



Modalidad Abierta y a Distancia

Sistemas Distribuidos

Guía didáctica



Facultad de Ingenierías y Arquitectura

Departamento de Ciencias de la Computación y Electrónica

Sistemas Distribuidos

Guía didáctica

Carrera	PAO Nivel
▪ <i>Tecnologías de la Información</i>	VIII

Autor:

Torres Tandazo Rommel Vicente



D R B D _ 4 0 1 9

Asesoría virtual
www.utpl.edu.ec

Sistemas Distribuidos

Guía didáctica

Torres Tandazo Rommel Vicente

Universidad Técnica Particular de Loja



4.0, CC BY-NY-SA

Diagramación y diseño digital:

Ediloja Cía. Ltda.

Telefax: 593-7-2611418.

San Cayetano Alto s/n.

www.ediloja.com.ec

edilojainfo@ediloja.com.ec

Loja-Ecuador

ISBN digital - 978-9942-39-219-0



La versión digital ha sido acreditada bajo la licencia Creative Commons 4.0, CC BY-NY-SA: Reconocimiento-No comercial-Compartir igual; la cual permite: copiar, distribuir y comunicar públicamente la obra, mientras se reconozca la autoría original, no se utilice con fines comerciales y se permiten obras derivadas, siempre que mantenga la misma licencia al ser divulgada. <https://creativecommons.org/licenses/by-nc-sa/4.0/deed.es>

20 septiembre, 2021

Índice

1. Datos de información.....	8
1.1. Presentación de la asignatura	8
1.2. Competencias genéricas de la UTPL.....	8
1.3. Competencias específicas de la carrera	8
1.4. Problemática que aborda la asignatura	8
2. Metodología de aprendizaje.....	10
3. Orientaciones didácticas por resultados de aprendizaje.....	11
Primer bimestre	11
Resultado de aprendizaje 1	11
Contenidos, recursos y actividades de aprendizaje recomendadas	11
Semana 1	12
Unidad 1. Contexto de los sistemas distribuidos.....	12
1.1. Definición de sistema distribuido	12
1.2. Usos de los sistemas distribuidos.....	13
1.3. Tendencias en sistemas distribuidos	15
1.4. Medioambiente de los sistemas distribuidos	17
1.5. Compartir recursos	18
1.6. Retos de los sistemas distribuidos.....	19
1.7. Caso de estudio: La Web y DNS	24
Actividades de aprendizaje recomendadas	26
Autoevaluación 1.....	27
Semana 2	30
Unidad 2. Relación de los sistemas distribuidos y las redes de computadoras	30
2.1. Los sistemas distribuidos y las redes de computadoras.....	30
2.2. Historia de las redes de computadoras.....	30
2.3. Internet	34
2.4. Principios de Internet.....	37
Actividades de aprendizaje recomendadas	38
Semana 3	39
2.5. Frontera o borde de la red	39

2.6. Núcleo de la red	41
2.7. Abstracción de la red por capas o niveles.....	43
2.8. Proceso de encapsulación	44
2.9. Identificación de componentes.....	45
Actividades de aprendizaje recomendadas	46
Autoevaluación 2.....	47
Resultado de aprendizaje 2	50
Contenidos, recursos y actividades de aprendizaje recomendadas	50
Semana 4	51
Unidad 3. Programación de redes de computadoras (sockets).....	51
3.1. Implementación de la comunicación en redes de computadoras..	51
3.2. Protocolos	52
3.3. Capas de comunicaciones y el software	53
3.4. Interfaz de programación de aplicaciones	54
3.5. Interfaz de la capa de red	56
3.6. Comunicación entre procesos	58
Actividades de aprendizaje recomendadas	60
Semana 5	61
3.7. Sockets de red.....	61
3.8. Comunicación de datagramas con UDP	63
Actividades de aprendizaje recomendadas	69
Semana 6	70
3.9. Comunicación de segmentos con TCP.....	70
Actividades de aprendizaje recomendadas	76
Autoevaluación 3.....	78
Semana 7	81
Unidad 4. Representación de datos y mensajes multicast	81
4.1. Representación, transformación y codificación de datos	81
4.2. Comunicación multicast	83
Actividades de aprendizaje recomendadas	89
Autoevaluación 4.....	91
Actividades finales del bimestre	94

Semana 8	94
Segundo bimestre	95
Resultado de aprendizaje 3 y 4	95
Contenidos, recursos y actividades de aprendizaje recomendadas	95
 Unidad 5. Diseño y representación de sistemas distribuidos.....	96
5.1. Sistemas distribuidos y sistemas centralizados	96
5.2. Requerimientos de diseño para sistemas distribuidos	101
 Actividades de aprendizaje recomendadas	106
Semana 10	107
5.3. Tiempo y estado global en sistemas distribuidos	107
5.4. Tiempo global en sistemas distribuidos	108
5.5. Sincronización de relojes físicos	109
 Actividades de aprendizaje recomendadas	112
Semana 11	113
5.6. Sincronización a través de relojes lógicos	113
5.7. Estado global de un sistema distribuido	119
 Actividades de aprendizaje recomendadas	121
 Autoevaluación 5.....	123
 Unidad 6. Modelamiento de sistemas distribuidos	126
6.1. Configuración, definición y gestión de un sistema distribuido	126
6.2. Modelo físico.....	128
6.3. Modelo de arquitectura	130
 Actividades de aprendizaje recomendadas	138
Semana 13	139
6.4. Modelo fundamental.....	144
 Actividades de aprendizaje recomendadas	151
 Autoevaluación 6.....	153
Resultado de aprendizaje 5	156
 Contenidos, recursos y actividades de aprendizaje recomendadas	156
Semana 14	156

Unidad 7. Sincronización, coordinación y consenso	156
7.1. Contexto de la coordinación y el consenso.....	156
7.2. Algoritmos de coordinación.....	158
7.3. Algoritmos para elección de líder	161
Actividades de aprendizaje recomendadas	165
Semana 15	166
7.4. Algoritmos para exclusión mutua.....	166
7.5. Consenso.....	172
Actividades de aprendizaje recomendadas	180
Autoevaluación 7.....	182
Actividades finales del bimestre	185
Semana 16	185
4. Solucionario	186
5. Glosario.....	193
6. Referencias bibliográficas	194
7. Anexos	198



1. Datos de información

1.1. Presentación de la asignatura



1.2. Competencias genéricas de la UTPL

- Orientación a la investigación e innovación.

1.3. Competencias específicas de la carrera

- Implementar aplicaciones de baja, mediana y alta complejidad integrando diferentes herramientas y plataformas para dar solución a requerimientos de la organización.

1.4. Problemática que aborda la asignatura

La asignatura aborda los fundamentos teóricos, metodológicos y científicos de las tecnologías de información en los contextos empresariales permitiendo a los estudiantes diseñar soluciones para empresas de diferentes sectores que integren diversos elementos de

tecnología informática y computacional para resolver problemas de distinta naturaleza, estas propuestas involucrarán tecnologías como accesibilidad, ingeniería de software, gestión de proyectos, seguridad, además el uso de infraestructura de TI.

