
switch (this.getOperacion_actual()) {
case "READ":
time = (16.8056 * fileSize) + 1.63;
break;
case "WRITE":
time = (23.7843 * fileSize) - 0.2575;
break;
default:
Log.printLine("Error lineal regresion");
break;}
return time;}
```

Figura 39. *Método "linealRegresion" (imagen propia)*

Del mismo modo, se define en el siguiente recuadro el método llamado "exponentialRegresion":

```
public double exponencialRegresion(int fileSize) {
double time = 0.00;
switch (this.getOperacion_actual()) {
case "READ":
```

```
time = (12.717 * Math.exp(0.4442 * fileSize));
break;
case "WRITE":
time = (16.3524 * Math.exp(0.4638 * fileSize));
break;
default:
Log.printLine("Error exponential regresion");
break;}
return time;}
```

Figura 40. *Metodo "exponentialRegresion"* (imagen propia)

Los dos métodos reciben como parámetro de entrada el tamaño de archivo y, como salida, devuelven el tiempo correspondiente.

Para el caso particular del método "exponentialRegresion", se utilizó la clase "Math" de JAVA, para poder trabajar con el número de Euler (*e*).

Si se ejecuta el simulador tres veces seguidas, por ejemplo, para un tamaño de archivo determinado, siempre se obtiene el mismo tiempo. No hay aleatoriedad ni diferencia entre ejecuciones, algo que no sucede en sistema real o virtual. Para que no suceda esto, es necesario contar con un rango de valores aleatorios que modifiquen de manera aceptable los tiempos obtenidos.

La estrategia que se utiliza para obtener valor y, a partir de este, un rango de valores consiste en obtener el desvío estándar de los tiempos promedios de cada tamaño de archivo por operación, sin importar la cantidad de nodos.

El desvío estándar es un valor numérico que indica el grado o nivel de dispersión

de un conjunto de datos. Mientras mayor es el valor, mayor es la dispersión del conjunto de datos.

Una vez que se tienen los desvíos, se calcula un promedio de desvíos por operación. En la tabla 4, se observa el resultado de la realización de este proceso:

	Desvío estándar	
	write	***read***
1G	2,185178355	2,179894238
2G	2,334495901	2,206358488
3G	3,076129603	4,768654714
4G	1,821196493	1,135858974
Promedio de los desvíos	2,354250088	2,572566603

Tabla 4. *Promedio de los desvíos (segundos)* (tabla propia)

A partir de estos resultados, se obtienen los rangos para las dos operaciones de entrada/salida:

- Para un *write*, el rango queda determinado desde los -2,354250088 segundos hasta los 2,354250088 segundos.
- Para un *read*, el rango se establece entre -2,572566603 segundos hasta 2,572566603 segundos.

Si se obtiene un valor aleatorio dentro del rango y si se lo suma al tiempo que se obtiene de utilizar la aproximación lineal o exponencial, se logra aleatoriedad en la

salida del simulador.

Al implementar esto, es necesario utilizar el método "random" de la clase "Math" de JAVA. Como el evento "RESULTS" de la clase "Ior" se encarga de calcular los tiempos y mostrar los resultados, a continuación, se muestra el método "output" que se invoca en este evento.

```
public void output(Double total_time) {
int fileSize = this.megaToGiga(this.getBlockSize() * this.getSegmentCount());
Log.printLine("Display results...");
Log.printLine("########## IOR TEST ##########");
Log.printLine("operation: " + this.getOperacion_actual());
Log.printLine("API: " + API);
Log.printLine("access: single shared file");
Log.printLine("ordering in a file: sequencial offsets");
Log.printLine("xfersize: " + this.getTransferSize() + "MB");
Log.printLine("blocksize: " + this.getBlockSize() + "MB");
Log.printLine("aggregate filesize (GB): " + fileSize );
Double random_time = total_time + (Math.random() * (2.572566603 + 2.572566603) -
2.572566603);
if ("WRITE".equals(this.getOperacion_actual())) {
random_time = total_time + (Math.random() * (2.354250088 + 2.354250088) -
2.354250088);
Log.printConcatLine("total time (" + getTipo_regresión() + "): " + random_time + "
seconds");
} else {
Log.printConcatLine("total time (" + getTipo_regresión() + "): " + random_time + "
seconds");}
Log.printLine("Max bandwith (MB/s): " + (this.getBlockSize() * this.getSegmentCount()) /
random_time);
Log.printLine("####### END IOR TEST ########");}
```

Figura 41. *Método "output" de la clase IOR (imagen propia)*

Firma Estudiante:	Firma Docente Supervisor:	Firma docente tutor TAPTA:	Firma tutor Organizacional:

Capítulo 10. Resultados

En la siguiente sección, se presentan los resultados obtenidos en el simulador. De cada escenario se muestran el tiempo en segundos y el ancho de banda en *megabytes* por segundo.

10.1 Tiempos para 5 nodos

En la tabla 5, se observan los valores obtenidos del simulador correspondientes a cinco nodos con regresión exponencial.

EXPONENCIAL	Tiempo (segundos)	
	write	***read***
1G	25,6056563455148	20,6553903353487
	26,1422526134942	19,6680603538157
	26,5176968988781	17,4586509214366
	26,9149078310508	19,6993142042857
	28,113678726528	18,103566586649
Promedio	26,6588384830932	19,1169964803071
2G	41,5225522421506	28,684248239395
	43,5324918430524	31,018962406116

	39,0477045797315	31,0619658052201
	39,777942058539	32,6483058000262
	41,8855271845308	32,305360103006
Promedio	41,1532435816009	31,1437684707527
3G	66,5609047138411	47,5081957596578
	68,0170906979608	49,2243190415473
	66,4471782628337	47,327586338076
	63,8554130959533	46,8017526675324
	64,5095439302419	46,6557838083501
Promedio	65,8780261401662	47,5035275230327
4G	104,791623334706	74,3283297753558
	105,252656158062	73,5584375481182
	104,053718419439	73,5041273328774
	104,756001702743	75,432884056659

Firma Estudiante:	Firma Docente Supervisor:	Firma docente tutor TAPTA:	Firma tutor Organizacional:

	105,664177889188	74,8147060785629
Promedio	104,903635500828	74,3276969583147

Tabla 5. *Operaciones write y read, con cinco nodos y regresión exponencial* (tabla propia)

En la figura 42, se presentan los datos de la tabla 5 de forma gráfica.

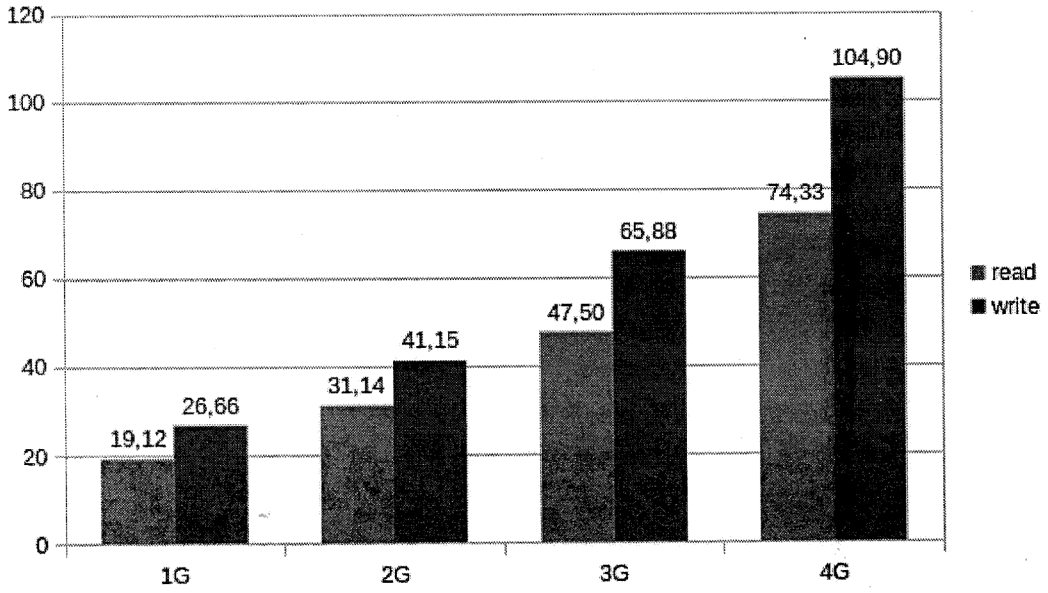


Figura 42. *Operaciones write y read, con cinco nodos y regresión exponencial* (imagen propia)

En la tabla 6, se muestran los resultados obtenidos aplicando regresión lineal.

LINEAL	Tiempo (segundos)	
	write	***read***

1G	21,1826853342418	20,6729243994615
	23,3260690061736	20,9816186884293
	22,5886048171554	19,2936417741675
	24,5109265214339	20,0087973633165
	23,1197744988888	16,6801467719016
Promedio	22,9456120355787	19,5274257994553
2G	45,6453642086531	36,6655440219427
	45,9104930765408	35,0146698489124
	46,0960495026619	33,9881145203226
	47,2567762872859	32,6892348122716
	47,2498264429698	37,3661344136778
Promedio	46,4317019036223	35,1447395234254
3G	70,9014188677864	52,0598336501768

	70,1270791859813	51,2579739127325
	69,3769091933621	49,7566707732555
	73,3130801647943	51,800132873805
	72,3644753271289	50,0400656229362
Promedio	71,2165925478106	50,9829353665812
4G	96,0269046320084	70,3478272217828
	96,8969122962234	69,7909995207443
	92,5922485347302	66,6707713943828
	94,2038294854832	68,9622703918661
	94,0047749206029	70,252856006136
Promedio	94,7449339738096	69,2049449069824

Tabla 6. *Operaciones write y read, con cinco nodos y regresión lineal* (tabla propia)

En la figura 43, se presentan los datos de la tabla 6 de manera gráfica.

