Unidad 4 / Escenario 7

Lectura fundamental

Cloud computing y paradigmas de la computación distribuida

Contenido

1 *Cloud computing*

2 Algoritmos de sistemas distribuidos

Palabras clave:
Cloud computing, IaaS, PaaS, SaaS, servicio, algoritmos distribuidos, exclusión mutua, tolerancia a fallos, Chandy, Berkeliy, Lamprot

Introducción

Esta Lectura fundamental tiene como propósito explorar uno de los paradigmas más importantes de la computación distribuida: el *cloud computing*. La computación en la nube se ha convertido en una de las soluciones tecnológicas más utilizadas para proveer de forma dinámica y escalable, recursos computacionales teniendo en cuenta la forma en la que el usuario final consumirá el servicio.

En el modelo de prestación de servicios del *cloud* existen tres métodos de prestación de servicios: infraestructura, plataforma y *software* (Furht y Escalante, 2010).

El *cloud* se ha convertido en una tendencia tan grande que en el mercado existen proveedores que prestan servicios de infraestructura, de forma masiva, proveedores como Amazon EC2, Microsoft Azure, HP Helion, Google cloud, etc. Cada uno de estos ha desarrollado tecnologías privativas, pero también existen implementaciones basadas *en software* libre tales como Open Stack y Open Nebula.

En este documento también se va a profundizar en los fundamentos de los algoritmos pioneros que han hecho posible la computación distribuida y el comportamiento del *hardware* y el *software* para tolerar fallos.

1. Cloud computing

El término *cloud computing* nace a partir del concepto de *hosting* cuando se popularizaron las páginas Web y se requería un host para alojarlas y esto en conjunto con las comunicaciones con base en las implementaciones de las redes privadas virtuales -VPN, requeridas para comunicar datos a una red que no es accesible a través de Internet.

La computación en la nube fue acuñada para hacer énfasis en un servicio de computación usando Internet que se puede prestar a cualquier usuario, desde cualquier lugar del mundo, en cualquier instante de tiempo, sin tener en cuenta el lugar en donde estén ubicados los *hosts* de la nube, lo que significa que estos recursos deberán estar publicados todo el tiempo.

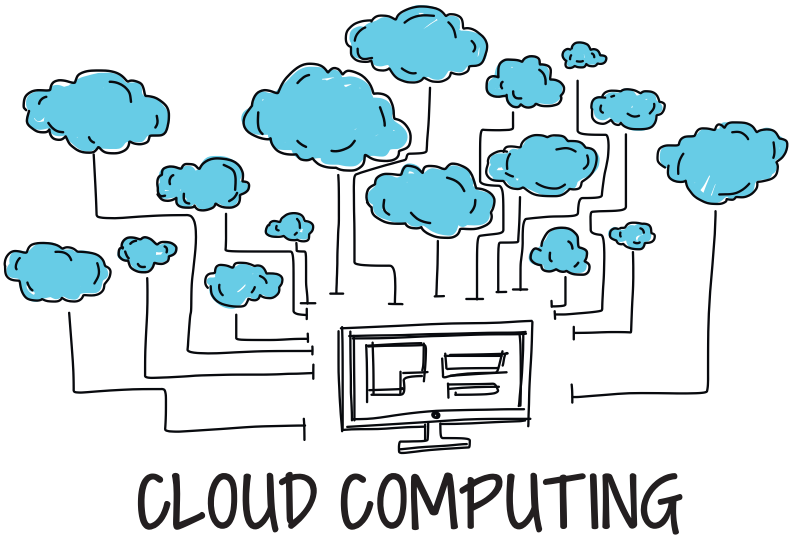

Figura 1. Paradigma de computación en la nube
Fuente: Rarelif

El concepto de *cloud computing* se atribuye a John McCarthy (Armbrust et al., 2010), el cual fue el primero en dar a conocer que la tecnología en la que el tiempo de un computador era compartido llevaría a un futuro donde la utilización óptima de los recursos podrían hacer que varios usuarios usaran una misma máquina y, así mismo, maximizar las ganancias prestando un servicio de cómputo.

Cloud computing es una forma de ver la tecnología orientada a un modelo de negocio encaminado a proveer servicios sin importar su ubicación o su capacidad como se muestra en la Figura 1.

El Instituto Nacional de Normas y Tecnología (NIST) dice que:

> "La computación en la nube es un modelo para habilitar un grupo de recursos computacionales ubicuos, convenientes y en demanda, con capacidad de acceso a la red, que pueden ser aprovisionados rápidamente y desplegados con un mínimo de esfuerzo de administración o interacción con el proveedor del servicio"

Este modelo de nube está compuesto de cinco características esenciales, tres modelos de servicios y cuatro modelos de despliegue (Mell & Brance, 2011).