Está destinada a favorecer el desarrollo de soluciones para empresas integrando múltiples plataformas que les permitan ser más competitivas, complementario a ello, se investigan cuatro problemas relacionados: el primero es cómo fortalecer la acción de las pequeñas y medianas empresas Pymes en ramas estratégicas con el apoyo de las tecnologías de la información; cómo fomentar las inversiones en I+D, sobre todo en el sector empresarial, de los actores identificados, principalmente las áreas de la Banca; la poca incidencia en el desarrollo de la economía de parte de las empresas y escasa cooperación por falta medios y tecnologías que propicien su desarrollo; y, finalmente, la inclusión de todos los sectores de la sociedad en el uso de las tecnologías de la información y la comunicación.



2. Metodología de aprendizaje

Para lograr el aprendizaje de la asignatura se utilizará una combinación de las siguientes metodologías de aprendizaje:

1. Aprendizaje guiado a través del material didáctico, a saber: el plan docente de la asignatura, la guía didáctica, la tarea, la tutoría y la interacción a través de la plataforma EVA con el docente tutor.
2. Autoaprendizaje, ya que usted debe leer el contenido de la guía didáctica de la asignatura, subrayar los contenidos y utilizar su cuaderno de apuntes para tomar notas de los aspectos más relevantes, así como desarrollar las actividades recomendadas en la guía didáctica.
3. Aprendizaje basado en método de casos, a través del desarrollo de las actividades de aprendizaje del componente práctico-experimental de la asignatura que complementa la formación en la asignatura mediante la aplicación de los contenidos a un contexto definido.
4. La técnica de la pregunta, que se utilizará en las autoevaluaciones con el objetivo de identificar la comprensión sobre los fundamentos de la gestión de la calidad de software.



3. Orientaciones didácticas por resultados de aprendizaje



Primer bimestre

Resultado de aprendizaje 1

- Prepara un sistema de ordenador para usar en un servidor.

Contenidos, recursos y actividades de aprendizaje recomendadas

A través de este resultado de aprendizaje se podrá dar cuenta que los sistemas distribuidos son necesarios en la mayoría de las aplicaciones que existen actualmente en el mercado, además, podrá determinar la relación y evolución conjunta de las redes de computadoras y los sistemas distribuidos.

En esta unidad se hace un análisis de las redes de computadores y su relación con los sistemas distribuidos. Los conceptos de las redes de computadores han sido usados y mejorados por los sistemas distribuidos, además, muchos de los sistemas distribuidos actuales, para su funcionamiento, deben necesariamente usar la red de computadores más grande que existe, Internet.

Es importante que se pueda determinar cuál es la relevancia de los sistemas distribuidos, qué retos deben afrontar, cómo han evolucionado desde el nacimiento de Internet. Esta unidad hace un recorrido general de los conceptos y principales protocolos que son parte del estudio de los sistemas distribuidos.

En las actividades, nos permitimos proponer una lectura de cursos abiertos publicados por prestigiosas universidades del mundo, de tal forma que pueda reforzar el contenido y se pueda apropiar del mismo desde la realidad del lector.



Unidad 1. Contexto de los sistemas distribuidos

La presente unidad tiene como objetivo mostrar cuál es la definición de sistemas distribuidos, qué retos deben afrontar, cuál es su evolución y qué contexto se debe considerar para su implementación en un servidor o en una granja de servidores.

1.1. Definición de sistema distribuido

En esta asignatura vamos a ver las principales tecnologías, algoritmos que se requieren para generar y construir un sistema distribuido, por lo que es necesario definir un concepto:

Hemos preparado un recurso en donde, en forma interactiva, podrá entender el concepto de sistemas distribuidos.

[Concepto de sistemas distribuidos](#)

Además de la falta de acoplamiento, los sistemas distribuidos deben considerar tres aspectos importantes:

1. Concurrencia de componentes.
2. Falta de un reloj global.
3. Fallas autónomas de los procesos o elementos.

Un sistema distribuido debe soportar una alta **conurrencia** de sus componentes, un claro ejemplo de concurrencia es cuando los niños más pequeños en una escuela juegan al fútbol, todos van al único recurso, la pelota. En sistemas distribuidos la concurrencia de componentes es más compleja porque se debe saber qué componente debe tomar el recurso y garantizar su acceso, y, si lo modifica, este cambio debe ser consistente en todo el sistema distribuido. Es decir, que los cambios deben ser comunicados a todos los componentes y ellos deben acordar que tienen el estado más actual del recurso.

El sistema distribuido no tiene un **mecanismo global** que sincronice las acciones de todos sus componentes. Pueden hacer el siguiente ejercicio

en su entorno, a través de un grupo de mensajería electrónica (por ejemplo: WhatsApp) soliciten a todos que envíen su hora. Aunque todos los que participen tienen claro el concepto de tiempo algunos lo colocarán en su propio formato, otros en su propio huso horario, e inclusive los que están en la misma zona horaria no tendrán exactamente la misma hora. En un sistema distribuido con infinidad de transacciones, estas deben ser sincronizadas con una forma de estandarización de representación compensando también los tiempos de procesamiento y transferencia de información, y acoplando los relojes locales en cada dispositivo.

Otro ejemplo que muestra la necesidad de una estandarización de formato y el requerimiento de una sincronización global es la edición en línea de archivo de texto compartido donde los participantes pueden modificar sin un orden específico.

En un sistema distribuido, los componentes físicos, lógicos y los elementos de la red (enrutadores y enlaces) pueden fallar, los componentes físicos pueden sufrir desgaste o también pueden ser afectados por un elemento externo (fallo eléctrico), el software puede fallar por un mal diseño y los elementos de la red, los enlaces, se pueden averiar, los enrutadores se pueden congestionar, etc. Los sistemas distribuidos, de acuerdo con su naturaleza, deben considerar un cierto nivel de tolerancia a este tipo de fallos y pérdidas.

1.2. Usos de los sistemas distribuidos

Los sistemas distribuidos son utilizados en casi todos los ambientes: comercio, educación, salud, redes sociales, telefonía, videoconferencia, juegos en línea, logística y transporte, etc. Cada uno de estos ambientes tiene sus propios requerimientos, tolerancia a fallos, pérdidas y retardo.

En comercio existen aplicaciones como PayPal, sistemas de comercio como la bolsa de valores. En redes sociales, Facebook o ClubHouse. Juegos multijugador en línea como Minecraft o Fortnite, etc. En aplicaciones multimedia, por ejemplo: YouTube o Netflix. En salud, el monitoreo de pacientes. En educación existen los entornos virtuales de aprendizaje como Moodle, y aplicaciones de videoconferencia como zoom. En aplicaciones de logística y transporte existen aplicaciones como las utilizadas para la gestión ferroviaria, y el uso de geolocalización como Google maps.