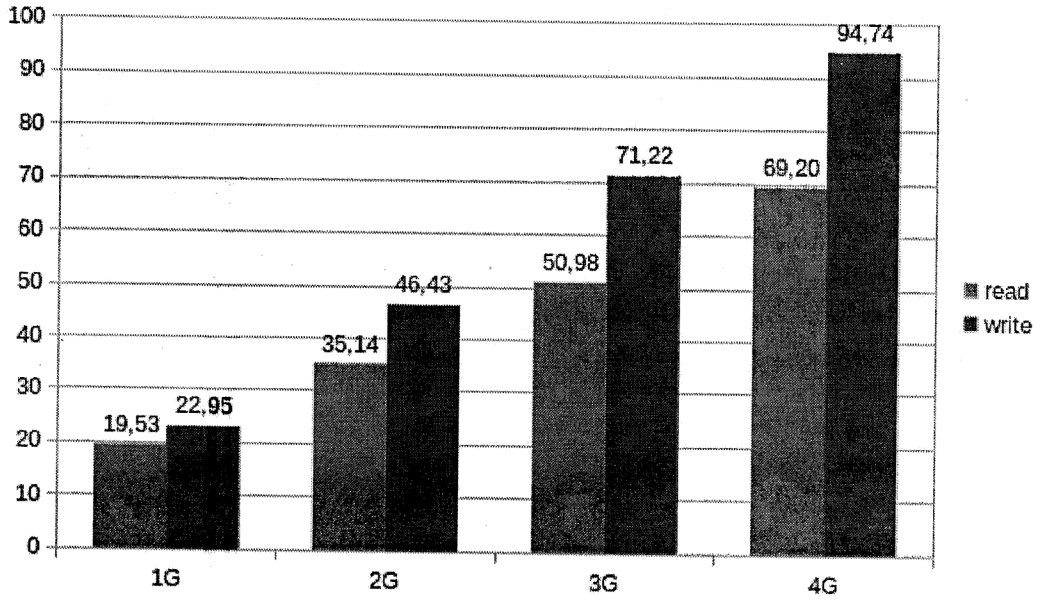


Figura 43. *Operaciones write y read, con cinco nodos y regresión lineal* (imagen propia)

10.2 Tiempos para 10 nodos

En la tabla 7, se observan los valores obtenidos del simulador, correspondientes a diez nodos, con regresión exponencial:

EXPONENCIAL	Tiempo (segundos)	
	write	***read***
1G	25,7106678697749	21,919068068077

Firma Estudiante:	Firma Docente Supervisor:	Firma docente tutor TAPTA:	Firma tutor Organizacional:

	25,2899535095326	20,1778323351485
	28,1737860378062	18,572800783326
	26,6768524207892	21,5053213820169
	24,4118812880873	18,0260891260753
Promedio	26,0526282251981	20,0402223389288
2G	43,4959957633568	30,3956841165914
	39,0537004465957	29,9910630903612
	40,0859520905485	28,6977207344583
	41,4681471667574	29,4917325223443
	39,8461289795742	33,0935050060967
Promedio	40,7899848893665	30,3339410939704
3G	64,0200370720169	48,7716126333731
	64,5414304783293	49,4604840720734

Firma Estudiante:	Firma Docente Supervisor:	Firma docente tutor TAPTA:	Firma tutor Organizacional:

	66,6981186971695	48,8210562527938
	67,7906839027196	45,7566496408812
	65,9982864885329	47,6694804526175
Promedio	65,8097113277536	48,0958566103478
4G	105,654769703563	74,4323568564446
	104,004146044807	73,7171972131128
	104,82109866093	72,6932711221246
	104,584177557708	76,7333291504508
	102,678383344033	75,3509401201655
Promedio	104,348515062208	74,5854188924597

Tabla 7. *Operaciones write y read, con diez nodos y regresión exponencial* (tabla propia)

En la figura 44, se presentan los datos de la tabla 7, de manera gráfica:

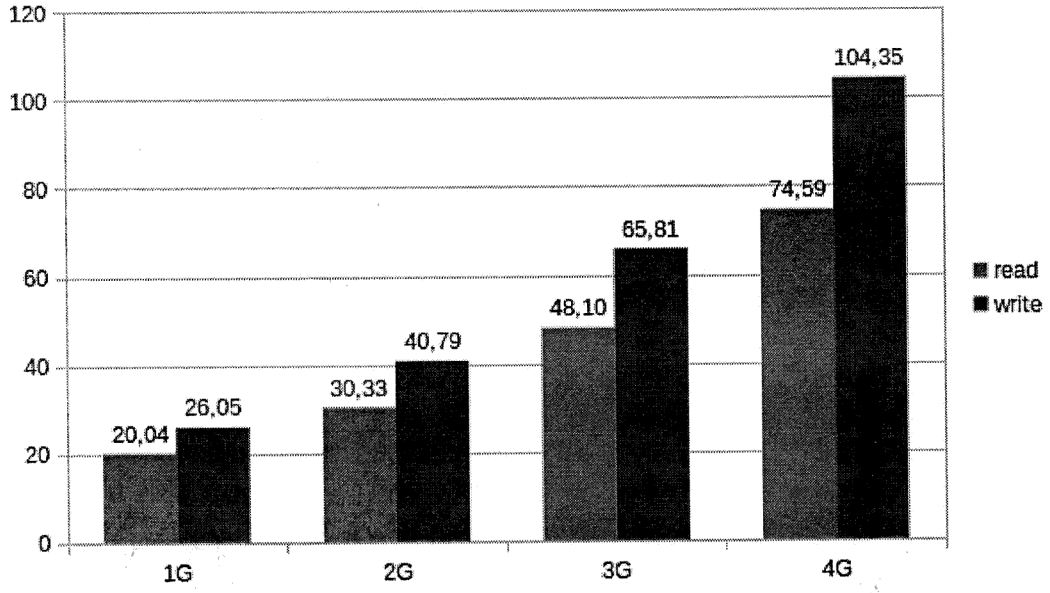


Figura 44. *Operaciones write y read, con diez nodos y regresión exponencial* (imagen propia)

En la tabla 8, se muestran los resultados obtenidos aplicando regresión lineal

LINEAL	Tiempo (segundos)	
	write	***read***
1G	23,3967764607104	20,4944942149776
	25,4364873294467	16,8179723019947
	23,5571762195053	18,7600621549889

Firma Estudiante:	Firma Docente Supervisor:	Firma docente tutor TAPTA:	Firma tutor Organizacional:

	22,0035639959472	16,5363546122947
	22,0028942993157	17,3943293556014
Promedio	23,2793796609851	18,0006425279715
2G	49,1070095916762	34,980490407094
	45,9246683321086	36,4906032663213
	48,4262327258949	33,3928516837557
	48,7511447281892	35,1155702577301
	49,6229637690178	36,6209468466393
Promedio	48,3664038293774	35,3200924923081
3G	72,6132491629249	53,3266948389201
	69,178316656029	53,1100298717599
	72,7783287147814	54,5615623523911
	72,0743631050705	52,7829356713174

Firma Estudiante:	Firma Docente Supervisor:	Firma docente tutor TAPTA:	Firma tutor Organizacional:

	72,4487729181047	50,2459506775774
Promedio	71,8186061113821	52,8054346823932
4G	94,8563645894825	68,6050828472325
	93,1385193022292	70,9593848605562
	96,7557200631915	67,4468151625029
	98,7050239123965	67,1447674623709
	94,298862684018	68,0843459609081
Promedio	94,5508981102635	68,4480792587141

Tabla 8. *Operaciones write y read, con cinco nodos y regresión lineal* (tabla propia)

En la figura 45, se presentan los datos de la tabla 8 de manera gráfica.

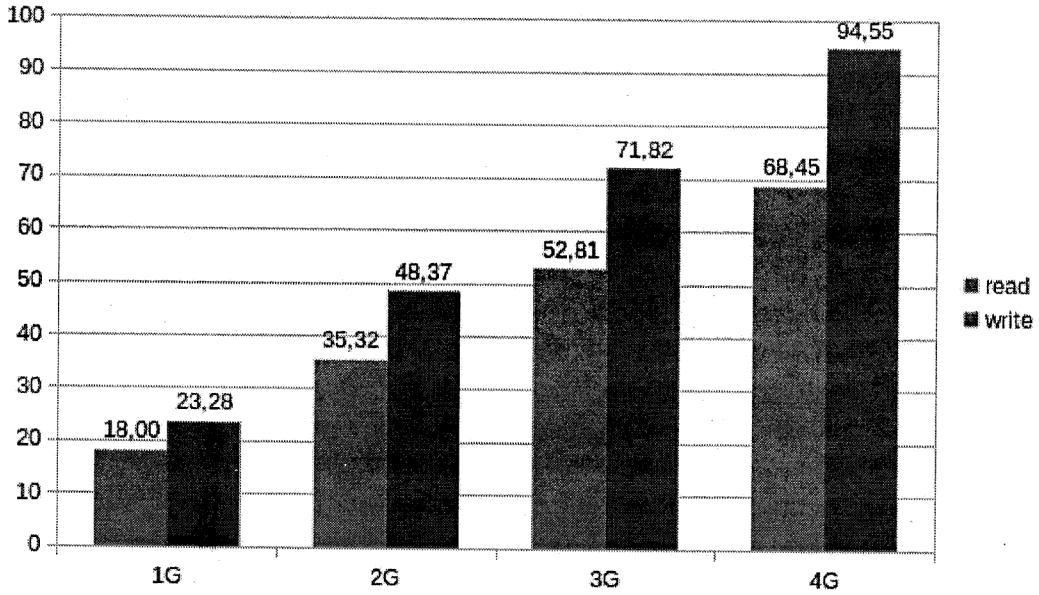


Figura 45. *Operaciones write y read, con diez nodos y regresión lineal* (imagen propia)

10.3 Anchos de banda para 5 nodos

En la tabla 9 se observan los valores correspondientes al ancho de banda por cada tamaño de archivo y tipo de operación, para la aproximación exponencial.

EXPONENCIAL	***Bandwidth* (MB/s)**	
Tamaño	***write***	***read***
1G	39,9911639124754	49,5754368895935
	39,1703046841277	52,0641070638845
	38,6157215652964	58,6528709811526

	38,0458297099813	51,9815050098151
	36,4235506125265	56,5634398668811
Promedio	38,4493140968815	53,7674719622654
2G	49,322594335158	71,3980712657225
	47,0453197897251	66,0241297947542
	52,4486656012824	65,9327234097923
	51,4858208850039	62,7291355497644
	48,8951706630631	63,3950525073836
Promedio	49,8395142548465	65,8958225054834
3G	46,1532188182711	64,6625271888062
	45,1651190675243	62,4081766861439
	46,2322115146653	64,9092894375751
	48,1086857175885	65,6385674661096

Firma Estudiante:	Firma Docente Supervisor:	Firma docente tutor TAPTA:	Firma tutor Organizacional:

	47,6208606174916	65,8439265026387
Promedio	46,6560191471082	64,6924974562547
4G	39,0870936975309	55,1068483898324
	38,9158825013296	55,6836188550172
	39,3642828167762	55,7247619776572
	39,1003850225485	54,2999257051264
	38,7643199599354	54,7485944233851
Promedio	39,0463927996241	55,1127498702037

Tabla 9. *Anchos de banda para 5 nodos. Aproximación exponencial* (tabla propia)

En las figuras 46 y 47, se presentan los anchos de banda o *bandwidth* de la tabla 9 de manera gráfica. Al mismo tiempo, se los compara con los tiempos obtenidos anteriormente:

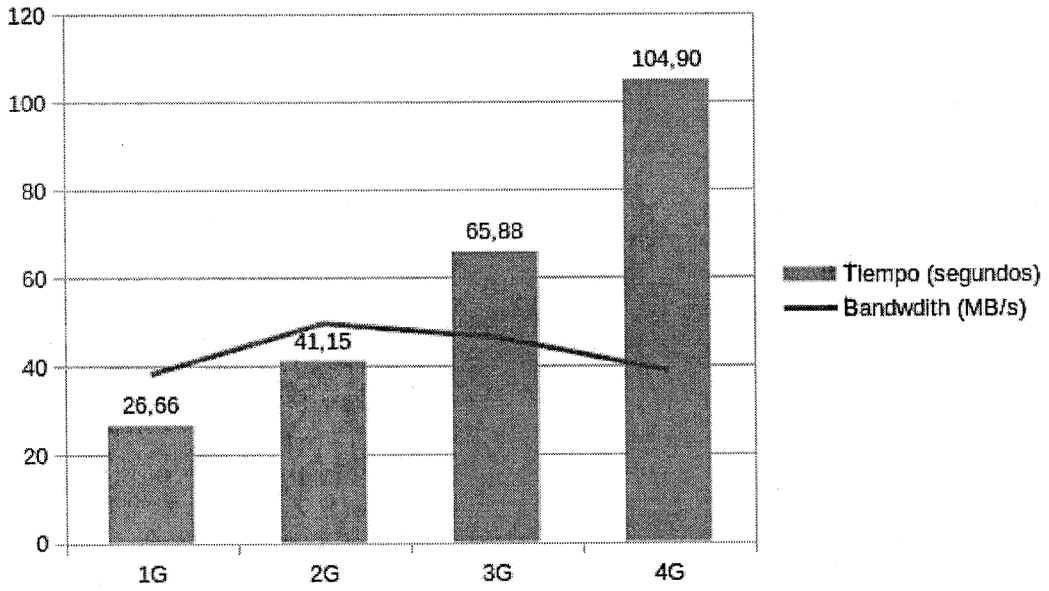


Figura 46. *Ancho de banda vs. tiempo correspondientes a un write. Cinco nodos, aproximación exponencial* (imagen propia)

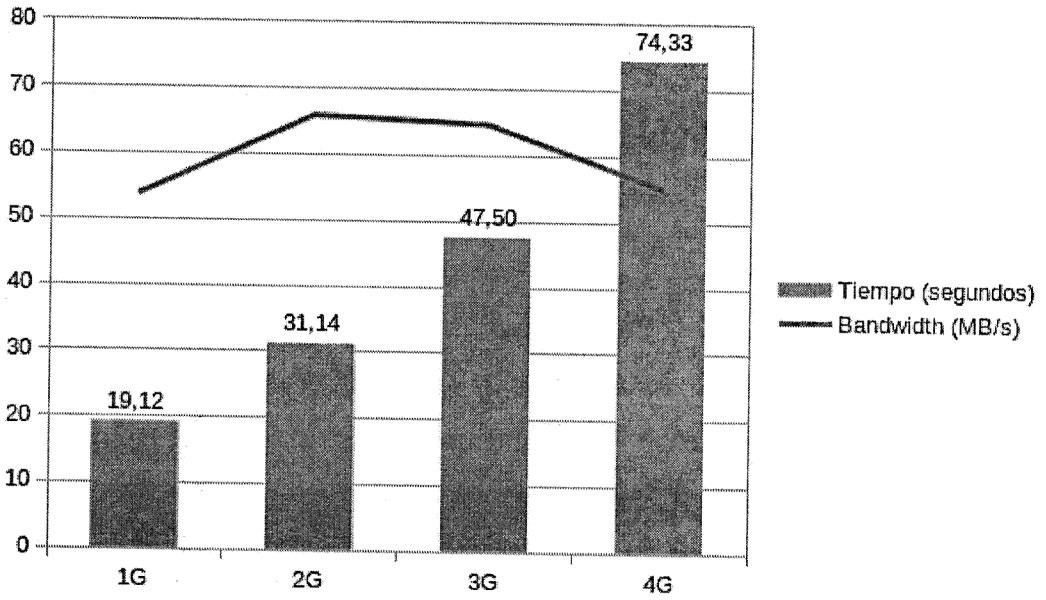


Figura 47. *Ancho de banda vs. tiempo correspondientes a un read. Cinco nodos, aproximación exponencial* (imagen propia)

En la tabla 10, se observan los valores correspondientes al ancho de banda, pero para la aproximación lineal:

LINEAL	***Bandwidth (MB/s)***	
Tamaño	***write***	***read***
1G	48,3413686150879 43,899381405799	49,5333887075345 48,8046234757238

Firma Estudiante:	Firma Docente Supervisor:	Firma docente tutor TAPTA:	Firma tutor Organizacional:

	45,3325917332575	53,0744797683063
	41,7772865136106	51,1774886519352
	44,2910894329535	61,3903470996414
Promedio	44,7283435401417	52,7960655406282
2G	44,8676450611332	55,8562556381098
	44,6085385444593	58,4897703972958
	44,4289699897544	60,2563581094042
	43,3377001331976	62,6505946609425
	43,3440745538382	54,8089876605039
Promedio	44,1173856564765	58,4123932932512
3G	43,327764790272	59,0090245128851
	43,8061877902097	59,9321386606136
	44,2798625035018	61,7404651930856

Firma Estudiante:	Firma Docente Supervisor:	Firma docente tutor TAPTA:	Firma tutor Organizacional:

	41,9024817003284	59,3048671802441
	42,4517691327519	61,3908067816747
Promedio	43,1536131834128	60,2754604657006
4G	42,6547124027019	58,2249681583868
	42,2717288191612	58,689516243173
	44,2369643768144	61,4362173158401
	43,4801857033975	59,3947962664975
	43,5722547440756	58,3036794922935
Promedio	43,2431692092301	59,2098354952382

Tabla 10. *Anchos de banda para 5 nodos. Aproximación lineal* (tabla propia)

En las figuras 48 y 49, se presentan los anchos de banda o *bandwidth* de la tabla 10 de manera gráfica. Una vez más, se los compara con los tiempos obtenidos anteriormente.