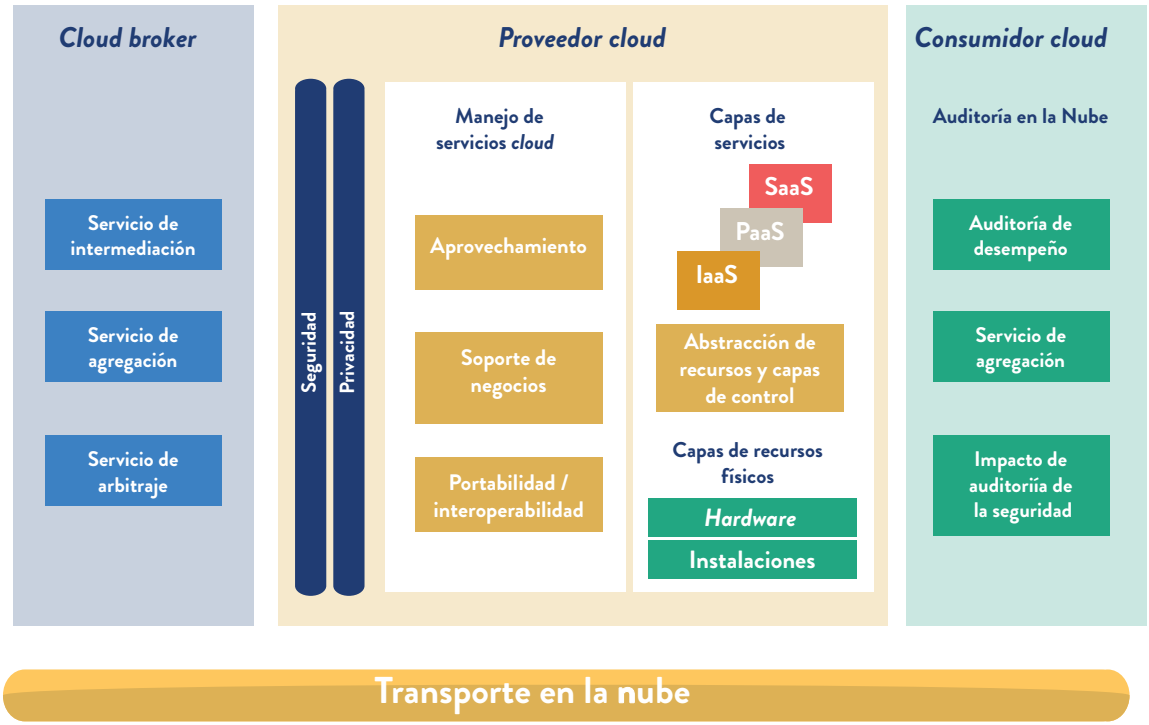

Figura 2. Modelo conceptual de una *cloud*

Fuente: elaboración propia

La Figura 2 presenta una descripción general de la arquitectura de la computación en la nube del NIST, que identifica a los principales actores, sus actividades y funciones en la computación en la nube. El diagrama muestra una arquitectura genérica de alto nivel y su objetivo es facilitar la comprensión de los requisitos, usos, características y estándares de la computación en la nube.

Tabla 1. Actores en una *cloud*

Actores	Definición
Consumidor *cloud*	Una persona u organización que mantiene una relación comercial y utiliza el servicio de los proveedores de la nube.
Proveedor *cloud*	Una persona, organización o entidad responsable de poner un servicio a disposición de las partes interesadas.

Auditor *cloud*	Un ente que puede realizar una evaluación independiente de los servicios en la nube, las operaciones del sistema de información, el rendimiento y la seguridad de la implementación de la nube
Broker *cloud*	Una entidad que administra el uso, el rendimiento y la entrega de servicios en la nube y negocia las relaciones entre los Proveedores de la nube y los Consumidores de la nube
Transportador *cloud*	Un intermediario que proporciona conectividad y transporte de servicios en la nube a los consumidores de la nube

Fuente: elaboración propia

La computación en la nube es un servicio para prestar una serie de recursos computacionales, de almacenamiento, de procesamiento o aplicaciones que se encuentran en un sistema bajo demanda accedido por una gran red de datos, el cual le brindan al usuario final un mínimo esfuerzo.

Por otro lado, y desde una perspectiva de administración, se logra el manejo eficiente de los recursos computacionales compartidos, las organizaciones que prestan este servicio están encaminadas en prestar beneficios a sus clientes finales tales como escalabilidad, flexibilidad, etc. (Tanenbaum & Steen, s.f.).

1.1. Características

Dentro del Manifiesto de la NIST se encuentran que esencialmente existen ocho características esenciales del *cloud computing*, pero adicionalmente a estas existen ocho que son de tipo común (Mell & Brance, 2011).

1.1.1. Características esenciales

Las características generales están orientadas al servicio a como la nube se debe comportar y que capacidades debe tener para estar enmarcadas en el concepto de la NIST de *cloud computing*.

Autoservicio por demanda: un consumidor de manera unilateral puede aprovisionar capacidades computacionales de manera automáticamente, tales como almacenamiento, etc., usando la red sin necesidad de interactuar con el proveedor del servicio.

Uso de redes de banda ancha: las capacidades sobre la red deben estar disponibles basándose en mecanismos estándares que promuevan el uso de clientes heterogéneos delgados tales como dispositivos móviles, computadores personales, etc.

Agrupación de recursos: el proveedor de recursos computacionales debe agruparlos para dar servicio a varios consumidores, usando un modelo multi usuario (*multi-tenant*), con recursos virtuales o físicos que son asignados de manera dinámica y que se pueden reasignar de acuerdo con la demanda del consumidor.

Rápido crecimiento: la capacidad puede ser aprovisionadas y/o liberada y muchas veces de manera automática, con el fin de crecer o decrecer rápidamente para satisfacer la demanda.

Medición del servicio: Los recursos de un sistema *cloud* se controlan y optimizan de forma automática, en donde una capa de medición es necesaria en algún punto para la abstracción del nivel de apropiación del servicio. Los recursos generalmente son monitoreados, controlados y reportados.

1.1.2. Características comunes

Así como su nombre lo dice están relacionadas con la propiedad natural de los sistemas *cloud* ser sistemas distribuidos naturales.

Escalabilidad: hace referencia a la capacidad de crecer según la demanda del usuario final de manera ágil y autónoma, esto aplicado al modelo de negocio es una gran ventaja ya que el usuario final de los recursos computacionales podrá costear solo los recursos utilizados.