Cuando hablamos de tolerancia a fallos nos referimos a que el sistema distribuido puede soportar que uno o algunos de sus componentes deje de funcionar, sin que esto impida que el sistema distribuido siga cumpliendo su tarea. El fallo de un porcentaje de los componentes del sistema afecta el rendimiento del sistema distribuido, sin embargo, el sistema distribuido puede seguir funcionando. Por ejemplo, considere un sistema de compartición de archivos que utilice una red malla (una conexión P2P), en la cual todos los dispositivos están conectados entre sí, y en el cual todos los nodos de la red tienen una copia del archivo compartido, cuando un nuevo nodo necesita el archivo, todos los nodos de la red envían partes del archivo. Mientras más nodos existan en la red, posiblemente el tiempo de transferencia será menor; cuando un porcentaje de nodos deja de funcionar, el sistema distribuido aún puede entregar el archivo con un tiempo de transferencia mayor.

Cuando se menciona la tolerancia a pérdidas, el sistema distribuido debe considerar que los elementos de la red subyacente pueden fallar, los enlaces y los enrutadores pueden dejar de funcionar, lo que ocasiona que la información enviada puede ser eliminada en tránsito o se puede retrasar. Es común que la tolerancia a fallos y la tolerancia a pérdidas se consideren como un solo tipo de tolerancia.

Cuando se habla de tolerancia a retardo, el sistema distribuido, dependiendo de su aplicación, debe considerar los tiempos de actualización de la información para obtener transacciones consistentes. Las aplicaciones distribuidas en la bolsa de valores, por ejemplo, son poco tolerantes al retardo. En estos sistemas, los valores de las acciones cambian rápida y dinámicamente, esta variación de precios se envía a los diferentes suscriptores en tiempo real para que puedan, de acuerdo a su mejor criterio, comprar o vender dichas acciones. La agilidad con la que se puede hacer la transacción comercial incide directamente en el éxito económico de la actividad. Estos sistemas distribuidos mantienen a cientos de suscriptores y alimentadores de información. Otra aplicación que es poco tolerante al retardo son los juegos en línea.

1.3. Tendencias en sistemas distribuidos

Las tendencias en sistemas distribuidos están relacionadas con el avance de la tecnología y de los algoritmos de comunicación, entre las tendencias podemos listar las siguientes:

1. El uso masivo de las redes de computadoras.
2. La movilidad de los nodos.
3. La ubicuidad de los nodos.
4. Los sistemas distribuidos para aplicaciones con tráfico interactivo.
5. La computación distribuida como utilidad.

El uso masivo de las redes de computadoras está dado por el aparecimiento de nuevas tecnologías, por ejemplo 5G. También por la miniaturización de hardware de los procesadores que son cada vez más eficientes, permitiendo equipos más pequeños y por lo tanto con una mejor gestión de energía.

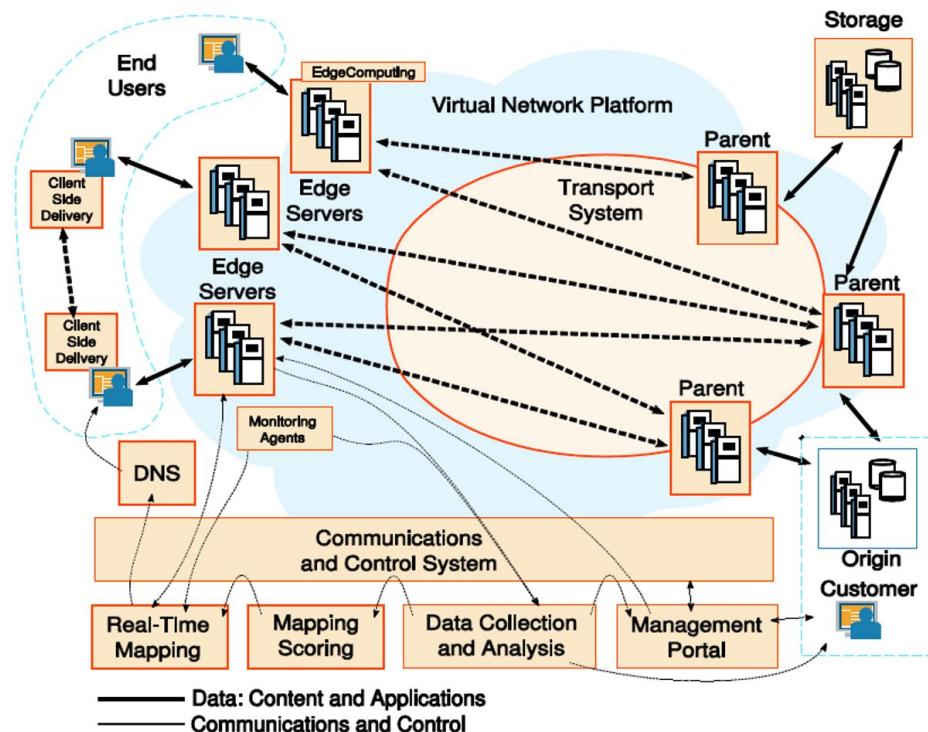
La movilidad y la ubicuidad de los nodos se da porque existe una mejora constante en la gestión de la energía en los equipos intermedios y terminales. Esto permite que tengamos la capacidad de tener comunicación en cualquier lugar y en movimiento. Aquí existe avances no sólo en el hardware, sino también en los protocolos de comunicación y el software que permiten que el nodo siga conectado a la red con estrategias como *roaming*.

Los sistemas distribuidos de multimedia permiten mejorar la calidad de experiencia de los usuarios que requieren de audio y video en forma síncrona o asíncrona. Los proveedores de contenido, por ejemplo YouTube, tienen servidores repartidos en todo el mundo que se replican con una geolocalización para hacer el acceso transparente al usuario. Utilizan estrategias que combinan el uso del servicio de DNS y la geolocalización de direcciones IP.

Akamai es una empresa de Internet que brinda distribución de contenido multimedia, ofrece escalabilidad masiva junto con la capacidad de ofrecer eventos multimedia, lanzamientos de juegos y actualizaciones de software a millones de usuarios al mismo tiempo. Coloca sus servicios en el límite entre la red de núcleo y la red de frontera.

La Figura 1 muestra la arquitectura de la plataforma de distribución de contenido multimedia Akamai, en la cual se puede ver la interacción del servicio de Akamai con los servidores de DNS y con los usuarios. Una granja de servidores que atiende a los usuarios cercanos e interactúan con los servidores de almacenamiento de los recursos creando una plataforma de red virtual.

Figura 1.
Plataforma Akamai



Nota. Tomado de Nygren et al. (2010).

Cuando hablamos de la computación distribuida como utilidad, involucra que todo el sistema distribuido ya está implementado y para usarlo se emplea una suscripción, generalmente acompañado del pago de algún valor. Para el usuario simplemente es un framework, como ejemplo tenemos AWS de Amazon.

Los sistemas distribuidos utilizan también conceptos de inteligencia artificial para mejorar su tiempo de respuesta y de servicio. Imagine un centro de datos donde un sistema distribuido puede hacer que sus

componentes o servidores distribuyan el trabajo inteligentemente para distribuir la carga de procesamiento equitativamente.