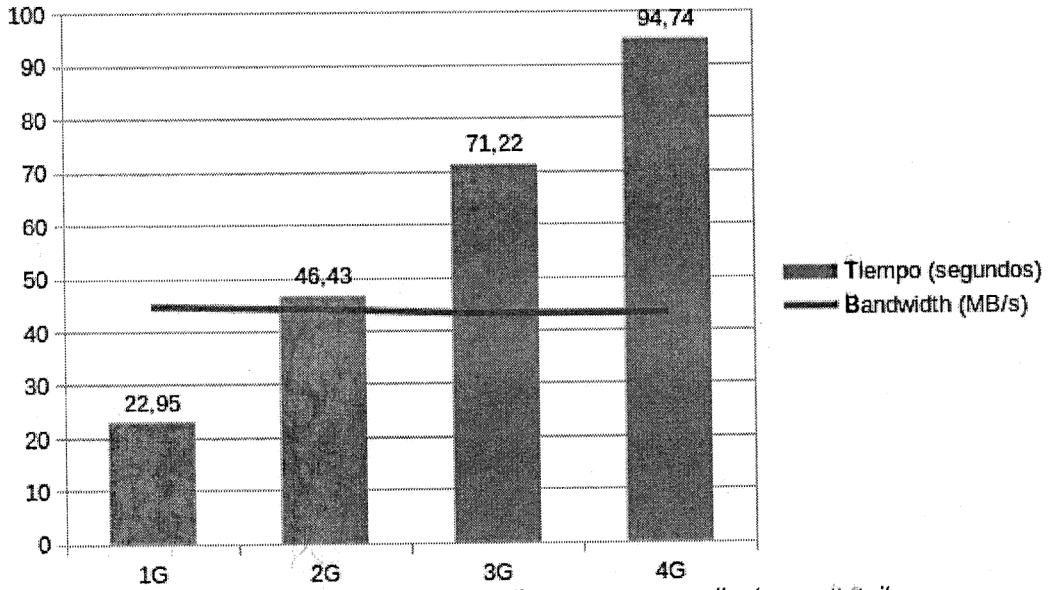


Figura 48. *Ancho de banda vs. tiempo correspondientes a un write. Cinco nodos, aproximación lineal* (imagen propia)

Firma Estudiante:	Firma Docente Supervisor:	Firma docente tutor TAPTA:	Firma tutor Organizacional:

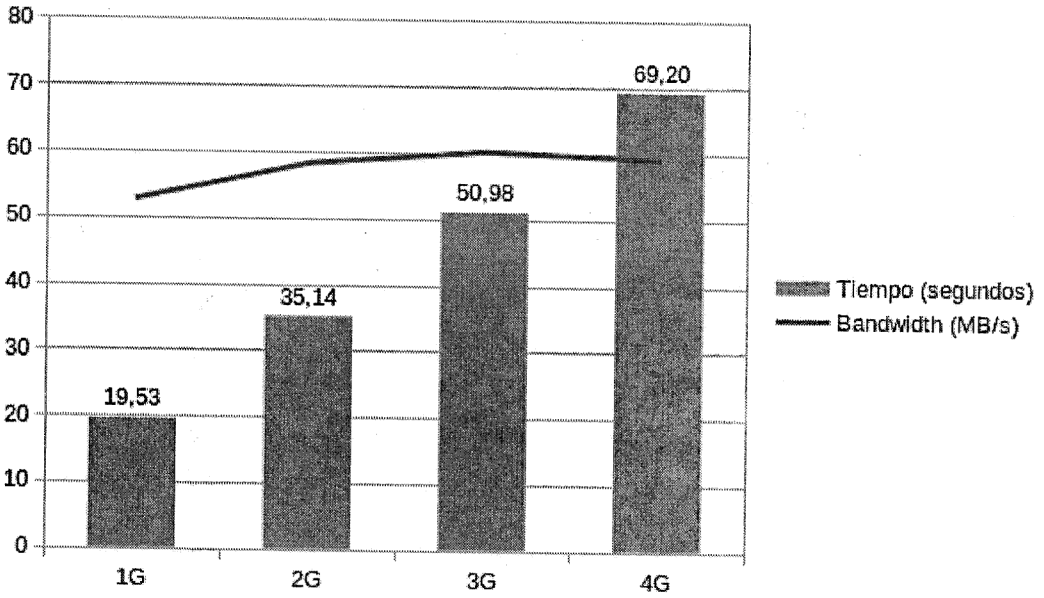


Figura 49. *Ancho de banda vs. tiempo correspondientes a un read. Cinco nodos, aproximación lineal* (imagen propia)

10.4 Anchos de banda para 10 nodos

En la tabla 11, se observan los valores correspondientes al ancho de banda por cada tamaño de archivo y tipo de operación, para la aproximación exponencial:

EXPONENCIAL	Bandwidth (MB/s)	
Tamaño	***write***	***read***
1G	39.827825756475264 40.49038680968786	46.7173146604419 50.7487614621644

	36,3458428564021	55,134387750463
	38,385338114402	47,6161216942466
	41,9467876283545	56,8065537032519
Promedio	38,8926561997195	51,4046278541136
2G	47,0847939921251	67,3779866952263
	52,440613221801	68,2870091610125
	51,0902172255721	71,364552570229
	49,3873042305049	69,4431905093517
	51,3977154731854	61,8852551164557
Promedio	50,2801288286377	67,671598810455
3G	47,9849768994084	62,9874600024587
	47,597333638762	62,1101887220413
	46,058270608019	62,9236693301614

Firma Estudiante:	Firma Docente Supervisor:	Firma docente tutor TAPTA:	Firma tutor Organizacional:

	45,3159611785058	67,1377826853679
	46,5466630036487	64,443748302512
Promedio	46,7006410656688	63,9205698085083
4G	38,7677717862827	55,0298307482031
	39,3830453473977	55,5636968692484
	39,0761025435301	56,346343159035
	39,1646240918221	53,3796727621316
	39,8915513334094	54,3589767223598
Promedio	39,2566190204884	54,9357040521956

Tabla 11. *Anchos de banda para 10 nodos. Aproximación exponencial* (tabla propia)

En las figuras 50 y 51, se presentan los anchos de banda o *bandwidth* de la tabla 11 de manera gráfica. Al mismo tiempo, se los compara con los tiempos obtenidos anteriormente.

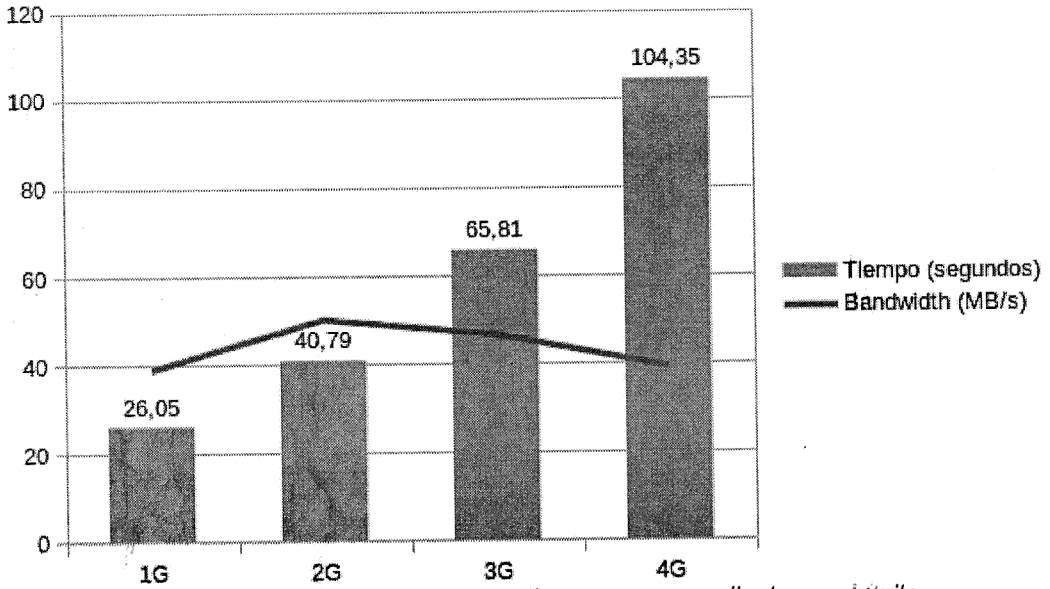


Figura 50. *Ancho de banda vs. tiempo correspondientes a un write. Diez nodos, aproximación exponencial* (imagen propia)

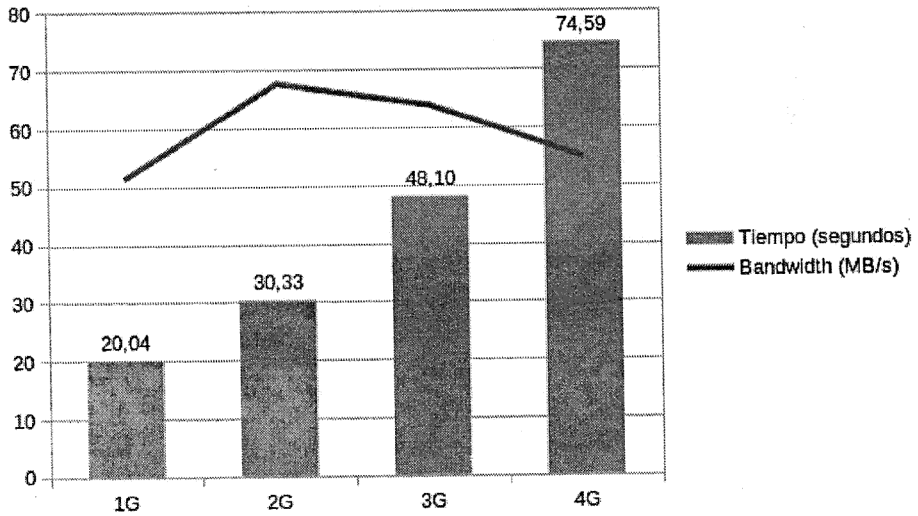


Figura 51. *Ancho de banda vs. tiempo correspondientes a un read. Diez nodos, aproximación exponencial* (imagen propia)

Firma Estudiante:	Firma Docente Supervisor:	Firma docente tutor TAPTA:	Firma tutor Organizacional:

En la tabla 12, se observan los valores correspondientes al ancho de banda por cada tamaño de archivo y tipo de operación, para la aproximación lineal.

LINEAL	Bandwidth (MB/s)	
Tamaño	**write**	**read**
1G	43,7667129794387	49,9646387590112
	40,2571308977306	60,8872450026896
	43,4687073891365	54,5840409024278
	46,5379154117309	61,9241679323122
	46,5393318747092	58,8697603147458
Promedio	44,1139597105492	57,2459705822373
2G	41,7048404500514	58,5469207597121
	44,5947695297371	56,124037880464
	42,2911278602285	61,3304913097995

	42,0092699652198	58,3217070082798
	41,2712148660228	55,9242776702794
Promedio	42,3742445342519	58,049486925707
3G	42,306328878181	57,6071704664869
	44,4069782049585	57,842181738886
	42,210367485068	56,3033730625086
	42,6226451078259	58,2006279288734
	42,4023744815189	61,1392551752614
Promedio	42,7897388315105	58,2185216744033
4G	43,1810771762822	59,7040311010313
	43,9775082391928	57,7231610455634
	42,3334144722905	60,7293315500711
	43,7116371031429	61,0025197018587

Firma Estudiante:	Firma Docente Supervisor:	Firma docente tutor TAPTA:	Firma tutor Organizacional:

	43,4363669233754	60,160671916446
Promedio	43,3280007828568	59,8639430629941

Tabla 12. *Anchos de banda para 10 nodos. Aproximación lineal* (tabla propia)

En las figuras 52 y 53, se presentan los anchos de banda o *bandwidth* de la tabla 12 de manera gráfica. Al mismo tiempo, se los compara con los tiempos obtenidos anteriormente.