Homogeneidad: está dada de acuerdo con la forma en la que los recursos computacionales se adaptan a la necesidad del sistema siendo un mismo a través de la solución utilizada por el usuario final.

Virtualización: complementando la característica anterior, la virtualización es la capacidad de anteponer o crear una capa intermedia de comunicación entre el *hardware* y la aplicación final, aportando a esa homogeneidad sin importar la parte física de la solución.

***Software* de bajo costo:** teniendo en cuenta el modelo de negocio bajo demanda y dividiendo la necesidad en dos roles, el prestador del servicio y el cliente, el cliente puede arrendar una licencia de un *software* específico y el proveedor de servicio pude desplegarlo de manera ágil y dinámica en su pool de recursos.

Computación resiliente: es el manejo de los fallos (tolerancia a fallos) con el fin de poder brindar ciertos estándares de confiabilidad del servicio y mejorar la continuidad de negocio del cliente final, una vez más sin importar la ubicación geográfica y la localización de la información.

Distribución geográfica: esta característica es el resultado del conglomerado de habilidades y de servicios desplegados en el *pool* de recursos que combinados resultan en el olvido de que las maquinas pueden estar en diferentes lugares físicos, tales como edificios, ciudades, etc.

Orientación al servicio: una arquitectura *cloud* debe estar orientada al servicio a que los recursos, aplicaciones, interfaces, datos, etc., deben presentarse al usuario final como servicio.

Seguridad: debido a que todos los datos se encuentran bajo la responsabilidad del proveedor del servicio debe existir una capa de seguridad bastante alta, estas se pueden diferenciar en Seguridad del Servicio, Seguridad del Explorador, Seguridad de los Datos, etc. (Mell & Brance, 2011).

1.2. Modelos de despliegue

Los modelos de despliegue hacen referencia hacia a quien están dirigidos los servicios del *cloud*.

1.2.1. Cloud privada

Le da a la organización del mismo dominio el acceso y uso exclusivo de los recursos y de la infraestructura computacional. Se puede manejar desde otra organización o por una tercera parte y puede estar implementada bajo las premisas de la organización o relegada a una compañía de *hosting*.

La *cloud* privada no es un producto, pero es una aproximación para el diseño, implementación y manejo de los servidores de la organización, aplicaciones y recursos del centro de datos reduciendo la complejidad, incrementando la estandarización y automatización, y proporcionando elasticidad para evolucionar, tanto el negocio como los requerimientos técnicos de la organización.

Los *clouds* privadas aplican los mismos principios usados de escalabilidad y administración que los grandes *clouds* públicas hacia el interior del centro de datos de la organización.

1.2.2. *Cloud* pública

Es una de las formas en las que la infraestructura y los recursos computacionales se ponen a disposición del público en general, esta infraestructura debe tener acceso a redes públicas como por ejemplo Internet.

Una *cloud* pública les pertenece a organizaciones que venden los servicios *cloud* y sirve a un grupo de clientes heterogéneos.

1.2.3. *Cloud* Hibrida

Una *cloud* híbrida es una composición de al menos dos *clouds*, ya sean privadas o públicas, las cuales se comportan como entidades separadas únicas unidas por tecnologías estándares que posibilitan la portabilidad de datos y de aplicaciones (Hogan et al., 2011).

1.2.4. *Cloud* Comunitaria

Está conformada por varias organizaciones, ya sean públicas, privadas o hibridas, la diferencia con este modelo de despliegue es el soporte lo da una comunidad que comparte un mismo fin.

1.3. Modelos de Servicio

Los modelos de servicio dividen o segmentan las responsabilidades del proveedor de servicio, desde la parte *hardware* hasta el *software*.

1.3.1. Infraestructura como servicio

IaaS (*Infrastructure as a Service*) brinda la capacidad del *hardware*, establece al usuario en un punto más cercano al *hardware* en que este se hace cargo del despliegue y montaje de su aplicación para proveer un servicio para el mismo o una tercera persona adicional. En esta capa el usuario despliega o migra una solución. Ver Figura 3.

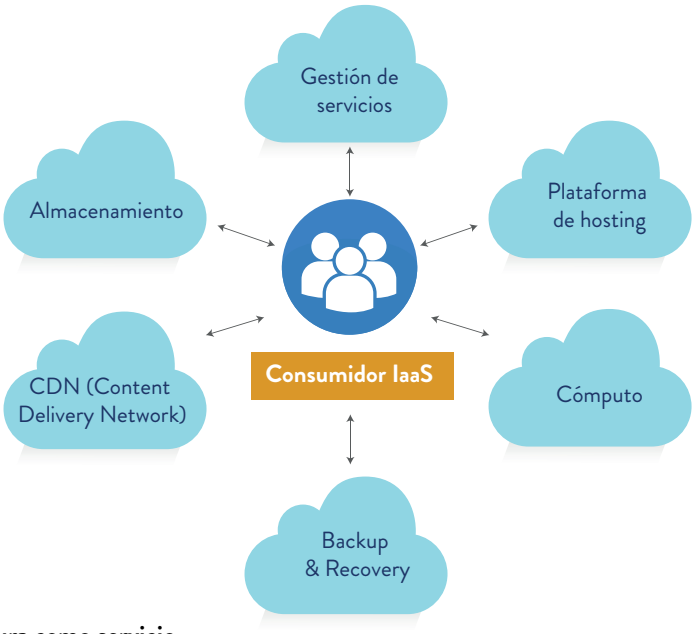

Figura 3. Infraestructura como servicio

Fuente: elaboración propia

1.3.2. Plataforma como servicio

PaaS (*Platform as a Service*) es muy similar a IaaS pero se diferencia porque adiciona valor a los servicios y viene en dos diferentes tipos:

Una plataforma colaborativa para el desarrollo de *software*, enfocada en el manejo la oleada de trabajos sin que le interese el origen de los datos que están siendo utilizados por la aplicación. Por ejemplo: Heroku.