1.4. Medioambiente de los sistemas distribuidos

Los sistemas distribuidos deben observar las carencias y características de la tecnología en la cual se implementan, entre las cuales podemos mencionar:

1. Tecnologías subyacentes

Existe una gran cantidad de tecnologías y características subyacentes que un sistema distribuido debe considerar. Por ejemplo, el ancho de banda y los diferentes dispositivos que se conectan a la red. Para conectarnos a un Proveedor de Servicios de Internet (ISP) se puede hacer por diferentes equipos (teléfono, computador, enrutador) y por diferentes tecnologías (fibra óptica, wifi). Para ser más específico, la red interna de una empresa tiene los requerimientos de computación y de comunicación más altos que el de un hogar; y, por lo tanto, la tecnología de conexión y de cómputo en cada caso es diferente tanto en tecnología, capacidad y costo. Mientras que en el hogar, para conectarse a Internet, es necesario un solo equipo, en la empresa se requiere de tecnologías, procesos, personas trabajando en forma coordinada.

2. Apertura de la red

Es importante que un sistema distribuido permita que un nuevo equipo se conecte a la red. Cuando se generaron los primeros protocolos de comunicación no se determinó que los equipos que conectaban a la red pudieran ser teléfonos móviles con conexiones inalámbricas, que podían tener la capacidad de un computador, en ese entonces los equipos no se podían mover y se necesitaba una conexión física a través de un cable para enlazarse a la red.

3. Internet existente

Los sistemas distribuidos globales funcionan sobre Internet. Sin embargo, Internet fue creado como una estructura rígida. Esto hace que los sistemas distribuidos generen sus propias estrategias para que, utilizando la red subyacente, puedan lograr sus objetivos. Por ejemplo, Skype crea su

propia red usando el Internet. Cuando ingresamos a Skype un servidor de autenticación verifica el usuario y su presencia en el servicio. Para lograr una comunicación efectiva, Skype utiliza un concepto de supernodos que están geolocalizados de tal forma que, al iniciar una conexión, es muy común que el supernodo cercano atienda la petición y se comunique con el supernodo cercano al destino. La utilización de supernodos permite a Skype mejorar la experiencia de usuario determinando dinámicamente el estado de la red y aplicando cambios en demanda, por ejemplo, puede cambiar la codificación, perdiendo calidad, pero manteniendo la experiencia de usuario.

1.5. Compartir recursos

El objetivo principal de un sistema distribuido es el de compartir recursos en forma eficiente y eficaz. Eficiente porque el acceso a los recursos debe considerar el tiempo de entrega y la pérdida de información. Eficaz debido a que el recurso compartido no debe ser alterado, la información debe guardar integridad. En sistemas distribuidos el concepto de compartir recursos se logra con:

1. Definición de un servicio.
2. Definición de interfaces.
3. Invocación remota cliente-servidor.
4. Invocación remota orientada a objetos.

El sistema distribuido debe ser implementado con un objetivo que debe responder a un **servicio** o servicios con requisitos definidos por la aplicación y su utilización. Se puede pensar en un servicio como un procedimiento, una función o un objeto en software.

Los servicios definidos por el sistema distribuido deben contar con interfaces. Un servicio puede ser definido como una caja negra en la cual existe dos puntos de comunicación o interfaces, una de entrada y otra de salida. Las interfaces definen el tipo de dato, la representación. Las interfaces de entrada definirán los tipos de datos y su representación requerida, por ejemplo, un entero de 10 dígitos, etc. La interfaz de salida de igual forma define el formato del resultado luego de ser procesado por el servicio.

Una vez definido el servicio y sus interfaces, un programador puede implementarlos y acceder a ellos a través de dos formas, la primera a través de una **invocación remota** desde un cliente a un servidor en donde está el servicio. El cliente envía una petición con la información necesaria en la cantidad y formato definida en la interfaz de entrada y el servidor responde al cliente con la información acordada a través de la interfaz de salida. Una segunda forma, más actual y utilizada, de acceder y crear un servicio es a través del paradigma de la orientación a objetos con una **invocación remota orientada a objetos**, en esta última se utiliza una abstracción a objetos para representar variables, clases y sus comportamientos para la definición de las interfaces de entrada y de salida.

1.6. Retos de los sistemas distribuidos

Existen algunos retos que deben considerar los sistemas distribuidos. Vamos a revisar los que están entre los más importantes:

1. Heterogeneidad
2. Apertura o accesibilidad
3. Seguridad
4. Escalabilidad
5. Gestión de fallos
6. Concurrencia
7. Calidad de Servicio
8. Transparencia
 - a. Acceso: los recursos locales y remotos deben ser usados con operaciones idénticas.
 - b. Ubicación: los recursos pueden ser accedidos indiferente de su ubicación física y su ubicación en la red.
 - c. Concurrencia: los procesos pueden operar concurrentemente usando recursos compartidos sin interferir entre ellos.
 - d. Replicación: múltiples copias del recurso pueden ser usados para mejorar la confiabilidad y rendimiento sin conocimiento de los usuarios o programadores.

- e. Fallos: el sistema distribuido debe encubrir las fallas, permitiendo a los usuarios o programas completar sus tareas sin importar las fallas de hardware o software.
- f. Movilidad: permite mover los recursos y clientes dentro del sistema sin afectar la operación de los usuarios o programas.
- g. Rendimiento: permite al sistema ser reconfigurado para mejorar el rendimiento de acuerdo con la variación de la carga de trabajo.
- h. Escalamiento: permite al sistema y las aplicaciones expandirse sin cambiar la estructura del sistema o los algoritmos de aplicación.

Cuando hablamos de **heterogeneidad** nos referimos a que los sistemas distribuidos deben considerar la infinidad de tecnologías de redes y la disponibilidad de hardware existentes, las características de los diferentes sistemas operativos y la representación de datos en un computador. Por ejemplo, un entero se puede almacenar en memoria de una forma diferente dependiendo del hardware. También hay que considerar la infinidad de lenguajes de programación e inclusive en un mismo lenguaje de programación la implementación de un mismo algoritmo puede ser asignada a diferentes desarrolladores, cada cual con sus propias metodologías y técnicas de representación e implementación. Esto es importante porque cuando se comparte un recurso o información se debe asegurar que ese recurso compartido sea del mismo tipo de dato y la representación sea la misma que cuando se creó. Otro ejemplo es cuando se usa un código que es móvil, que se descarga al dispositivo de destino para ejecutarse, en este caso se debe asegurar que el hardware y el software anfitriones puedan ejecutar el código sin problemas.

Los sistemas distribuidos deben ser **accesibles**, se deben documentar las especificaciones de todas las interfaces, la documentación debe permitir que software de terceros pueda utilizar el sistema distribuido. Ahora mismo existen sistemas distribuidos propietarios, en los cuales su arquitectura es parte de sus activos de negocio, o, por otro lado, sus técnicas y algoritmos son patentados y por lo tanto no son accesibles debido a que el acceso a la información es restringido o tienen derechos de autor.