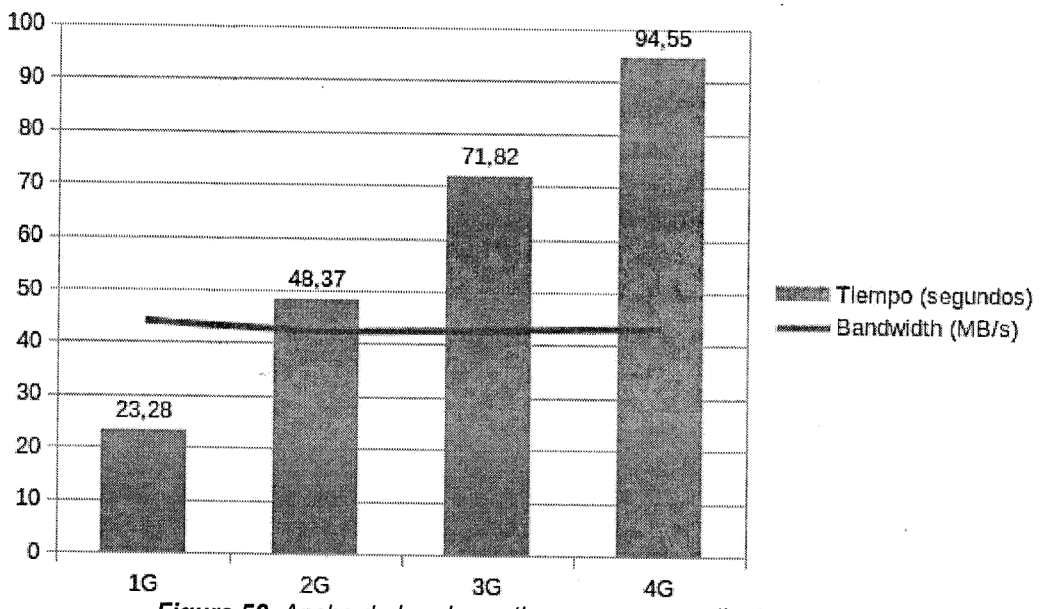


Figura 52. *Ancho de banda vs. tiempo correspondientes a un write. Diez nodos, aproximación lineal* (imagen propia)

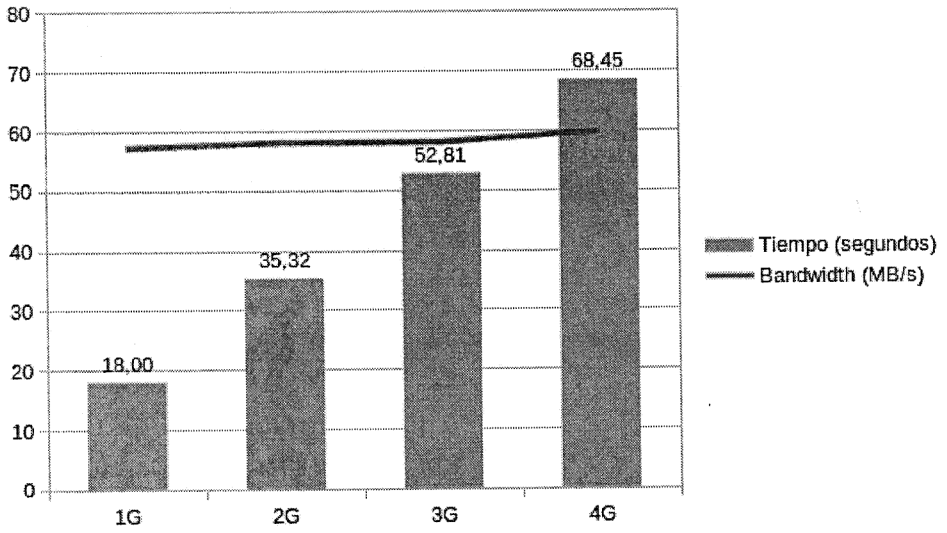


Figura 53. *Ancho de banda vs. tiempo correspondientes a un read. Diez nodos, aproximación lineal* (imagen propia)

Capítulo 11. Validación

Para validar la salida del simulador, se ejecutan pruebas con diez y cinco nodos, con una aproximación lineal, para tamaños de archivo de 1,5; 2,5 y 3,5 *gigabytes*. Luego, se comparan estos resultados con los que se obtienen en AWS para la misma cantidad de nodos y los mismos tamaños de archivos. A su vez, se incluyen los tiempos promedios.

11.1 Cinco nodos

En la tabla 13, se observa la comparación de tiempos del simulador con los de AWS, correspondientes a la operación de entrada/salida *write*.

	Tiempo de *write* (segundos)	
	Simulador	**AWS**
1,5G	36,6646316567951	33,9
	35,6100188485859	31,92
	36,8162948604073	31,65
	36,127116834608	35,2
	33,462867420562	33,65

Promedio	35,7361859241916	33,264
2,5G	60,2088284181229	56,21
	57,1322188744416	59,56
	57,2131821069383	60,05
	60,0663150221916	58,87
	59,8534007351963	59,11
Promedio	58,8947890313781	58,76
3,5G	82,3838661505437	88,53
	82,1590061675238	82,72
	83,5147444275587	83,36
	84,4435405264007	81,83
	82,0328334721007	79,09
Promedio	82,9067981488255	83,106

Firma Estudiante:	Firma Docente Supervisor:	Firma docente tutor TAPTA:	Firma tutor Organizacional:

Tabla 13. *Tiempos simulador vs. AWS. Operación write* (tabla propia)

En la figura 54, se observan los resultados de la tabla 13 en un gráfico de barras.

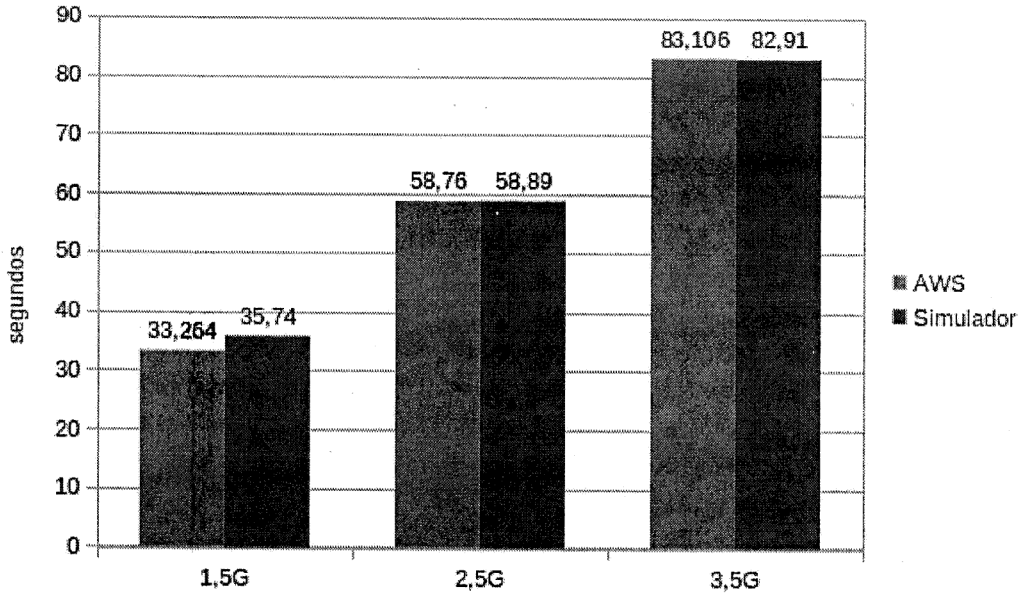


Figura 54. *Tiempos simulador vs. AWS. Write* (imagen propia)

En la tabla 14, se observa la comparación de tiempos del simulador con los de AWS, correspondientes a la operación de entrada/salida *read*.

	Tiempo de *read* (segundos)	
	Simulador	**AWS**
1,5G	25,9438898782144	24,23

	25,5504326558959	24,22
	26,4426051875609	24,24
	24,6267821037423	24,57
	25,6758001683219	24,27
Promedio	25,6479019987471	24,306
2,5G	42,4541162144949	40,97
	43,550000743728	41,09
	44,3618050097354	40,98
	44,03582675984	40,98
	42,4164192663089	41,04
Promedio	43,3636335988214	41,012
3,5G	61,4331943197858	57,75
	59,534576526122	58,32

Firma Estudiante:	Firma Docente Supervisor:	Firma docente tutor TAPTA:	Firma tutor Organizacional:

	62,2933287863054	57,75
	60,5414488452303	57,75
	62,0220198656486	57,76
Promedio	61,1649136686184	57,866

Tabla 14. *Tiempos simulador vs. AWS. Operación read* (tabla propia)

En la figura 55, se observan los resultados de la tabla 14 en un gráfico de barras.

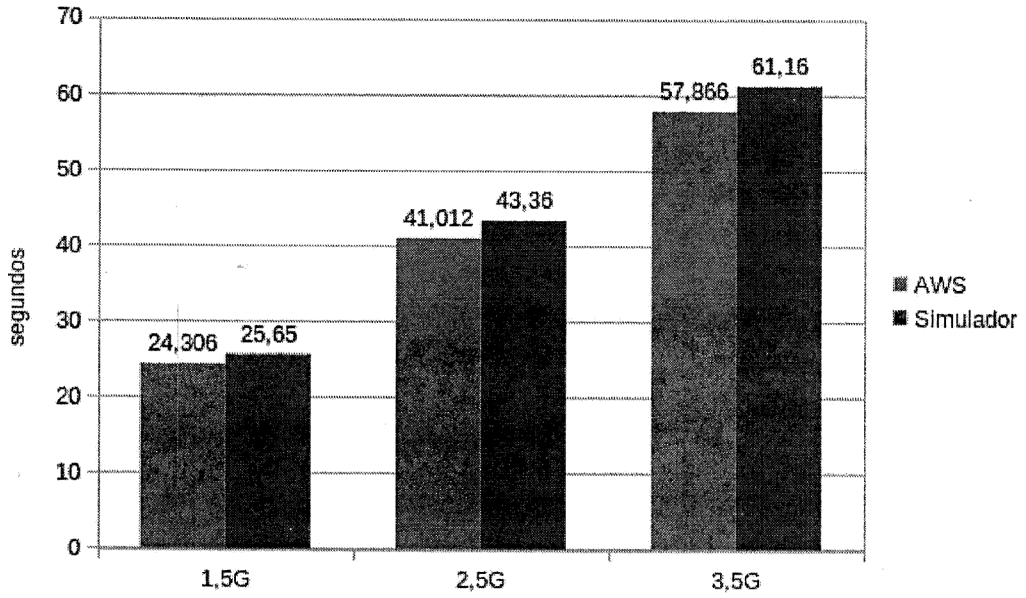


Figura 55. *Tiempos simulador vs. AWS. Read* (imagen propia)

11.2 Diez nodos

En la tabla 15, se observa la comparación de tiempos del simulador con los de

AWS, correspondientes a la operación de entrada/salida *write*.

	Tiempo de *write* (segundos)	
	Simulador	**AWS**
1,5G	34,933126540878	33,65
	34,4548091508186	31,06
	37,6893510624774	33,17
	33,8234628702863	30,76
	35,3981560910283	34,19
Promedio	35,2597811430977	32,566
2,5G	59,1370765867139	55,04
	57,5899749741724	57,08
	57,0113614901999	59,12
	60,7183247756508	60,45

	60,4715357863648	60,42
Promedio	58,9856547226204	58,422
3,5G	83,8558760151013	82,29
	81,5862384038785	80,6
	84,5323185691665	84,43
	83,2530037475124	82,75
	82,0991805660768	84,21
Promedio	*83,0653234603471*	*82,856*

Tabla 15. *Tiempos simulador vs. AWS. Operación write* (tabla propia)

En la figura 56, se observan los resultados de la tabla 15 en un gráfico de barras.

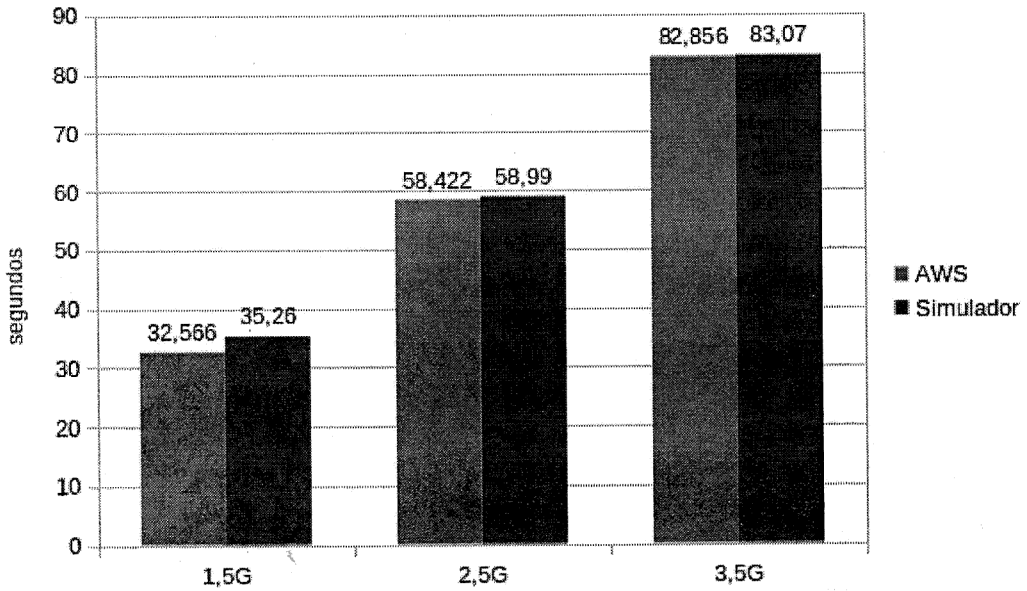


Figura 56. *Tiempos simulador vs. AWS. Write* (imagen propia)

En la tabla 16, se observa la comparación de tiempos del simulador con los de AWS, correspondientes a la operación de entrada/salida *read*.