Con base en datos Plataforma opios, se desarrolla *software* a la medida. PaaS puede verse como un método de creación

o testeo de aplicaciones con datos en

común. ejemplo: SAP, Oracle financial, etc. Ver Figura 4.

En esta capa el usuario construye encima de un grupo de utilidades.

Figura 4. Plataforma como servicio

Fuente: elaboración propia

1.3.3. Software como servicio

SaaS (*Software as a Service*) es el proveedor licencia una aplicación al cliente como un servicio por demanda, esto sucede a través de un modelo de subscripción llamado "*pay as you go*".

SaaS puede ser el aspecto más conocido de *cloud computing*, pero desarrolladores y organizaciones de todo el mundo están migrando a PaaS, debido a que mezcla la simplicidad del SaaS con el poder del IaaS, logrando así una gran dinámica en las organizaciones. Ver Figura 5.

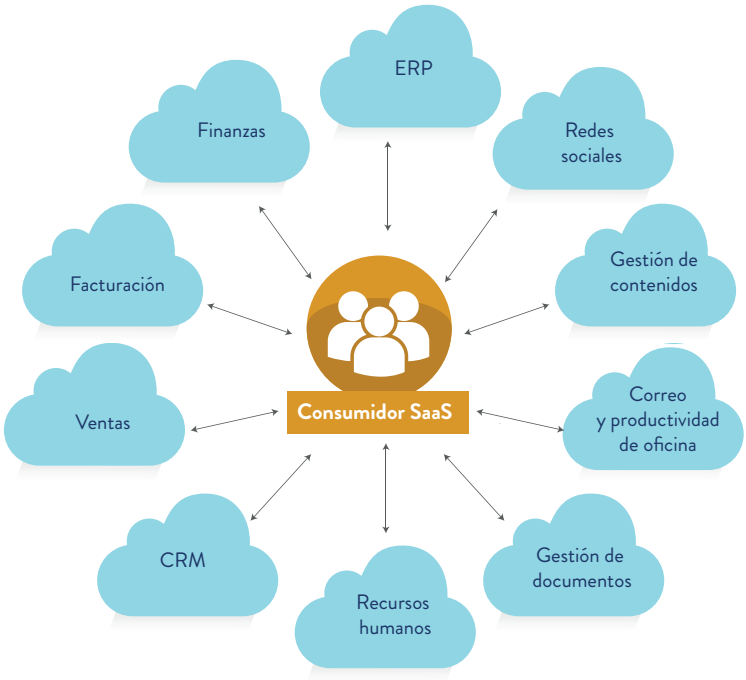

Figura 5. *Software* como servicio
Fuente: elaboración propia

1.3.4. Plataformas para el aprendizaje de montaje y desarrollo de una cloud

El proceso de aprendizaje para montar una *cloud computing* en un portátil es muy sencillo. Se crea una organización de tres máquinas virtuales donde una de ellas sea la máquina máster (locomotora de la cloud). En la máquina *server* se instala el OpenStack Server, el cual se puede hacer en CentOS, Red Hat, Ubuntu, etc.

Cómo mejorar...

Para profundizar sobre el tema puede consultar en
https://www.server-world.info/en/note?os=CentOS_7&p=openstack_mitaka

Posteriormente, se instala el sistema operacional, se actualiza y se comienza preparando el ambiente (requerimientos) y una vez alistado, se configura el sistema de seguridad llamado Keystone, después se configura el servidor de imágenes Glance (imagen es una ISO que se desarrolla en un servidor virtual, como por ejemplo instalar una base de datos Oracle y luego convertir esa máquina virtual en una ISO, para subirla al Glance). Hecho esto, se configura el motor del OpenStack Server que es el Nova. El Nova rige toda la administración de OpenStack, se comporta como el orquestador de toda la arquitectura.

Paso seguido se instalan todos los demás componentes como dice el manual del enlace.

Cuando la máquina máster queda lista, hay que empezar a agregarle vagones a la locomotora, es decir, usted va configurando servidores con OpenStack Cliente y los va matriculando en el DNS y en la configuración del máster, para que se comuniquen y en estos servidores, que se llaman contendores, es donde se hospedan las imágenes extendidas de las ISO (es decir, las ISO se convierten en un computador virtual que presta un servicio) que se cargaron en el máster.