Los sistemas distribuidos son procesos cuyo objetivo es compartir algún recurso, lo hacen a través de una red que, de por sí, está expuesta a temas de **seguridad**. Los sistemas distribuidos deben tener la capacidad de cumplir con los requisitos principales de seguridad, confidencialidad, disponibilidad, autenticación, integridad y alcance. La **confidencialidad** se puede lograr con cifrado, la capacidad del sistema distribuido para que los mensajes que son transportados por la red no sean entendibles en el caso de que sean analizados por un tercero. Si la información no es alterada en la red, es decir, el mensaje que llega al receptor es el mismo que fue generado, esto es **integridad**. **Disponibilidad** es que el sistema distribuido pueda soportar un porcentaje de pérdidas y un límite en el retardo. La autenticación se refiere a que los componentes del sistema distribuido tengan la capacidad de determinar que el emisor y el receptor de los mensajes sean realmente las entidades que deben ser. El sistema distribuido debe tratar también de determinar el **alcance** de los componentes, un ejemplo, las cookies utilizan un código para ir recolectando información posiblemente de la identidad y las preferencias del usuario, sin embargo, este código solo debe tener acceso y obtener la información estrictamente relacionada y necesaria.

Otro de los retos que se debe considerar es la **escalabilidad**, no es lo mismo que el sistema distribuido atienda a 10 usuarios que a 10.000 usuarios. La escalabilidad tiene un costo económico y computacional, y es tarea de los administradores encontrar la relación costo-beneficio. Como una analogía, si un sistema distribuido tiene un servicio en un solo equipo servidor con una capacidad de atender peticiones, al agregar otro servidor no necesariamente va a atender peticiones. Esto es debido a que el sistema distribuido debe controlar la interacción entre esos dos servidores para intentar duplicar la respuesta a un mismo equipo. Si se agrega más servidores al sistema distribuido la complejidad aumenta e inclusive se debe buscar una estabilidad en el servicio. Se debe considerar en este caso平衡adores de carga o peticiones. Además, ahora el sistema distribuido debe implementar estrategias para hacer un consenso, para controlar las eventuales fallas, definir qué componente tendrá más relevancia en el sistema y liderará algunas acciones, este debe, por lo tanto, tener un proceso de elección de líder. Los conceptos y estrategias de consenso, elección de líder y detección de fallos, son parte importante de los sistemas distribuidos y se los verá en detalle en los apartados posteriores.

Un sistema distribuido puede dejar de funcionar adecuadamente luego de que se ha alcanzado un porcentaje de fallas. Cuando un componente del sistema distribuido falla, el sistema distribuido debe seguir trabajando. Por lo tanto, debe implementar procesos de detección y recuperación de fallas. Sin embargo, las fallas pueden ser eventuales y afectar directamente el rendimiento del sistema, entre las estrategias que el sistema distribuido puede implementar están el envío de mensajes entre componentes para determinar si están activos, implementar temporizadores para sincronizar los componentes y sus acciones. También debe considerar aquellos procesos necesarios para agregar nuevos componentes o reingresar componentes fallidos.

La **conurrencia** involucra que el sistema distribuido tiene que verificar que dos acciones no pueden cambiar el mismo recurso compartido al mismo tiempo. Por ejemplo, imaginen que existen dos acciones que están ingresando información a una cuenta bancaria electrónica con saldo en cero, una acción trata de hacer un retiro de 10 USD y la otra un depósito de 20 USD, inclusive aquí el sistema distribuido debe garantizar un orden específico para las acciones de tal forma que no existan errores.

Otro de los retos que debe afrontar un sistema distribuido es la **calidad del servicio**, cuando hablamos de calidad de servicio se debe diferenciar de calidad de experiencia. La calidad de servicio se refiere a una medida cuantitativa de los atributos propios del servicio. La calidad de experiencia es una medida cualitativa del servicio. Por ejemplo, cuando un usuario hace un reclamo a un ISP por su conexión a Internet, lo hace basado en su calidad de experiencia, en otras palabras, la calidad esperada no es la que el usuario subjetivamente espera. Se mide con escalas cualitativas, bueno, malo, etc. El ISP por su parte le da información cuantitativa de la conexión, por ejemplo, retardo, ancho de banda, y compara con el contrato de servicio que tiene con el usuario. Si la información está de acuerdo con el contrato, entonces el ISP tiene argumento para desestimar el reclamo del usuario a pesar de que no tiene la calidad de experiencia esperada. En sistemas distribuidos, la calidad de servicio debe ser llevada de manera similar, dependiendo de la naturaleza los requisitos de calidad de servicio cambiarán, por ejemplo, la calidad de servicio de un sistema multijugador en línea es diferente a la calidad de servicio para un sistema de mercado de valores.

El concepto de **transparencia** en un sistema distribuido tiene relación con algunos conceptos. Debe ser abierto, debe permitir que si accede a recursos en otro computador sea transparente como si el recurso estuviera en el mismo computador, usando las mismas operaciones que usa con recursos locales. En un sistema distribuido debe ser transparente la ubicación real del recurso, inclusive puede haber copias del recurso en todo el sistema distribuido, en este caso se debe garantizar que todas las copias sean las mismas y que los cambios en las mismas se repliquen en todo el sistema.

En un sistema distribuido, al usuario no le interesa conocer si existen muchos servicios localizados en diferentes servidores intentando acceder al mismo recurso. El sistema debe manejar esta concurrencia, debe garantizar el acceso atómico o exclusivo si hay cambios en el recurso. Si el recurso solo es accedido para obtener información, el sistema distribuido puede permitir que muchos servicios puedan acceder al recurso al mismo tiempo.

El sistema distribuido debe permitir, en forma transparente, la replicación con el ánimo de mejorar el rendimiento y disponibilidad de la red. Un claro ejemplo aquí es el servicio de YouTube, el cual mantiene copias de sus archivos geoposicionadas en servidores distribuidos en todo el mundo. De tal forma que, si una persona accede a un video, el sistema distribuido de YouTube redirecciona su petición al servidor de contenido más cercano al usuario mejorando los tiempos de respuesta, la calidad y fluidez del video.

La transparencia con respecto a las fallas le permite al sistema distribuido superar la falla de uno o algunos de sus componentes sin conocimiento del usuario. Sin embargo, el sistema distribuido debe manejar las fallas adecuadamente para garantizar que el recurso compartido no se dañe o se corrompa.

El sistema distribuido debe tener la capacidad de que sus componentes se puedan mover transparentemente sin alterar el rendimiento, por ejemplo, se puede cambiar la configuración o la cantidad de servidores sin que el cliente tenga conocimiento del cambio.

Un sistema distribuido debe escalar en forma transparente. Este es uno de los retos más fuertes que debe afrontar cuando se diseña y se programa un sistema distribuido. Un sistema distribuido debe tener la capacidad de ampliarse simplemente agregando nuevos componentes, sin cambiar su

estructura de base. Por ejemplo, un sistema distribuido que es escalable es el servicio web. Se pueden agregar más servidores, pero los protocolos como http y la estructura del servicio no cambian.

1.7. Caso de estudio: La Web y DNS

La Web y el servicio de nombres de dominio (DNS) son algunos de los primeros sistemas distribuidos, los dos afrontan en forma eficiente con la mayoría de los retos de los sistemas distribuidos.