	Tiempo de *read* (segundos)	
	Simulador	**AWS**
1,5G	28,6945060695365	24,24
	28,1052065153148	24,24
	28,4509087814129	24,66

Firma Estudiante:	Firma Docente Supervisor:	Firma docente tutor TAPTA:	Firma tutor Organizacional:

	27,9224712085637	24,19
	26,3128266575903	25,84
Promedio	27,8971838464837	24,634
2,5G	43,5013750080117	40,97
	44,9261367866297	40,87
	44,2673593735508	40,97
	44,7864290183289	40,97
	42,2166828394371	40,98
Promedio	43,9395966051917	40,952
3,5G	57,9496690938003	57,97
	62,9729152771159	57,98
	61,2202633502607	58,12
	60,9943621346183	58,12

Firma Estudiante:	Firma Docente Supervisor:	Firma docente tutor TAPTA:	Firma tutor Organizacional:

	62,2977770793861	58,84
Promedio	61,0869973870363	58,206

Tabla 16. *Tiempos simulador vs. AWS. Operación read* (tabla propia)

En la figura 57, se observan los resultados de la tabla 16 en un gráfico de barras.

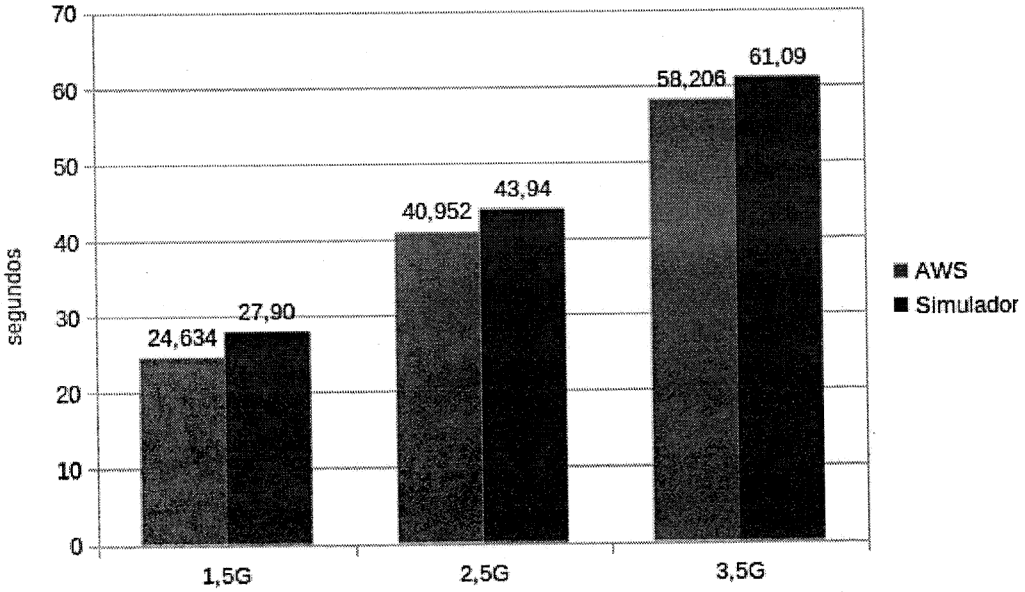


Figura 57. *Tiempos simulador vs. AWS. Read* (imagen propia)

Firma Estudiante:	Firma Docente Supervisor:	Firma docente tutor TAPTA:	Firma tutor Organizacional:

Capítulo 12. Conclusiones

Como se ha mencionado en la Introducción (capítulo 1), el principal objetivo del presente trabajo fue desarrollar una herramienta de simulación de sistemas de archivos distribuidos en entornos de *cloud computing*. Dicha herramienta debía permitir realizar el análisis temporal y espacial del sistema de archivos Network File System. Para poder cumplir con este objetivo, fue necesario estudiar y analizar los conceptos relacionados a: sistemas distribuidos, sistemas de archivos distribuidos, *cloud computing*, simulación, modelado, *benchmarking* y técnicas de análisis de datos. Como resultado final, esto permitió obtener:

- Un modelo del funcionamiento de NFS, realizado con diagramas de clase UML y máquinas de estado finito, con los principales protocolos, procedimientos básicos (*write*, *read*, *lock*, *unlock* y *mount*) y tecnologías que lo componen: *Mount Protocol*, *Remote Procedure Calls*, *External Data Representation* y *Network Lock Manager*.
- Un segundo modelo, matemático, que se obtuvo a partir de del despliegue de un escenario en la nube pública de *Amazon*. Con esto, se logró incorporar la variable temporal "tiempo de operación de entrada/salida".
- Un tercer modelo, correspondiente al funcionamiento básico del *benchmark* sintético IOR. De esta forma, se logró incluir los siguientes parámetros espaciales: tamaño de archivo, *block size* y *segment count*. A partir de estos parámetros y del tiempo de operación de entrada/salida, se obtuvo la salida principal del simulador (el ancho de banda).
- La metodología para poder incorporar nuevas funcionalidades al *framework* de simulación CloudSim.

A partir de los resultados obtenidos del análisis, fue posible implementar los

modelos en CloudSim. Como resultado, se ha obtenido un simulador que está orientado al análisis del performance, de las variables espaciales y temporales del sistema de archivos NFS. A su vez, permite a usuarios que poseen conocimientos básicos de programación orientada a objetos, entender cómo es el funcionamiento de NFS (desde la perspectiva de un protocolo), de forma dinámica y desde un punto de vista más específico.

Referido a los resultados obtenidos por el simulador (capítulo 10), se observa que:

- La salida que provee el simulador es muy similar a la que devuelve IOR.
- La aproximación lineal devuelve resultados que son muy similares a los resultados obtenidos del escenario en *Amazon*.
- La aproximación exponencial, a medida que el tamaño de archivo aumenta, presenta diferencias frente a los resultados obtenidos del escenario en *Amazon*.
- Al ejecutar IOR sin utilizar MPI (*Message Passing Interface*), con acceso secuencial y con POSIX, la cantidad de nodos no afecta el tiempo de operación.
- Los tiempos de *Amazon*, y, por ende, los del simulador, están sesgados o acotados por la capa gratuita de *Amazon* y, en consecuencia, por la elasticidad de este tipo de escenarios.

A futuro, se plantea la incorporación de nuevos modelos y funcionalidades al simulador. Concretamente, dos desarrollos posteriores planificados en este punto serían, por un lado, incorporar un modelo que permita ejecutar IOR con MPI, y, por otro lado, realizar otro modelo que permita calcular y obtener la cantidad de instrucciones de entrada/salida por segundo (IOPS).

El presente trabajo está enmarcado dentro del proyecto de investigación titulado

"Simulación y tecnología en Cómputo de Altas Prestaciones (High Performance Computing, HPC) para aplicaciones de interés social", cuyo director es el Mg. Diego Omar Encinas. Siguiendo este marco de trabajo, se realizaron las siguientes publicaciones:

- "Performance de cloud computing para HPC: despliegue y simulación. Autores: Galarza, Brian; Zaccardi, Gonzalo; Rossatto, Daniel; Bond, Román; Morales, Daniel Martín; Encinas, Diego. En: *XVIII Workshop de Investigadores en Ciencias de la Computación* (WICC 2016).
- "Performance de arquitecturas multiprocesador: técnicas de modelado y simulación en HPC y cloud computing". Autores: Encinas, Diego; Jara, Jimena; Rosatto, David; Bond, Román; Bermudez, Andrea; Morales, Daniel Martín. En: *XIX Workshop de Investigadores en Ciencias de la Computación* (WICC 2017).
- "Modelado y simulación de arquitecturas de cloud computing con cloudsim: comunicación entre entidades". Autores: Rosatto, Daniel; Bond, Román; Belizán, Maximiliano; Morales, Daniel Martín; Encinas, Diego. En: *XXIII Congreso Argentino de Ciencias de la Computación* (CACIC 2017).
- "Performance de arquitecturas multiprocesador: técnicas de modelado y simulación en HPC y cloud computing". Autores: Encinas, Diego; Jara, Jimena; Rosatto, David; Bond, Román; Belizán, Maximiliano; Morales, Daniel Martín. En: *XX Workshop de Investigadores en Ciencias de la Computación* (WICC 2018).
- "Performance de cloud computing para HPC en IaaS privados y públicos". Autores: Galarza, Brian; Zaccardi, Gonzalo; Bond, Román; Montes de Oca, Federico; Maxit, Eduardo; Osio, Jorge; Duarte, David; Morales, Martín;

Encinas, Diego. En: *XXI Workshop de Investigadores en Ciencias de la Computación* (WICC 2019).

- "Performance de arquitecturas multiprocesador: técnicas de modelado y simulación en HPC y cloud computing". Autores: Encinas, Diego; Jara, Jimena; Rosatto, Daniel; Bond, Román; Morales, Martin. En: *XXI Workshop de Investigadores en Ciencias de la Computación* (WICC 2019).

Finalmente, y como parte fundamental de la divulgación científica de nuestra investigación, dichas publicaciones constituyen, asimismo, el resultado tangible del proceso de trabajo realizado a lo largo de la presente Práctica Profesional Supervisada, orientada al campo de la indagación en el área.

Bibliografía

Amazon (2019). *Amazon Elastic Compute Cloud: User Guide for Linux Instances*. Disponible en: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-ug.pdf [Consulta: 12 de junio de 2019].

Booch, Grady; Rumbaugh, James; Jacobson, Ivar (1999). *The unified modeling language: User Guide*. Massachusetts: Addison Wesley.

Buyya, Rajkumar (2008). *Market-oriented cloud computing: Vision, hype, and reality for delivering it services as computing utilities*. Disponible en: http://www.buyya.com/papers/MarketGridUtilityComp2011_1007.pdf [Consulta: 11 de junio de 2019].

Buyya, Rajkumar (2010). "CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms", *Journal of Software: Practice and Experience*, 41, 23-50. Disponible en: https://www.buyya.com/papers/CloudSim2010.pdf [Consulta: 4 de junio de 2019].

Callaghan, Brent (1999). *NFS Illustrated*. Massachusetts: Addison Wesley, 2000.

Coulouris, George; Dollimore, Jean y Kindberg, Tim (2001). *Distributed Systems: Concepts and Design*. Massachusetts: Pearson, 2014.

Foster, Ian (2001). *The anatomy of the grid: enabling scalable virtual organizations*. Disponible en: https://arxiv.org/pdf/cs/0103025.pdf [Consulta: 11 de junio de 2019].

Gordon, Geoffrey (1962). "A general purpose systems simulator". *IBM Systems Journal*. New Jersey: IBM Corporation, 1, pp. 18-31.

Hein, James (2010). *Discrete Structures, Logic, and Computability*. Massachusetts: Jones & Bartlett Publishers.

Neuman, Clifford (1994). "Scale in Distributed Systems". En: *Readings in Distributed Computing Systems*. California: IEEE Computer Society Press, pp. 463-489.

Montanaro, Franco (2014). *Concurrencia en RPC: Análisis de Factibilidad y Relación con Protocolos de Comunicaciones*. Mimeo de tesina de Licenciatura, UNLP. Disponible en: http://sedici.unlp.edu.ar/bitstream/handle/10915/63689/Documento_completo.%20Concurrencia%20en%20RPC.pdf-PDFA.pdf [Consulta: 29 de mayo de 2019].

Firma Estudiante:	Firma Docente Supervisor:	Firma docente tutor TAPTA:	Firma tutor Organizacional:

Ríos, Sixto (1995). *Modelización*. Madrid: Alianza.

Sandberg, Rusell (1985). "Design and Implementation of the Sun Network Filesystem". *Proceedings of the Summer 1985. USENIX Conference & Exhibition*. Oregon: USENIX Associtation Office, pp. 9-20. Disponible en: https://www.cs.ucf.edu/~eurip/papers/sandbergnfs.pdf [Consulta: 31 de mayo de 2019].

Shan, H. y Shalf, J. (2007). "Using IOR to analyze the I/O Performance for HPC Platforms". *Proceedings of the Cray User Group Meeting*. Washington DC: Cray User Group, pp. 1-15. Disponible en: https://www.osti.gov/servlets/purl/923356 [Consulta: 4 de junio de 2019].

Shannon, Robert (1998). "Introduction to the art and science of simulation". *Winter Simulation Conference*, 1, pp. 7-14. Disponible en: https://web1.cs.wright.edu/~fciarall/ISE195/Readings/ShannonSimulationART.pdf [Consulta: 3 de junio de 2019].