En la Figura 6, se puede observar la red de servidores de Hospedaje

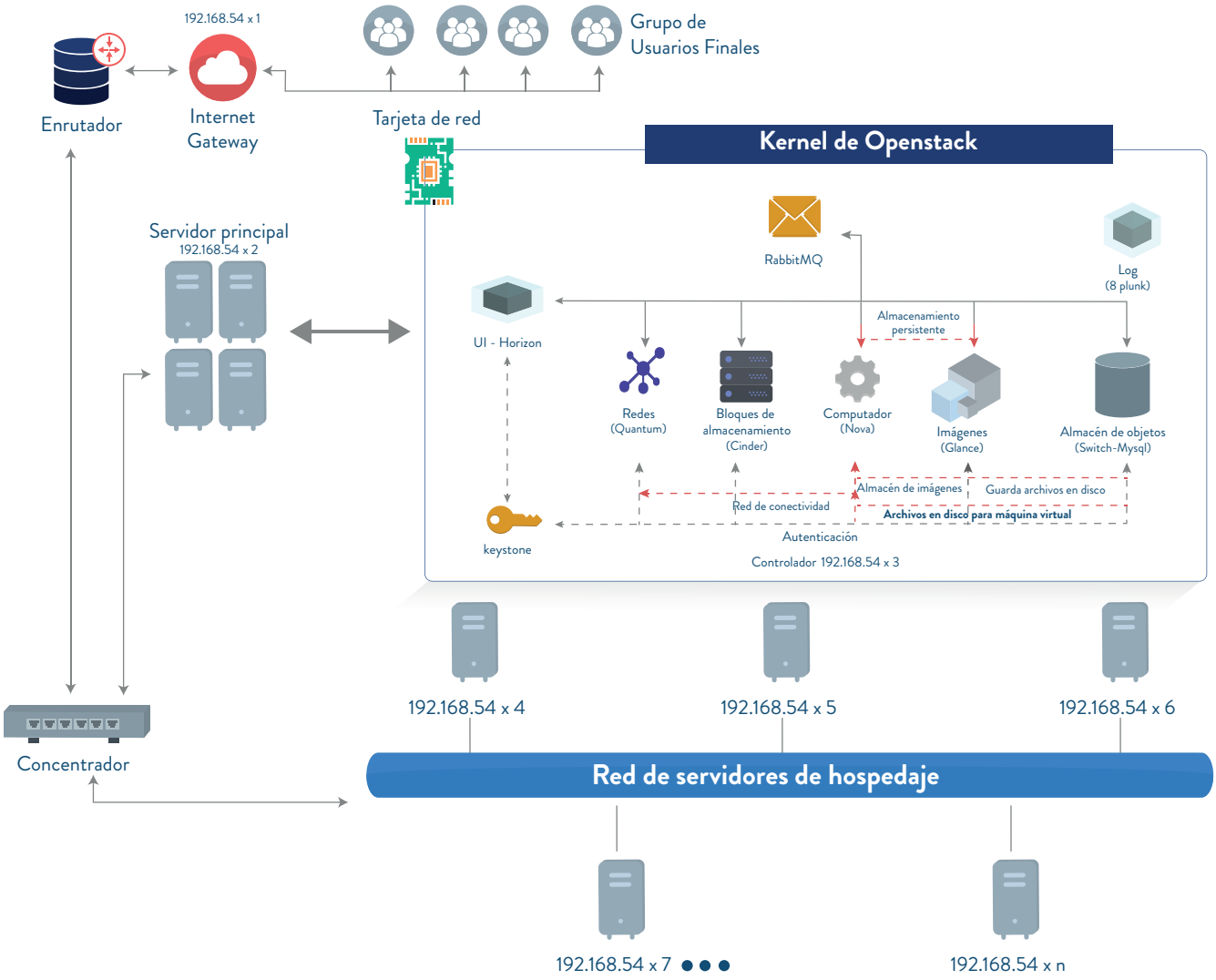

Figura 6. Modelo de un servidor máster de una *cloud* OpenStack

Fuente: Politécnico Grancolombiano adaptado de Cardozo, V., Bernal, M. y Sierra, J. (2015).

2. Algoritmos de sistemas distribuidos

Un algoritmo distribuido es un algoritmo diseñado para ejecutarse en una malla computacional que consta de un conjunto de procesadores interconectados. Los algoritmos distribuidos se utilizan en muchas áreas de aplicación variada de la computación distribuida, como las telecomunicaciones, la computación científica, el procesamiento distribuido de información y el control de procesos en tiempo real.

Los problemas estándar resueltos por algoritmos distribuidos incluyen la elección del líder, el consenso, la búsqueda distribuida, la generación del árbol de expansión, la exclusión mutua y la asignación de recursos.

Los algoritmos distribuidos son un subtipo de algoritmos paralelos, que generalmente se ejecutan al mismo tiempo, con partes separadas del algoritmo que se ejecutan simultáneamente en procesadores independientes, y que tienen información limitada sobre lo que están haciendo las otras partes del algoritmo. Uno de los principales desafíos en el desarrollo e implementación de algoritmos distribuidos es la coordinación exitosa del comportamiento de las partes independientes del algoritmo ante fallas del procesador y enlaces de comunicaciones poco confiables.

La elección de un algoritmo distribuido apropiado para resolver un problema dado depende tanto de las características del problema como de las características del sistema en el que se ejecutará el algoritmo, como el tipo y la probabilidad de fallas del procesador o del enlace, el tipo de comunicación entre procesos que se puede realizar, y el nivel de sincronización de tiempo entre procesos separados.

Vea algunos algoritmos de estos tipos:

2.1. Algoritmo de Chandy y Lamprot

El algoritmo de Chandy y Lamprot funciona utilizando el concepto de marcado de mensajes. Cada proceso que desea iniciar una instantánea registra su estado local (Color azul de la Figura 7) y envía una señal de marcado a cada uno de sus canales salientes. Todos los demás procesos, al recibir un marcador, registran su estado local, el estado del canal desde el cual el marcador ha llegado y luego envían mensajes de marcado a todos sus canales salientes.

Si un proceso recibe un marcador después de haber grabado su estado local, registra el estado del canal entrante desde el cual el marcador venía cargando todos los mensajes recibidos desde que registró su estado local.