El servicio web, por ejemplo, puede soportar diferentes tecnologías, hardware y software – heterogeneidad, además los protocolos que lo conforman están publicados en los Request for Comments (RFC) y son accesibles por cualquier lector – accesibilidad. Uno de los principales problemas cuando inició el servicio web fue la seguridad, actualmente para lograr acercarse a parámetros de seguridad debe apoyarse en aplicaciones externas, por ejemplo el uso de SSL (secure socket layer). El servicio web es altamente escalable, se pueden agregar miles de servidores y el servicio sigue funcionando.

En la Tabla 1, se hace un análisis cualitativo de cada uno de los tipos de transparencia que puede tener un servicio web, se puede ver las respectivas observaciones.

Tabla 1.
Transparencia del servicio web

Transparencia	Nivel (Alto, Medio, Bajo)	Observaciones
Acceso: los recursos locales y remotos deben ser usados con operaciones idénticas.	Alto	Cualquier cliente utiliza siempre las mismas operaciones de http, GET, POST, etc.
Ubicación: los recursos pueden ser accedidos indiferente de su ubicación física y su ubicación en la red.	Alto	Cuando se accede a un sitio web, los elementos del sitio pueden estar en cualquier servidor.
Concurrencia: los procesos pueden operar concurrentemente usando recursos compartidos sin interferir entre ellos.	Alto	La misma página puede ser accedida por diferentes usuarios a la vez.

Transparencia	Nivel (Alto, Medio, Bajo)	Observaciones
Replicación: múltiples copias del recurso pueden ser usados para mejorar la confiabilidad y rendimiento sin conocimiento de los usuarios o programadores.	Alto	Puede existir el mismo recurso en diferentes servidores web.
Fallos: el sistema distribuido debe encubrir las fallas, permitiendo a los usuarios o programas completar sus tareas sin importar las fallas de hardware o software.	Alto	Cuando se trabaja en conjunto con DNS, la misma página puede estar ubicada en diferentes servidores, de tal forma que, si uno falla, el sistema sigue funcionando.
Movilidad: permite mover los recursos y clientes dentro del sistema sin afectar la operación de los usuarios o programas.	Alto	Igual que en el ítem anterior, los recursos se pueden mover entre servidores web.
Rendimiento: permite al sistema ser reconfigurado para mejorar el rendimiento de acuerdo con la variación de la carga de trabajo.	Alto	Se pueden agregar servidores de acuerdo con la demanda o cantidad de peticiones de los clientes.
Escalamiento: permite al sistema y las aplicaciones expandirse sin cambiar la estructura del sistema o los algoritmos de aplicación.	Alto	Para el usuario final es transparente que la cantidad de servidores que hay para un servicio web.

Nota. Elaborado por el autor

El servicio de DNS también es un sistema distribuido, uno de los más importantes para el funcionamiento de Internet. Afronta de manera similar que un servicio web los retos que debe afrontar un sistema distribuido. Se invita al lector a hacer el análisis cualitativo de los tipos de transparencia que soporta el servicio de DNS.

Recursos:

En el REA Curso Abierto de Sistemas Distribuidos de la Universidad Carlos III de Madrid (López Fuentes, 2015), específicamente en el tema 1 ([Tema 1 - Introducción a los Sistemas Distribuidos – \(uc3m.es\)](#)), existe un enfoque similar de esta primera unidad, invito a hacer un cuadro comparativo de lo analizado en esta unidad de tal forma que se amplíen sus conocimientos.



Actividades de aprendizaje recomendadas

Es importante que en esta semana usted realice las siguientes actividades para consolidar su aprendizaje:

- Leer y familiarizarse con el material didáctico de la asignatura.
 - Lea nuevamente los contenidos de esta guía y cree un mapa conceptual.
 - Consulte el plan docente donde se muestra la secuencia de estudio de los contenidos, así como las actividades que complementan el aprendizaje organizadas por fechas, con su respectiva valoración, rúbricas de evaluación y distribución de las calificaciones por bimestre.
 - El entorno virtual de aprendizaje es el medio de interacción permanente entre docente y estudiante, donde encontrará orientaciones semanales, recursos, herramientas de interacción con sus compañeros y docente.
- Ingrese al entorno virtual de aprendizaje para conocer las orientaciones de estudio y actividades a desarrollar esta semana, así como para realizar consultas a su docente tutor e interactuar con sus compañeros.
- Lea los conceptos presentados en la primera semana, complemente los conceptos que allí se entregan con los expuestos en los recursos, con estos insumos elabore su propio resumen.
- Haga un análisis cualitativo similar al presentado en el texto de los tipos de transparencia que soporta el servicio de DNS.

Nota: conteste las actividades en un cuaderno de apuntes o en un documento Word.

A continuación, se invita a que revise como van sus conocimientos haciendo la respectiva autoevaluación.



Autoevaluación 1

En los siguientes enunciados seleccione la alternativa correcta:

1. Los componentes de un sistema distribuido están ubicados:

- a. En el mismo computador.
- b. En máquinas virtuales dentro del mismo equipo.
- c. En una red de computadoras.
- d. Están ubicados en los puntos de acceso de una red.

2. Los componentes en un sistema distribuido pueden ser

- a. Memoria.
- b. Procesador.
- c. Objetos.
- d. Interfaces de funciones.

3. ¿Cuándo los componentes en un sistema distribuidos coordinan sus acciones, que debe analizarse?

- a. La capacidad de procesamiento de los nodos.
- b. El nivel de acoplamiento de los nodos en tiempo y espacio.
- c. La capacidad de memoria de los nodos.
- d. La velocidad de los dispositivos de almacenamiento secundario.

4. ¿Por qué es necesario un reloj global en un sistema distribuido?

- a. Para sincronizar los procesos locales en cada componente.
- b. Para elegir un proceso líder en el sistema distribuido.
- c. Para mejorar las operaciones de lectura y escritura en la memoria primaria.
- d. Porque los componentes no tienen un conocimiento directo de los relojes del resto.

- 5. ¿Cuál de las siguientes aplicaciones es un ejemplo de un sistema distribuido para un uso comercial?**
- a. PayPal.
 - b. Netflix.
 - c. Moodle.
 - d. Google Maps.
- 6. ¿Cuál de los siguientes sistemas distribuidos es tolerante a pérdidas?**
- a. Sistema bancario.
 - b. YouTube.
 - c. Bolsa de valores.
 - d. Juego en línea.
- 7. Seleccione las opciones que considere correctas:**
- ¿Qué estrategias usan las redes de distribución de contenido como Akamai para su sistema distribuido?
- a. Geolocalización de servidores
 - b. Servidores replicados.
 - c. Procesamiento replicado.
 - d. Integración de tecnología de red.
- 8. En las siguientes interrogantes seleccione la alternativa correcta:**
- ¿Una interfaz de entrada qué debe tener definido?
- a. La cantidad de argumentos y el tipo de dato de cada uno.
 - b. El nivel de acoplamiento entre los servicios.
 - c. La cantidad de memoria requerida para el procesamiento.
 - d. El tipo de datos del resultado del servicio.
- 9. ¿Qué tipo de transparencia tenemos cuando mencionamos que el sistema distribuido puede expandirse?**
- a. Escalamiento.
 - b. Movilidad.
 - c. Rendimiento.
 - d. Concurrencia.