Srinivasan, R. (1995). XDR: *External Data Representations Standard*. RFC 1832. Disponible en: http://tools.ietf.org/html/rfc1832 [Consulta: 25 de mayo de 2019].

Tanenbaum, Andrew (1996). *Sistemas Operativos Distribuidos: Principios y paradigmas*. Massachusetts: Pearson-Prentice-Hall, 2012.

Tanenbaum, Andrew (1992). *Sistemas Operativos Modernos*. Massachusetts: Pearson-Prentice Hall, 2009.

The National Institute of Standards and Technology (NIST) (2011). *The NIST Definition of Cloud Computing*. Disponible en: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf [Consulta: 3 de junio de 2019].

Van Steen, Martin y Tanenbaum, Andrew (2017). *Distributed Systems*. California: CreateSpace Independent Publishing Platform.

Vaquero, Luis (2008). "A Break in the Clouds: Towards a Cloud Definition". *Computer Communication Review. Barcelona:* ACM SIGCOMM, 39, pp. 50-55. Disponible en: http://ccr.sigcomm.org/online/files/p50-v39n1l-vaqueroA.pdf [Consulta: 3 de junio de 2019].

Wampler, Bruce (2002). *The Essence of Object Oriented Programming with Java and UML*. Massachusetts: Addison-Wesley Professional.

Zhang, Qi; Cheng, Lu; y Boutaba, Raouf (2010). "Cloud Computing: State of the Art and Research Challenges". *Journal of Internet Services and Applications*. Londres: Springer London, 1, pp. 7-18. Disponible en: https://u.cs.biu.ac.il/~ariel/download/ds590/resources/cloud/cloud_sota.pdf

Firma Estudiante:	Firma Docente Supervisor:	Firma docente tutor TAPTA:	Firma tutor Organizacional:

[Consulta: 3 de junio de 2019].

Firma Estudiante:	Firma Docente Supervisor:	Firma docente tutor TAPTA:	Firma tutor Organizacional:

Anexo A

Explicar qué expone el anexo o poner el epígrafe debajo de la tabla, que indique el contenido. Algo como lo que está en el inicio del anexo D!!!
En este apartado, se incluye una tabla que lista los procedimientos NFS (correspondientes a la versión tres) junto a una breve descripción de cada uno.

Número V3	Nombre	Descripción
0	null	No hace nada. Para testeo.
1	getattr	Devuelve los atributos de un archivo remoto.
2	setattr	Cambia los atributos de un archivo.
3	lookup	Devuelve el *filehandle* de un archivo para que el cliente lo utilice.
5	readlink	Lee el nombre de un archivo mediante el enlace simbólico.
6	read	Lee datos de un archivo.
7	write	Escribe datos a un archivo.

8	create	Crea un archivo en servidor.
12	remove	Eliminar un archivo en servidor
14	rename	Cambia el nombre de un archivo.
15	link	Crea un enlace fuerte a un archivo.
10	symlink	Crea un enlace simbólico a un archivo.
9	mkdir	Crea un directorio en servidor.
13	rmdir	Eliminar un directorio.
16	readdir	Lee el contenido de un servidor.
4	access	Determina los permisos de acceso que tiene un usuario sobre un archivo.
11	mknod	Crea un archivo especial (*pipe or device file*).
17	readdirplus	Devuelve información adicional de un directorio
18	fstat	Devuelve información sobre el estado del sistema de

		archivos. Por ejemplo, la cantidad de espacio disponible.
19	fsinfo	Devuelve información estática sobre el sistema de archivos. Por ejemplo, cómo es utilizado.
20	pathconf	Devuelve información adicional sobre ín archivo o directorio a nivel POSIX.
21	commit	Realiza un flush sobre información que está en caché en el servidor.

Anexo B

B.1 UML (*Unified Modeling Language*)

El lenguaje unificado de modelado es un lenguaje de modelado visual que se usa para especificar, construir y documentar componentes de un proyecto o sistema de *software* (Booch, 1999).

Las ventajas de utilizar este lenguaje de modelado son las siguientes:

- Captura decisiones y conocimientos sobre los sistemas que se deben construir.
- Está pensado para ser utilizado en todas las metodologías y ciclos de desarrollo de software.
- Brinda un estándar en técnicas de modelado.
- Pretende dar apoyo, principalmente, a los desarrollos orientados a objetos.
- Es posible representar los comportamientos estáticos y dinámicos del sistema.

Cabe destacar que UML no es un lenguaje de modelado de sistemas continuos (como los modelos físicos), sino que pretende ser un lenguaje universal, de propósito general, para sistemas discretos.

Para el modelado de este proyecto, se utilizaron diagramas de clases. Existen otros tipos de diagramas provistos por UML, por ejemplo, diagramas de secuencia. Un diagrama de clases describe los tipos de objetos en el sistema con sus relaciones. Además, muestra las propiedades y operaciones de la clase y las restricciones de cada objeto.

Una clase es una descripción de conjunto de objetos que comparten los mismos atributos, operaciones, métodos, relaciones y semántica. Son gráficamente

representadas por cajas con compartimentos para el nombre de la clase, atributos, métodos y operaciones. Al respecto, puede verse la figura B1.

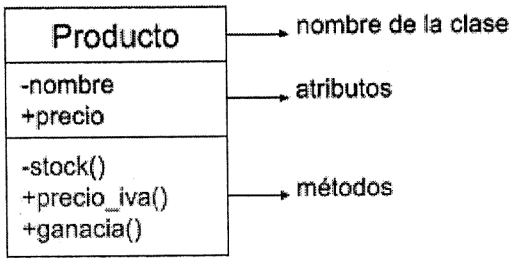


Figura B1. *Estructura de una clase* (imagen propia)

Los atributos, formalmente, tienen que tener la siguiente estructura al momento de declararlos:

- visibilidad: se especifica mediante tres símbolos distintos. El símbolo "+" indica que el atributo es público. En cambio, el símbolo "-" indica que es privado. Por otro lado, el símbolo "#" indica que es protegido.
- nombre: debe tener un nombre obligatorio y lo suficientemente descriptivo.
- tipo de dato: puede ser un entero, texto, carácter, doble, booleano, etc.
- multiplicidad: una clase puede tener uno o varios atributos del mismo tipo. Por ejemplo, la clase persona tiene uno o varios atributos teléfono.
- dato por *default* (opcional): se trata de un valor por defecto. Por ejemplo, el tipo de dato booleano con nombre "existe" es de este tenor. El valor por default puede ser "falso."

En la figura B2, se observan los tipos de relaciones que se utilizan para caracterizar las interacciones entre clases. Se focaliza la atención en la composición, la agregación y las relaciones de herencia.

Un objeto agregado incluye (tiene) otros objetos. Cada uno de estos objetos es considerado como parte del objeto agregado. Al respecto, puede verse la figura B3.a. Si el objeto agregado es destruido, el resto de los objetos siguen su ciclo de ejecución normal. En la figura, se observa claramente que un cliente puede o no realizar una compra.

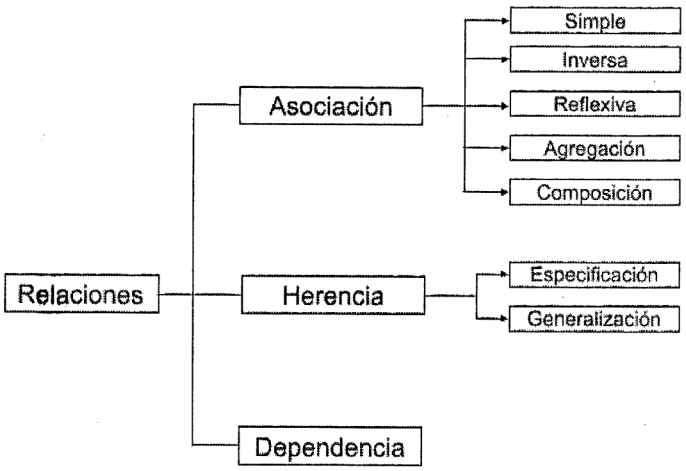


Figura B2. *Tipos de relaciones entre clases* (imagen propia)

Por otro lado, la composición es una forma de agregación (fuerte), donde el objeto "todo" no puede existir, o carece de sentido, sin los otros objetos que forman parte de la relación. Esto se grafica en la figura B3.b. Si el objeto "todo" es destruido, el resto de los objetos de la composición son destruidos (Wampler, 2002). En la figura B3.b, se observa que carece de sentido la existencia de un auto sin ruedas.

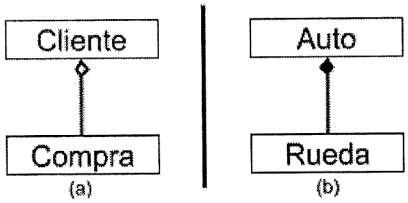


Figura B3. *En **"a"** agregación. En **"b"** composición* (imagen propia)

B.2 Máquinas de estado finito

Un autómata finito es un modelo que captura las características de una computadora o sistema. Desde el punto de vista del modelado, permite describir el comportamiento de un sistema de acuerdo a entradas y de cómo van cambian en el tiempo a través de eventos discretos.

Informalmente, sobre un alfabeto "A", un autómata finito puede ser pensado como un grafo dirigido, con la salvedad de que cada nodo tiene un conector con información de cada elemento o letra del alfabeto "A". (Hein, 2010). Esto se grafica en la figura B4. Los nodos pasarían a llamarse "estados". Uno de esos nodos es el "estado inicial" y otro, "estado final" (aunque, *a priori*, no es necesario).

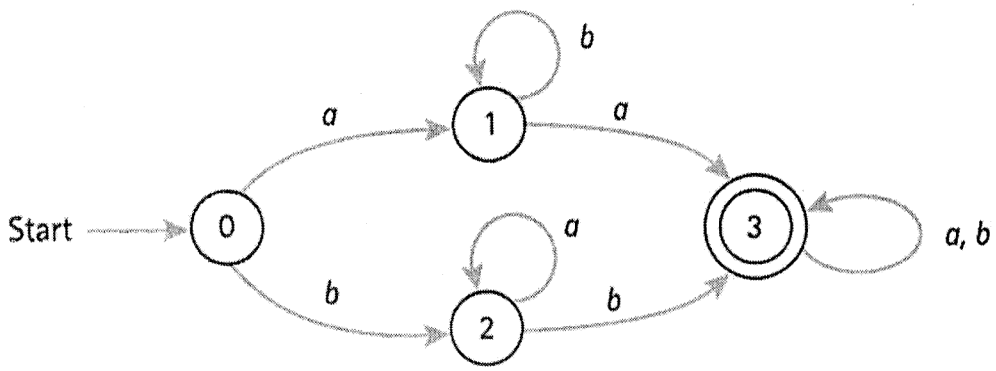


Figura B4. *Automata finito* (Hein, 2010: 701)

Para el modelado, se utilizaron autómatas finitos traductores, y, específicamente, los traductores de Moore. Los autómatas finitos traductores toman una entrada y la traducen en una salida. Particularmente, en los de Moore, las salidas están asociadas a los estados. Puede verse esto en la figura B5:

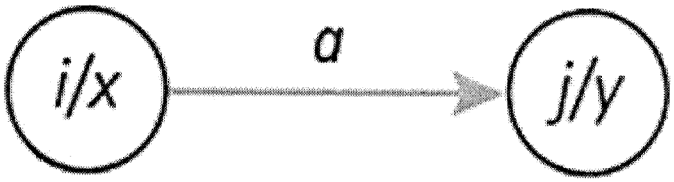


Figura B5. *Autómata traductor de Moore* (Hein, 2010: 714)

Formalmente, un autómata traductor de Moore queda definido como una séxtupla M = (S, Σ, Γ, δ, S0, f0), donde:

- S es un conjunto finito de estados, S≠∅.
- Σ es el alfabeto de entrada.
- Γ es el alfabeto de salida.
- δ: S x Σ → S es la función de transición
- S0 es el estado inicial, S0 ∈ S.
- f0: S → Γ* es la función de salida.