Los supuestos del algoritmo son los siguientes:

- No hay fallas y todos los mensajes llegan intactos solo una vez.
- Los canales de comunicación son unidireccionales y FIFO ordenados.
- Hay una ruta de comunicación entre dos procesos en el sistema
- Cualquier proceso puede iniciar el algoritmo de instancias de mensajes.
- El algoritmo de instanciación no interfiere con la ejecución normal de los procesos.
- Cada proceso en el sistema registra su estado local y el estado de sus canales entrantes.

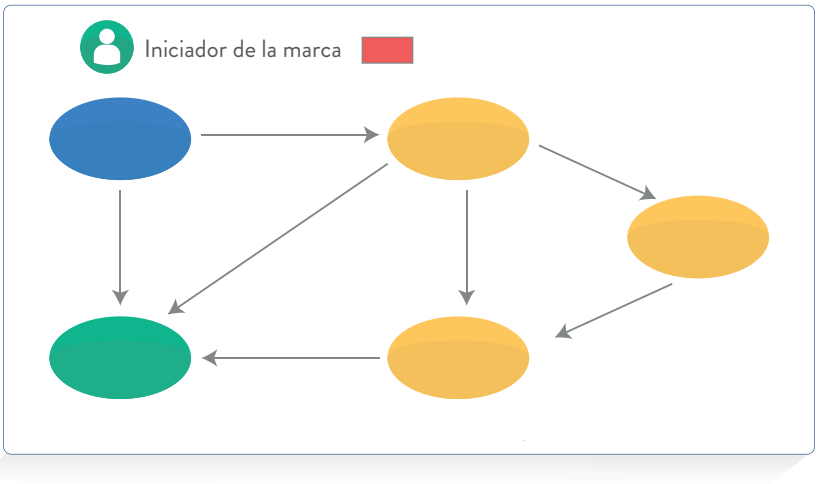

Figura 7. Algoritmo de Chandy y Lamprot

Fuente: elaboración propia.

Algunas de las suposiciones del algoritmo pueden facilitarse utilizando un protocolo de comunicación más confiable, como TCP/IP. El algoritmo se puede adaptar de modo que pueda haber múltiples instantáneas ocurriendo simultáneamente. El algoritmo termina cuando todos hayan recibido el token rojo con el mensaje y se pongan verdes, eso quiere decir que todos están trabajando en el desarrollo del mismo problema que les llevó el token (cuadrado rojo) de la Figura 7.

2.2. Algoritmo de Cristian

El algoritmo de Cristian fue escrito en 1989, con el fin de sincronizar los relojes entre los servidores. En Linux se utiliza en NTP (Network time Protocol), basado en el algoritmo de Cristian para sincronizar el reloj de un servidor contra un servidor de tiempo universal.

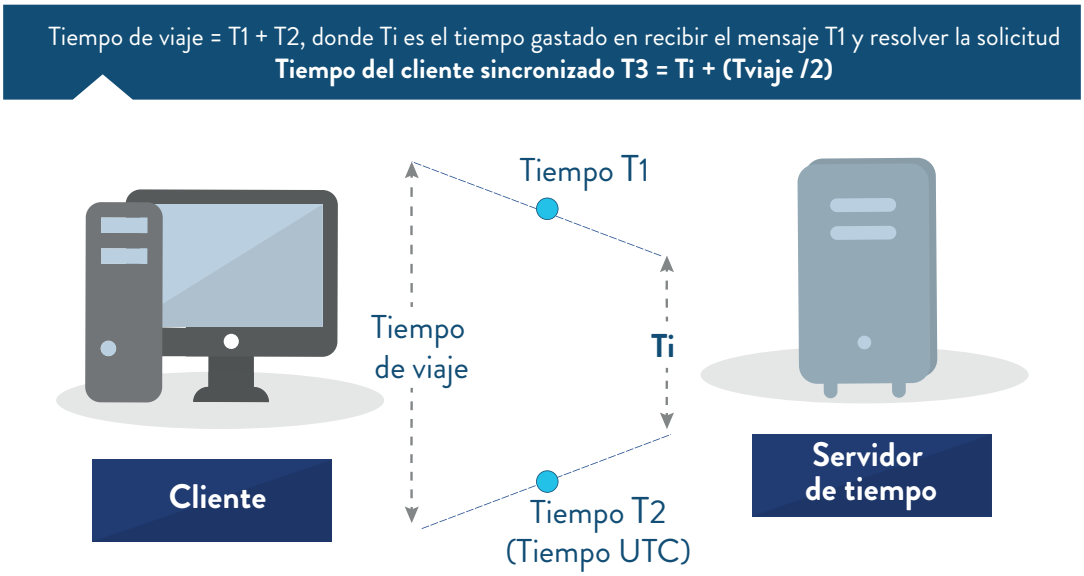

Figura 8. Cálculo de tiempos por algoritmo de Cristian

Fuente: elaboración propia

Entonces el algoritmo de Cristian ajustado se resume como:
El tiempo para sincronizar en el cliente es igual a
Tcliente = Ti + (T2 - T1) / 2 - (Tmin2 - Tmin1).
Donde:
donde T_ {min1} es el tiempo mínimo de solicitud observado y T_ {min2} se refiere al tiempo de respuesta mínimo observado en la red. Ver Figura 8.

2.3. Algoritmo de Berkeley

El algoritmo de Berkeley es una técnica de sincronización de reloj utilizada en sistemas distribuidos. El algoritmo asume que cada nodo en una máquina de la red, no tiene una fuente de tiempo precisa o no posee un servidor UTC (Coordinador Universal de Tiempo). Para ver cómo funciona el algoritmo, se tienen en cuenta dos pasos a saber (ver Figura 8):

- Se elige un nodo individual como nodo maestro de un grupo de nodos en la red. Este nodo es el nodo principal en la red que actúa como maestro y el resto de los nodos actúan como esclavos. El nodo maestro se elige mediante un proceso de elección o algoritmo de elección del líder.
- El nodo maestro hace ping periódicamente a los nodos esclavos y obtiene la hora del reloj utilizando el algoritmo de Cristian.