10. ¿Cuáles son los requisitos de seguridad de un sistema distribuido?

- a. Escalabilidad y alcance.
- b. Confidencialidad, integridad.
- c. Conurrencia y autenticación.
- d. Disponibilidad y procesamiento.

[Ir al solucionario](#)



Unidad 2. Relación de los sistemas distribuidos y las redes de computadoras

Los sistemas distribuidos han ido evolucionando en el mismo lapso que las redes de computadoras. Inclusive hay sistemas distribuidos que se implementaron justo con Internet, por ejemplo, la *world wide web (WWW)*. Sin embargo, los requisitos actuales de velocidad de transmisión, movilidad y ubicuidad requeridos por los sistemas distribuidos no fueron anticipados cuando se generaron los estándares para las redes de computadores e Internet. Los sistemas distribuidos deben implementar estrategias para suplir o enmascarar las limitaciones de las redes de computadores.

2.1. Los sistemas distribuidos y las redes de computadoras

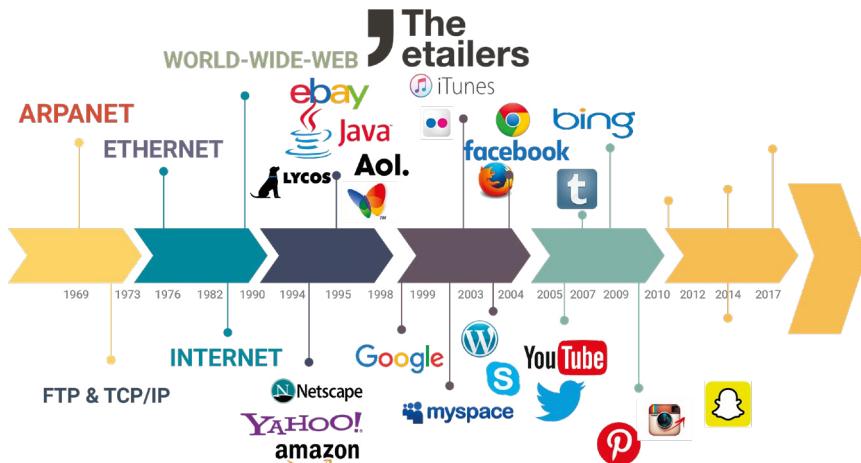
En este apartado vamos a hacer un repaso de los conceptos más relacionados entre las redes de computadores y el diseño e implementación de los sistemas distribuidos. Se hará un recorrido de las estrategias y componentes de las redes de computadoras para identificar dónde se acoplan los conceptos y metodologías de un sistema distribuido. El objetivo es determinar dónde y cómo se desarrollan las interfaces, cómo se puede abstraer una interfaz, cómo un sistema distribuido puede usar una infraestructura subyacente poco fiable como Internet y colocar sobre esta red estrategias relacionadas con los sistemas distribuidos como calidad del servicio, coordinación de procesos, consenso y detección de fallos.

2.2. Historia de las redes de computadoras

Como un poco de historia, cuando se ideó e implementó Internet, no fue creado para soportar los diferentes y servicios que ahora mismo existen en la red. Por ejemplo, el tamaño de los primeros computadores era similar al de una habitación, sin embargo, ahora en nuestra mano cabe un dispositivo que tiene muchísima más capacidad de procesamiento y almacenamiento, e inclusive tienen la capacidad de movimiento. La Figura 2 muestra una línea de tiempo de cómo los servicios de Internet han ido apareciendo,

como se puede ver WWW es uno de los primeros sistemas distribuidos en aparecer en la década de 1990.

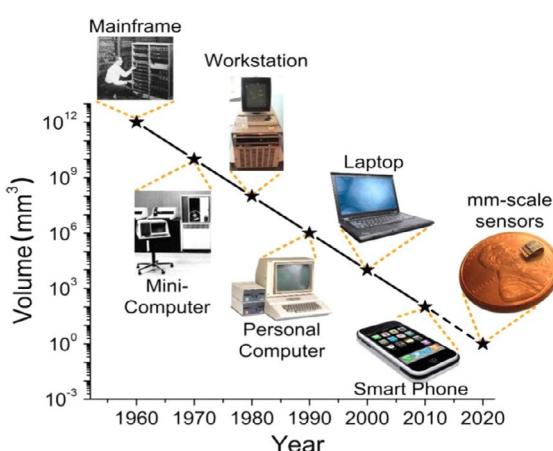
Figura 2.
Evolución de Internet



Nota. Tomado de Ipv6summit (2021).

La Figura 3 muestra, a su vez, cuál ha sido el avance del hardware y la tecnología para la conectividad y luego para las redes de computadores, desde los primeros teléfonos análogos hasta actualmente los dispositivos móviles. Los sistemas distribuidos tuvieron su inicio relativamente en el mismo período en que aparecieron los computadores personales.

Figura 3.
Evolución de los computadores



Nota. Tomado de Spicer (2015).

Los primeros computadores, por su tamaño, eran estáticos y trabajaban de manera autónoma, por lo tanto, no existía la probabilidad de que sean vulnerados o hackeados. En ese periodo de construcción de Internet, los protocolos fueron diseñados sin considerar la movilidad, la conectividad, la seguridad y, más aún, no se consideraban los requerimientos de los sistemas distribuidos. En la Figura 4 se muestra la primera computadora comercial en ser vendida en el mundo, la computadora Z4 de la empresa Zone KG en el año 1950.

Figura 4.
Computadora Z4

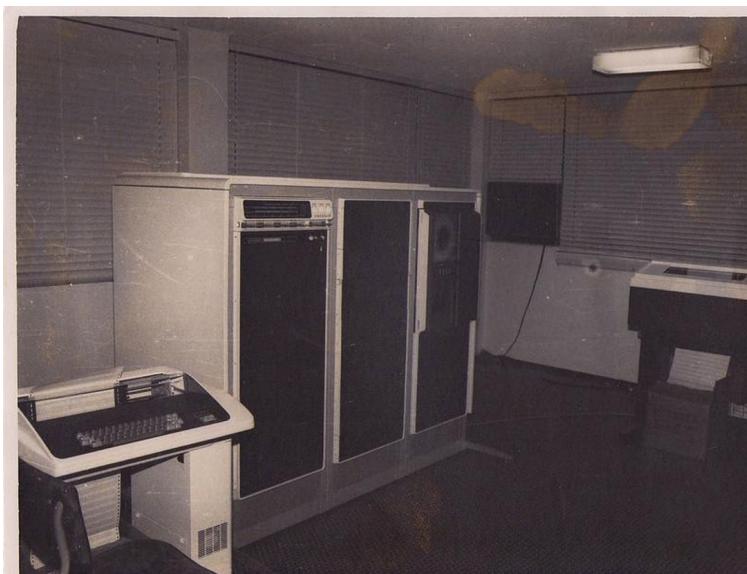


Nota. Tomado de *Wikipedia* (2021).