Anexo C

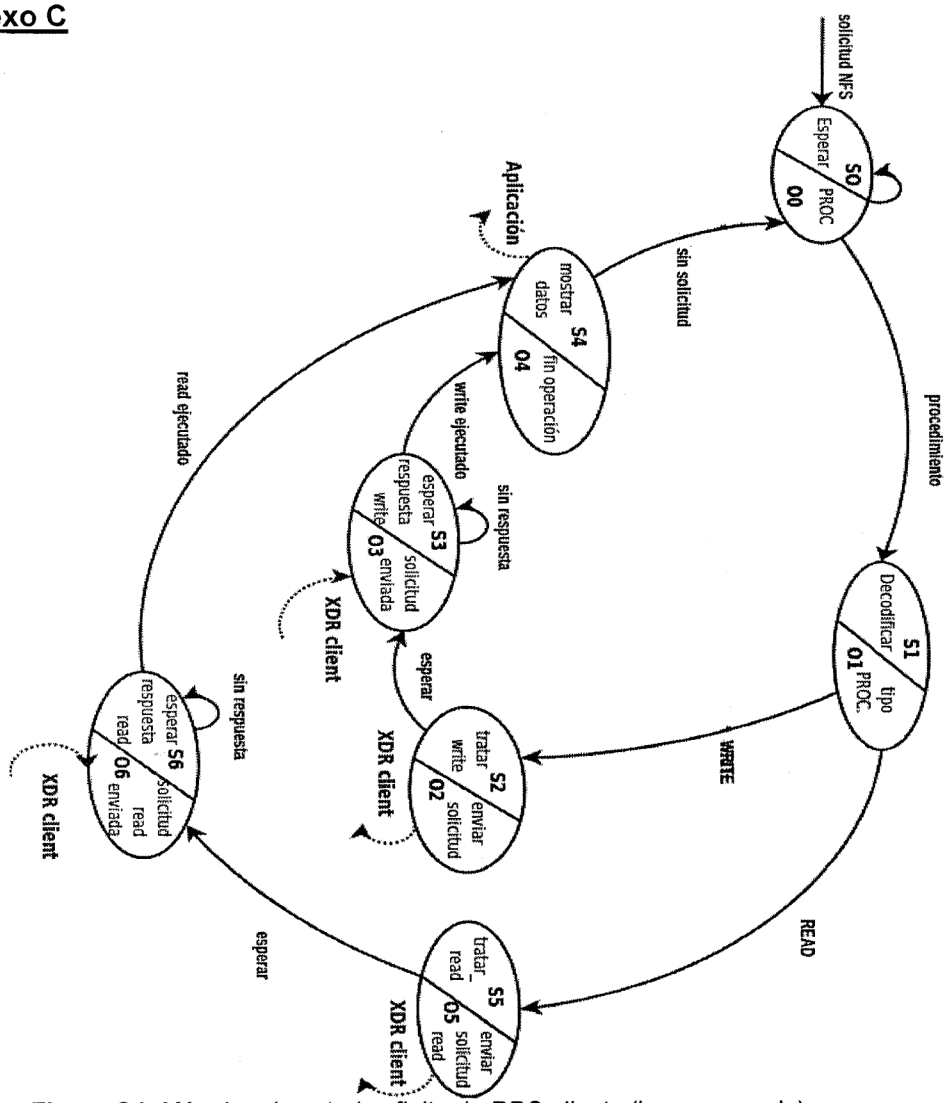


Figura C1. *Máquina de estados finito de RPC cliente* (imagen propia)

Firma Estudiante:	Firma Docente Supervisor:	Firma docente tutor TAPTA:	Firma tutor Organizacional:

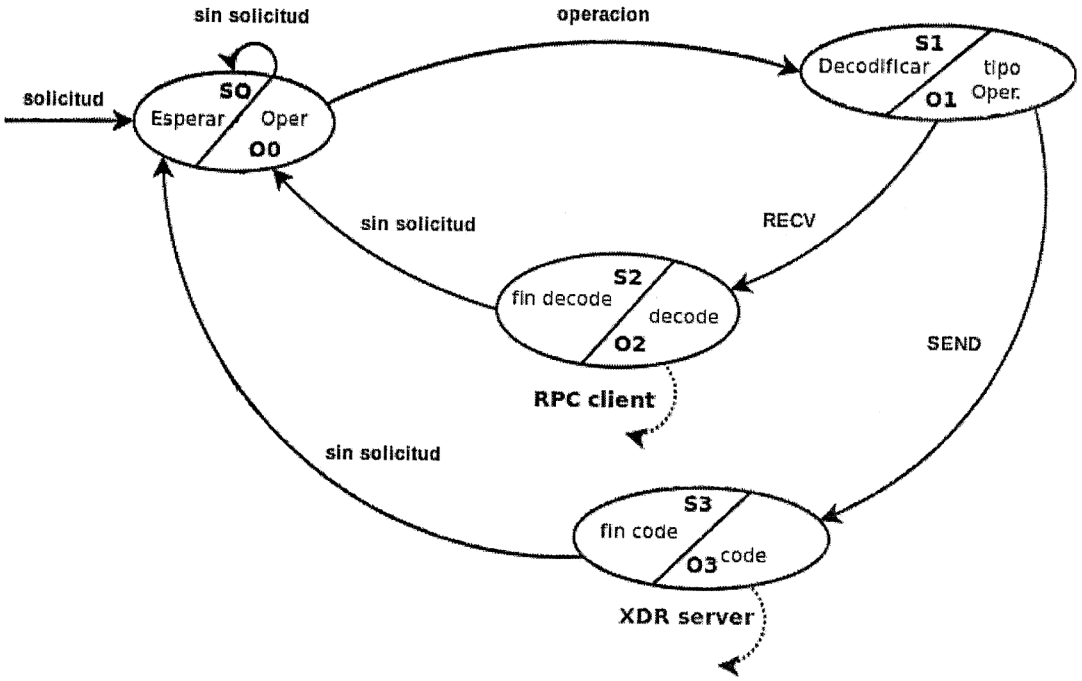


Figura C2. *Máquina de estados finito de XDR cliente* (imagen propia)

Firma Estudiante:	Firma Docente Supervisor:	Firma docente tutor TAPTA:	Firma tutor Organizacional:

Anexo D

En la presente sección, se muestra cómo se instalan las herramientas seleccionadas, paso a paso.

D.1 JAVA (JDK y JRE)

Para instalar JAVA en su versión 8, se ejecuta en terminal:

```
[user@localhost ~]$ sudo pacman -S jdk8-openjdk jre8-openjdk
```

Una vez instalado, se procede a configurar las variables de entorno necesarias y requeridas por JAVA. Para esto, se ejecuta en terminal:

```
[user@localhost ~]$ export PATH=/usr/lib/jvm/java-8-openjdk/jre/bin/:$PATH
```

Sin este último paso, resulta imposible ejecutar y compilar código JAVA.

D.2 Netbeans 8.2

```
[user@localhost ~]$ sudo pacman -S netbeans
```

Para configurar, el entorno de desarrollo, es necesario abrir el programa y dirigirse a la barra de menú principal; de allí, seleccionar, sucesivamente:

1. "*Tools*".
2. Luego seleccionar en el menú "*Plugins*".
3. Ir a la solapa "*Installed*" y seleccionar todas las opciones.
4. Pulsar el botón "*Activate*".
5. Reiniciar Netbeans.

D.3 CloudSim

Primero, hay que descargar el código fuente del *framework*. Para esto hay que dirigirse a "https://github.com/Cloudslab/cloudsim/archive/master.zip". Como resultado de este paso, se tiene un archivo llamado "*master.zip*".

Desde Netbeans, dirigirse a la barra de menú principal y seleccionar:

1. "*File*".
2. Del nuevo submenú seleccionar "*From zip*".
3. Aparece una nueva ventana, seleccionar el archivo "*master.zip*".
4. Presionar botón "Import".

Esta última serie de pasos extrae y genera nuevos proyectos de *cloudsim*. Específicamente, se generan cuatro: "*cloudsim*", "*cloudsim-examples*", "*distribution*" y "*modules*". Se seleccionan los cuatro proyectos y, desde el menú generado, al presionar *click* derecho, se debe elegir la opción "*Build with dependencies*". Este proceso puede tomar varios minutos. Una vez finalizado, estará el *framework* disponible para su uso.

Anexo E

En esta sección, se detalla cómo fue el proceso de instalación del entorno virtual en *Amazon Web Services*. Se muestra cómo se crea una AMI (*Amazon Machine Image*), tanto para el cliente como el servidor NFS. Además, se describe todo el proceso de instalación de las diferentes herramientas.

Para lograr esto, se utiliza el servicio EC2 (*Elastic Compute Cloud*), que ofrece capacidad de cómputo adaptable en la nube (Amazon, 2019). Algunas de las características de este servicio son las que enumeramos aquí:

- Brindar entornos virtuales, conocidos como instancias.
- Obtener plantillas preconfiguradas (AMI), que contienen sistemas operativos y paquetes de software destinados a determinadas actividades, como es el caso del servidor de base de datos no relacionales o de los servicios web.
- Disponer de varios tipos de instancias de acuerdo a CPU, memoria, capacidad de almacenamiento y de red.
- Disponer de volúmenes de almacenamiento persistentes (*Elastic Block Storage*) y volátiles.
- Contar con la posibilidad de elegir zonas físicas desde donde se lanzan las instancias y se encuentran los volúmenes de almacenamientos; por ejemplo, EE.UU. *East* (N. Virginia) y EE.UU. *East* (Ohio).
- Acceder a redes virtuales privadas (*Virtual Private Clouds*), a las que se pueden asignar un pool de direcciones privadas y públicas (provistas o no por *Amazon)*.

E.1 Creación de instancias

El método que se muestra a continuación se ejecuta, por primera vez, dos veces, para crear una AMI cliente y otro servidor. Luego, se crean instancias, a partir de las imágenes que modificamos con el mismo método detallado, con la salvedad de que, al momento de seleccionar el tipo de AMI, se elige la imagen modificada.

Para crear una instancia, desde la pestaña de servicios, se selecciona la opción EC2. Esto se muestra en la figura E1.

Figura E1. *Servicios de la categoría informática* (imagen propia)

Este paso redirige al panel de administración de instancias. Desde este panel, se elige la opción "*Launch Instance*", tal como se muestra en la figura E2:

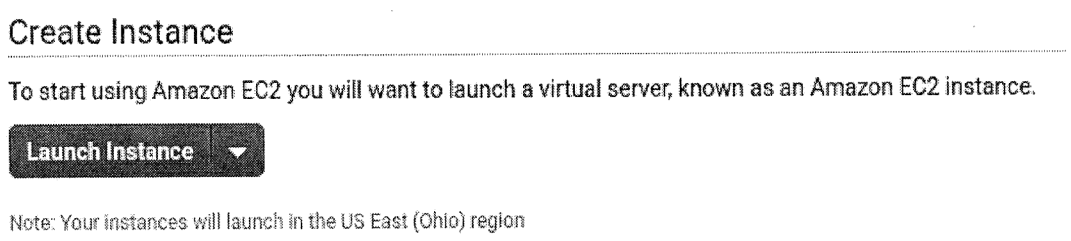


Figura E2. *Botón que permite ejecutar una instancia (imagen propia)*

Este paso redirige a un nuevo panel secundario, como se ve en la figura E3, donde se selecciona el tipo de AMI a utilizar. Para este caso, se trabaja con la de *Red Hat Enterprise Linux 8*, que ofrece virtualización HVM (*Hardware Virtual Machine*), ya que el sistema host de *Amazon* EC2 emula una parte o todo el hardware subyacente que se presenta al invitado. Por otro lado, cuenta con una unidad de estado sólido (SSD).

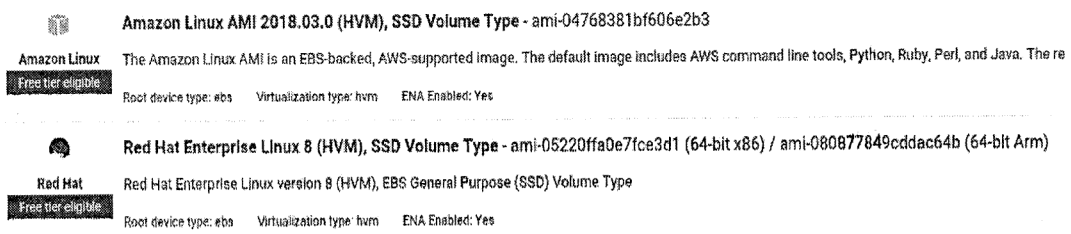


Figura E3. *AMI de Red Hat Enterprise Linux 8* (imagen propia)

Luego de seleccionar la AMI, hay que elegir el tipo de instancia, como se grafica en la figura E4. Cada tipo de instancia tiene diferentes características a nivel hardware. Para este caso, se trabaja con la de tipo "t2.micro". Las características de este tipo son las siguientes:

- ECU (Unidad de cómputo) variable.
- 1 vCPUs, 2.5 GHz,Intel Xeon Family.
- 1 GiB de memoria RAM.

- Almacenamiento EBS.

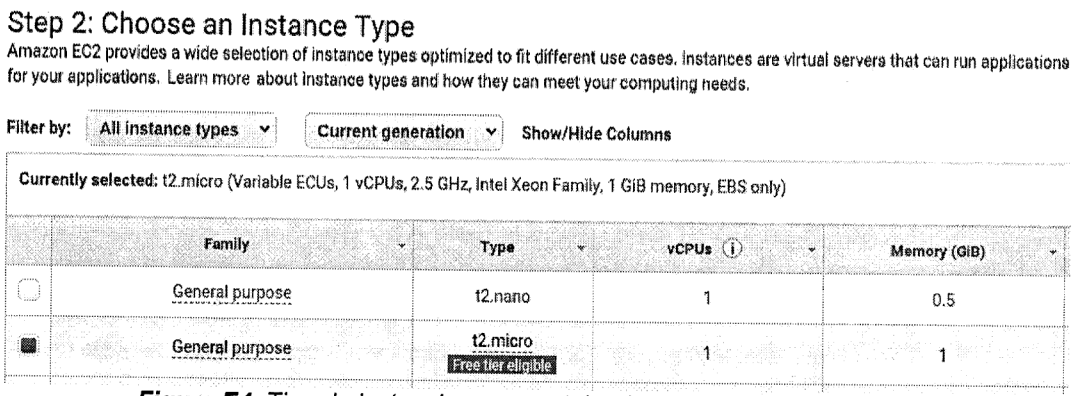


Figura E4. *Tipo de instancia capa gratuita de Amazon* (imagen propia)

Otro aspecto importante a configurar es el grupo de seguridad. Funciona de manera similar a un *firewall*: se definen reglas de entrada y salida permitiendo o no cierto de tráfico y acceso a la instancia. En la figura E5, se observa las reglas que se agregan.