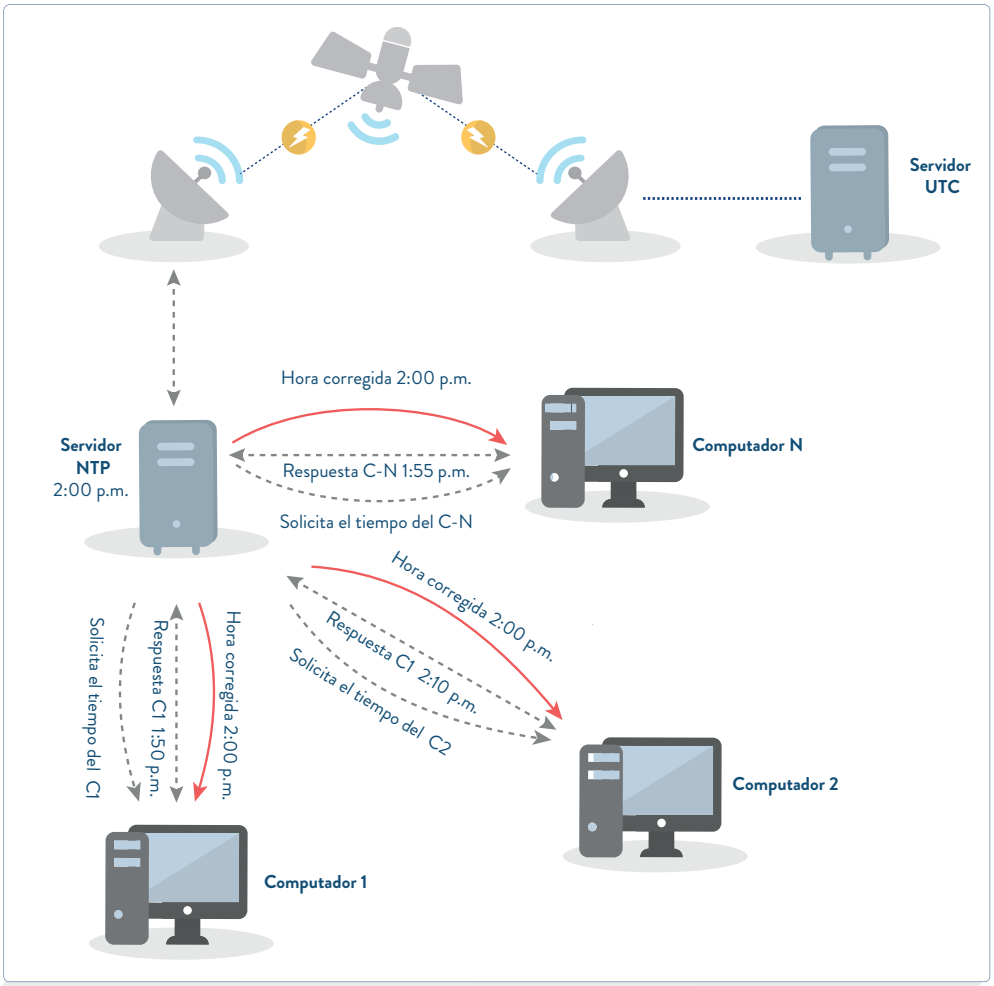

Figura 9. Algoritmo de Berkeley

Fuente: elaboración propia

A diferencia del algoritmo de Cristian, este algoritmo llama a los nodos esclavos y va revisando el tiempo de cada uno, para ver a quien hay que sincronizar. El servidor maestro siempre se auto sincroniza con el reloj Mundial y actualiza su hora, luego evalúa a sus servidores esclavos para ver que correcciones hay que hacer. ver Figura 9.
Como se observa en la figura 9, el servidor NTP, primero se sincroniza con el reloj mundial, luego empieza a sincronizar cada un de sus servidores esclavos, como por ejemplo el computador-1, el cual tiene la hora atrasada en 10 minutos dado que en el servidor NTP son las 2:00. El servidor NTP le solicita el tiempo al servidor C1, como se da cuenta del desfase, entonces le devuelve la hora correcta, es decir, las 2:00 p.m.

2.4. Algoritmos de exclusión mutua

Los algoritmos de exclusión mutua ya sean centralizados o distribuidos, fueron escritos por Dekker y mejorados por Peterson, para manejo de procesos concurrentes que tienen que hacer uso de una sesión crítica en un servidor, como por ejemplo la memoria, el procesador, el disco, la impresora, etc. (Tanenbaum A, página 150).

Peterson desarrollo una mejora al algoritmo de Dekker en 1981, y lo hizo para asignar la región crítica para dos procesos que estaban disputando el uso. Ver Figura 10.

```
bandera[0] = false
bandera[1] = false
turno  // No es necesario asignar un turno
p0: bandera[0] = true
    turno = 1
    while( bandera[1]  && turno == 1);
         { //no hace nada; espera. }
    // sección crítica

    // fin de la sección crítica
    bandera[0] = false
```

```
p1: bandera[1] = true
    turno = 0
    while( bandera[0] && turno == 0):
         { //no hace nada; espera. }
    // sección crítica

    // fin de la sección crítica
    bandera[1] = false
```

Figura 10. Algoritmos de exclusión mutua
Fuente: Tanenbaum A

El método del algoritmo centralizado hace un consenso entre los procesos que están corriendo en el servidor y nombra a un proceso como coordinador.