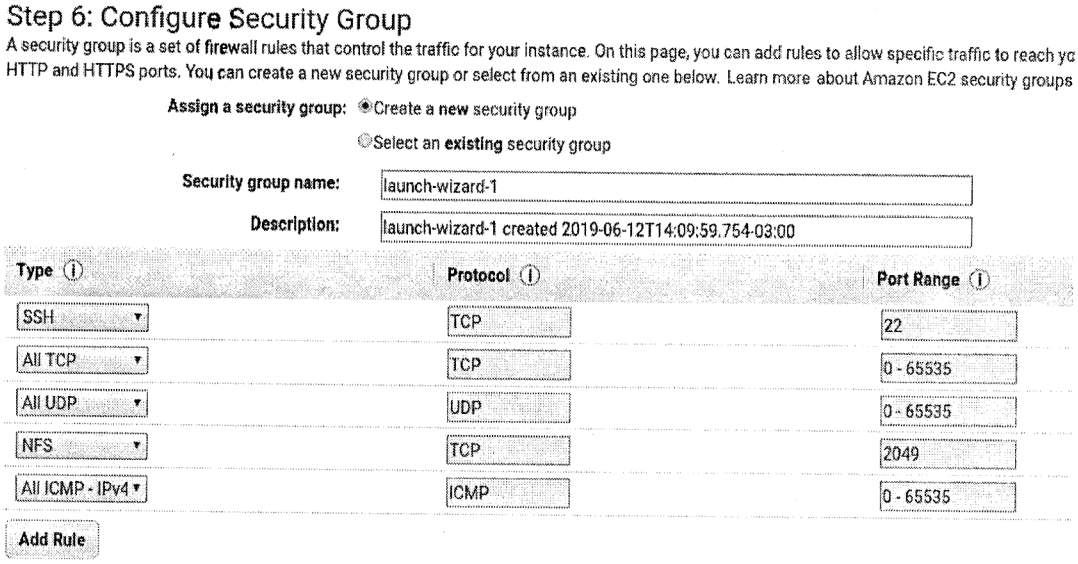


Figura E5. *Reglas del grupo de seguridad* (Imagen propia)

Firma Estudiante:	Firma Docente Supervisor:	Firma docente tutor TAPTA:	Firma tutor Organizacional:

Una vez definidas todas las reglas, se procede a ejecutar la instancia, presionando el botón "*Launch*". Esto abre una nueva ventana en el navegador en donde se selecciona el nombre de la clave "ssh" para poder acceder remotamente. Esto se muestra en la figura E6:

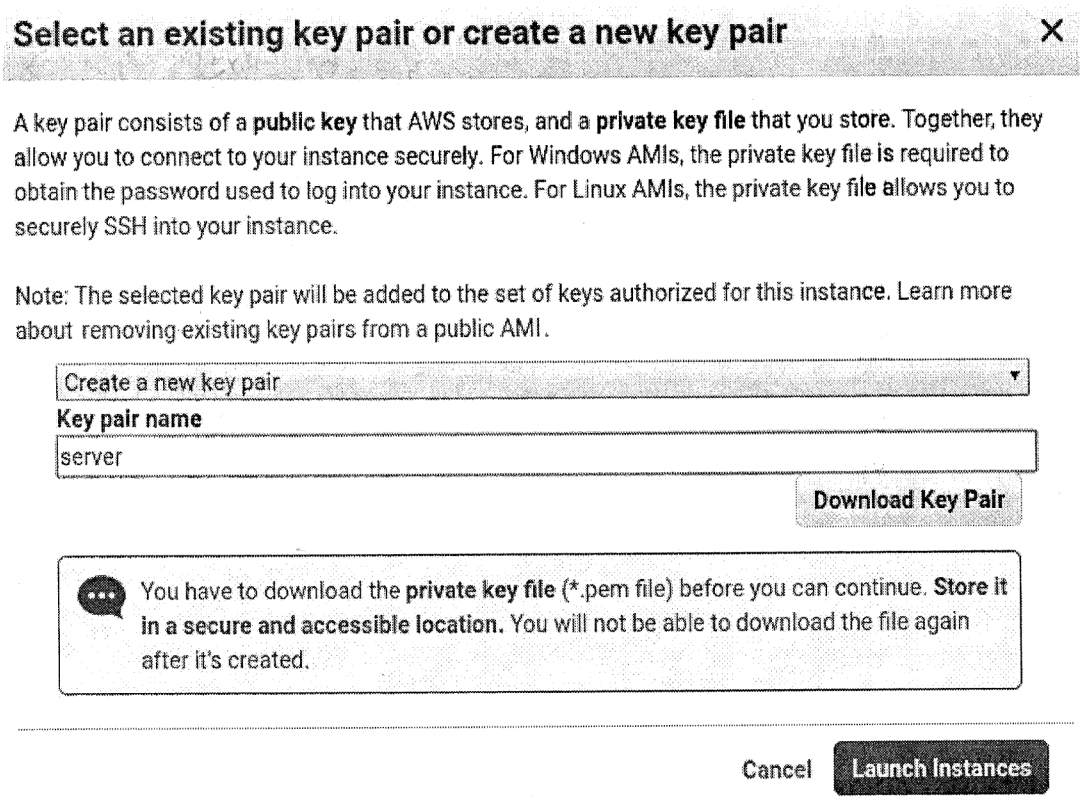


Figura E6. *Creación de claves de seguridad* (imagen propia)

Una vez que se descarga la clave, el sitio redirige al panel de administración de instancias.

E.2 Instalación y configuración del servidor NFS

Antes de acceder remotamente a la instancia creada, es necesario modificar los permisos de acceso de la clave privada. Para esto, se ejecuta en terminal:

```
[localuser@localhost ~]$ sudo chmod 700 Descargas/server.pem
```

Luego, se procede a conectarse a la instancia con el siguiente comando:

```
[localuser@localhost ~]$ ssh -i /home/pegroman/Descargas/server.pem ec2-user@ec2-13-58-10-49.us-east-2.compute.amazonaws.com
```

Una vez conectados a la instancia, instalamos los paquetes correspondientes e inicializamos los servicios relacionados a NFS:

```
[ec2-user@ip-172-31-2-55 ~]$ sudo yum install -y nfs-utils
[ec2-user@ip-172-31-2-55 ~]$ sudo systemctl enable nfs-server rpcbind
[ec2-user@ip-172-31-2-55 ~]$ sudo systemctl start nfs-server rpcbind
```

El siguiente paso consiste en crear el directorio al que los clientes van a acceder remotamente:

```
[ec2-user@ip-172-31-2-55 ~]$ mkdir nfsdir
```

Por último, se debe configurar en el archivo "/etc/exports", es decir, el directorio a acceder.

```
[ec2-user@ip-172-31-2-55 ~]$ echo "/home/ec2-user/nfsdir *(rw,sync,no_root_squash)" >> /etc/exports
```

El archivo "/etc/exports" controla los sistemas de archivos a exportar, para ser accedidos. A continuación se describen los parámetros:

- /home/ec2-user/nfsdir: el directorio a ser accedido.
- *: todos los clientes de la red pueden montar el directorio.
- rw: los clientes pueden escribir y leer en el directorio.
- sync: responde a peticiones únicamente después de que los cambios se vean reflejados en almacenamiento estable.
- no_root_squash: permite que el usuario *root* de los clientes sea el mismo usuario *root* en el sistema de ficheros que se exporta.

E.3 Instalación y configuración del cliente NFS

Se procede a crear una nueva instancia con el método mencionado en la sección 1, de este mismo anexo.

Una vez que la instancia está corriendo, se procede a acceder remotamente. Luego, se instalan los siguientes paquetes:

```
[ec2-user@ip-172-31-2-222 ~]$ sudo yum install -y nfs-utils git mpich
[ec2-user@ip-172-31-2-222 ~]$ sudo yum groupinstall "Development Tools"
[ec2-user@ip-172-31-2-222 ~]$ sudo yum groupinstall "Additional Development"
```

Luego se instala IOR. Para esto, hay que clonarlo desde su repositorio y compilarlo a mano:

```
[ec2-user@ip-172-31-2-222 ~]$ git clone https://github.com/hpc/ior.git
[ec2-user@ip-172-31-2-222 ~]$ cd ior/
[ec2-user@ip-172-31-2-222 ior]$ ./bootstrap
[ec2-user@ip-172-31-2-222 ior]$ ./configure
[ec2-user@ip-172-31-2-222 ior]$ ./make
```

Por último, se crea el directorio donde se va a montar el directorio del servidor y se

edita el archivo "/etc/fstab", para que se monte automáticamente el directorio del servidor:

```
[ec2-user@ip-172-31-2-222 ~]$ sudo mkdir /mnt/awsnfs
[ec2-user@ip-172-31-2-222 ~]$ echo "172.31.2.55:/home/ec2-user/nfsdir
/mnt/awsnfs  nfs defaults 0 0" >> /etc/fstab
```

E.4 Creación de una AMI a partir de una instancia

Para crear una imagen de máquina de *Amazon*, se procede a hacer clic derecho sobre la instancia de interés y se debe seleccionar la opción "Create image" de la entrada "Image". Esto se grafica en la figura E7.

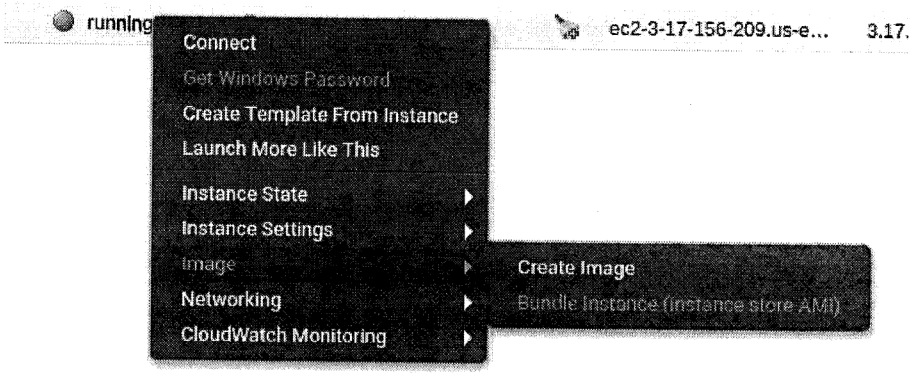


Figura E7. *Menú de creación de AMI* (imagen propia)

Para finalizar la creación de la imagen, es necesario nombrarla, agregar una descripción y definir tamaño de disco. En la figura E8, se observa este procedimiento.

Una vez realizados estos pasos, ya se encuentra disponible la imagen en

cuestión, para ser utilizada tantas veces como sea necesario.

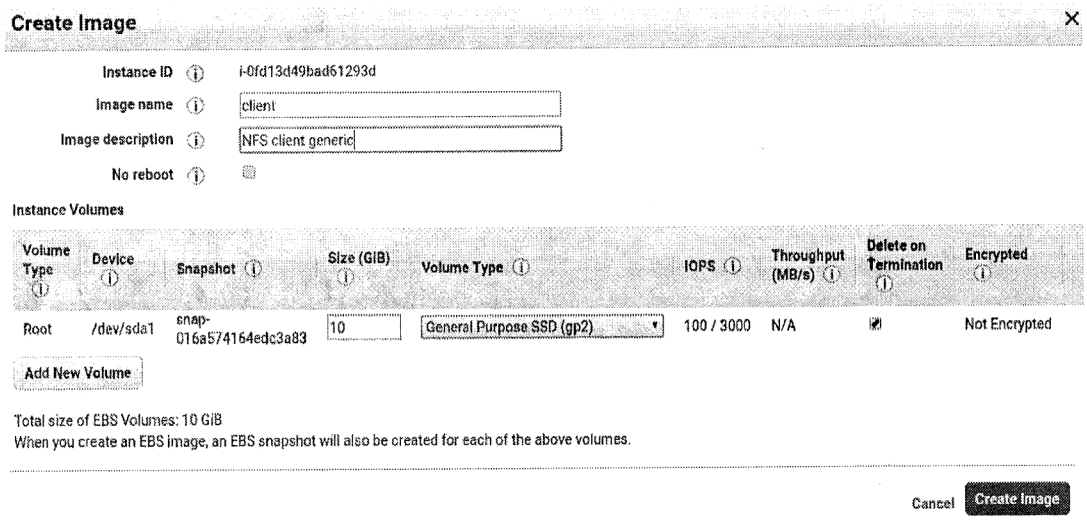


Figura E8. *Configuración de AMI a crear* (imagen propia)

Firma Estudiante:	Firma Docente Supervisor:	Firma docente tutor TAPTA:	Firma tutor Organizacional:

Índice de figuras

Firma Estudiante:	Firma Docente Supervisor:	Firma docente tutor TAPTA:	Firma tutor Organizacional:

Índice de tablas