Figura 11. Asignación de la región critica por algoritmo centralizado

Fuente: Tanenbaum A

Este proceso se encarga de revisar las secciones críticas y llevar el control para poder asignar una región crítica a un proceso que la esté necesitando, para lo cual crea una cola FIFO de solicitudes y va asignando la sección crítica, una vez el recurso haya sido liberado como se puede ver en la Figura 11, en donde el coordinador el proceso C en (a) indica al proceso 1 que la pila esta vacía y puede asignar la región crítica, pero no así le sucede al proceso 2 en (b), pero no así le sucede al proceso 2, que debe registrarse en la pila y esperar hasta que el proceso 1, libere el recurso. Ver Figura 11.

Este algoritmo falla cuando el coordinador sale del servicio; por esta razón Ricard y Agrawala, diseñaron el algoritmo de exclusión mutua distribuido, que consiste en que: cuando un proceso requiere usar la región crítica, genera un mensaje con un ID, la región crítica que necesita y la hora; lo envía a todos sus compañeros y a él mismo; los procesos reciben el mensaje y si no necesitan el recurso, responden diciendo que no lo necesitan, pero si alguno ya está en la región critica, no responde sino que manda el mensaje a una pila de solicitud de región crítica.

El que está solicitando el recurso, compara su marca de tiempo contra la marca de tiempo de los demás mensajes que están solicitando el recurso. Si su marca de tiempo es la menor, puede tomar el recurso pero si el recurso está ocupado, entonces espera hasta que el recurso quede libre ya que su turno le fue asignado.

Con base en estas metodologías, se evita que ocurran los denominados abrazos mortales, que es una disputa entre los procesos por el derecho a usar un recurso a la vez.

Referencias bibliográficas

Armbrust, M. y Fox, A. et al. (2010). Above the clouds: A berkeley view of cloud computing. *Communications of the ACM*, 53(4), 50–58.

Binildas, A. (2008). *Service Oriented Java Business Integration: Enterprise Service Bus integration solutions for Java developers*. Packt Publishing.

Cardozo, V., Bernal, M. y Sierra, J. *Modelo, diseño y técnicas básicas para implantar un Sistema de computación en la nube*. Recuperado de: http://repository.poligran.edu.co/bitstream/handle/10823/682/Modelo.....%20sistema%20de%20computacion%20en%20la%20nube.pdf?sequence=1&isAllowed=y

Coulouris, Dollimore y Kindberg. *Addison and Wesley Distributed Systems Concepts and Design*.

Foster, I., Kesselman, C., & Tuecke, S. (2001). *The Anatomy of the Grid: Enabling Scalable Virtual Organizations*. The International Journal of High-Performance computing Applications, 15(3), 200–222. https://doi.org/10.1177/109434200101500302

Foster, I., Zhao, Y., Raicu, I., & Lu, S. (2008). *Cloud computing and Grid computing 360-Degree Compared*. 2008 Grid computing Environments Workshop, 1–10. https://doi.org/10.1109/GCE.2008.4738445

Furht, B., & Escalante, A. (2010). Handbook of cloud computing. Springer.

Hogan, M., Liu, F., Sokol, A., Tong, J., Locke, G., & Gallagher, P. D. (2011). *Special Publication 500-291 NIST cloud computing Standards Roadmap-Version 1.0 NIST cloud computing Standards Roadmap Working Group*. Recuperado de: https://www.nist.gov/sites/default/files/documents/itl/cloud/NIST_SP-500-291_Jul5A.pdf

Mell, P., & Brance, T. (2011). *The NIST definition of cloud computing*.

Nicolai M. Josuttis. (2007). *SOA in Practice: The Art of Distributed System Design*. Theory in Practice. O'Reilly Media.

OpenStack Wiki. https://releases.openstack.org/

Scott O. Bradner. (1996). *The Internet Standards Process*. Revision 3. Internet Engineering Task.

Tanenbaum, A. S., & Steen, M. van. (s.f.). *Distributed systems: principles and paradigms*.

Referencias de figuras

Cardozo, V., Bernal, M. y Sierra, J. (2015). *Modelo de un servidor máster de una cloud OpenStack*. [Diagrama] Recuperado de: http://repository.poligran.edu.co/bitstream/handle/10823/682/Modelo.....%20sistema%20de%20computacion%20en%20la%20nube.pdf?sequence=1&isAllowed=y

Relif (s.f.). *Paradigma de computación en la nube.* [Vector]. Recuperado de https://www.gettyimages.es/detail/ilustraci%C3%B3n/cloud-computing-ilustraciones-libres-de-derechos/533430308

Tanenbaum, A. S., & Steen, M. van. (s.f.). *Algoritmos de exclusión mutua.* [Pantallazo]

Tanenbaum, A. S., & Steen, M. van. (s.f.). *Asignación de la región critica por algoritmo centralizado.* [Diagrama]

INFORMACIÓN TÉCNICA

FACULTAD DE
INGENIERÍA, DISEÑO
E INNOVACIÓN

Módulo: Sistema Distribuidos

Unidad 4: *Cloud computing*, algoritmos distribuidos, sistemas de archivos y bases de datos distribuidas

Escenario 7: *Cloud Computing* y paradigmas de la computación distribuida

Autor: Alexis Rojas

Asesor Pedagógico: Jeimy Lorena Romero Perilla

Diseñador Gráfico: Brandon Steven Ramírez Carrero

Asistente: Maria Elizabeth Avilán Forero