En un contexto más local, la Universidad Técnica Particular de Loja en Ecuador, tuvo uno de los primeros computadores en la ciudad de Loja y en el país en el año de 1979 cuando inauguró su primer centro de cómputo (UTPL, 1979a). En la Figura 5 se puede observar el espacio que ocupaba el primer computador y en la Figura 6 están partes del mismo computador, ahora en exhibición en el edificio del Departamento de Ciencias de la Computación.

Figura 5.

Primer computador en la UTPL (UTPL)



Nota. Tomado de UTPL (1979b).

Figura 6.

Primer computador de UTPL en exhibición 2021



Nota. Elaborado por el autor

La evolución de Internet tiene hitos importantes empezando en la década de 1960, en la cual se mostraron los principios de la conmutación de paquetes y las teorías de manejo de colas. En esta década ARPANET es desarrollada y en 1972 se envía el primer correo electrónico.

En la década de 1970 asoman redes propietarias, como SNA, y el concepto de comunicar varias redes entre sí toma el nombre de interconexión entre redes. Para 1979, ARPANET tiene cerca de 200 nodos conectados.

En la década de 1980 nuevos protocolos son desarrollados, se acoge como estándar el modelo de referencia TCP/IP. Son propuestos e implementados los protocolos para DNS, SMTP, FTP y TCP. En Norteamérica se crean redes de cobertura nacional y se logran interconectar más de 100 mil nodos. DNS ya es un protocolo distribuido que permite usar nombres nemotécnicos en vez de direcciones IP logrando abstracción de localización, ya que no es necesario conocer la dirección del recurso, solo se requiere recordar el nombre asociado.

En la década de 1990 empieza la comercialización de la red, aparece la WWW y se crean nuevas aplicaciones como mensajería instantánea, aplicaciones P2P para compartir archivos y redes sociales. Al final de esta década, los enlaces de comunicación tienen velocidades de transferencia en el rango de Gbps. Con la WWW y sus protocolos HTML y HTTP se impulsa y se fortalece lo que ahora es Internet.

Desde el 2000 a la fecha, la cantidad de usuarios y aplicaciones en red ha crecido exponencialmente, consolidando la necesidad de contar con sistemas distribuidos en los cuales el procesamiento y la compartición de recursos son obligatoriamente requeridos.

2.3. Internet

El Internet es la red de redes, permite la interconexión entre diferentes tipos de tecnologías desarrolladas por diferentes organizaciones.

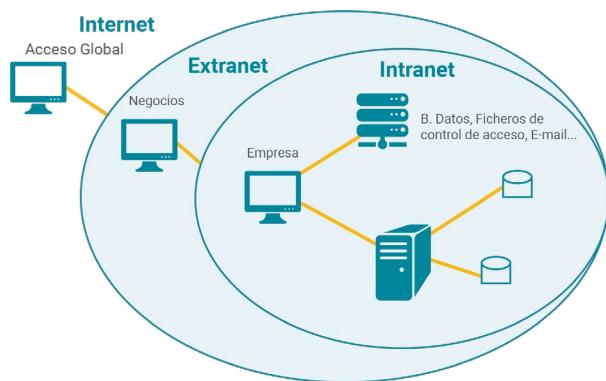
Continuemos con el aprendizaje mediante la revisión de este importante tema.

Internet, por lo tanto, es un tema de análisis interesante porque no hay un control o una gestión central, cada organización gestiona, por ejemplo, la

cantidad de ancho de banda que puede tener hacia el Internet a través de un ISP.

Al interno de la organización, en su red, también pueden coexistir diferentes tecnologías de comunicación, hardware y software, como se puede ver en la Figura 7.

Figura 7.
Internet e Intranet



Nota. Tomado de Priano et al. (2007).

La interconexión entre redes es la comunicación de redes de diferente tamaño, de diferentes operadores y/o propietarios, de diferentes tipos de tecnologías subyacentes, donde cada una tiene su propio nivel de direccionamiento a través de una comunicación transparente con una vista unificada de toda la red.

Vista general de los componentes de Internet

Desde una vista amplia podemos decir que Internet brinda un servicio a los dispositivos finales, para hacerlo necesita el uso e interacción de enlaces de comunicación, enrutadores, protocolos y estándares.

1. **Dispositivos finales**, aquellos que son el origen y el destino de la información, generan y consumen información, equipos que están en contacto directo con el usuario o la aplicación. Por ejemplo: dispositivos móviles, impresoras, computadores. Los dispositivos finales no necesariamente son parte de Internet.

2. **Enlaces de comunicación**, aquellos que unen las organizaciones con los ISP y con el Internet. Por ejemplo: fibra óptica y enlaces satelitales.
3. **Enrutadores**, son equipos de propósito específico, ubicados entre los dispositivos finales, tratan de enviar la información lo más rápidamente posible entre los dispositivos finales. Tienen poca capacidad de discernir qué tipo de tráfico está pasando por la red.
4. **Protocolos**, definen la forma y la secuencia de cómo los dispositivos finales intercambian información.
5. **Estándares**, los protocolos y la tecnología para los enlaces de comunicación son definidos de forma transparente, de tal forma que cualquier programador o creador de tecnología implemente sus soluciones de manera que pueda interactuar con dispositivos de otro proveedor. Los estándares para Internet están definidos y son de acceso público a través de los RFC (*Request for comments*) que son creados a través de la IETF (*Internet Engineering Task Force*).

Internet desde el punto de vista de uso para los sistemas distribuidos

Internet, desde la vista del uso, es un acercamiento a la infraestructura para la implementación de aplicaciones y un proveedor de servicios de comunicación no orientado a la conexión.

Como infraestructura, Internet habilita la interacción para los sistemas distribuidos, tales como DNS, correo electrónico, WWW, etc.

Internet utiliza un paradigma no orientado a la conexión, que significa que no hay un establecimiento de conexión previo que permita acordar y negociar parámetros en la comunicación. Por lo tanto, los enrutadores pueden enviar la información desordenada, cada paquete perteneciente a un mismo flujo puede tomar una ruta diferente al destino. Si el enrutador está saturado, simplemente elimina los paquetes en tránsito. Tampoco define estrategias para la cadencia de arribo de los paquetes, requisito necesario para el tráfico multimedia en vivo.

La responsabilidad de tener una comunicación fiable está a cargo de las aplicaciones que por lo general se ejecutan en los dispositivos finales.

2.4. Principios de Internet

Cuando hablamos de los principios de comunicación se debe considerar los siguientes:

1. Computación en la frontera y en el núcleo.
2. Paradigmas de comunicación.
3. Abordado a través de niveles o capas.
4. Red de redes colaborativas.

La inteligencia en Internet se lleva en los equipos finales, donde se genera la información y se consume. La red simplemente toma la información generada en los equipos emisores y trata de hacerlos llegar, lo más rápidamente posible, al destino. Por lo general, los equipos finales están en la frontera de la red y los enruteadores están en el núcleo de la red. En el siguiente recurso se puede ver el límite entre la red de núcleo y la frontera de la red.

Distribución vertical de las redes de computadores

En el recurso se muestra claramente qué equipos pertenecen al núcleo o a la frontera de la red.

En redes de computadoras hay cuatro diferentes paradigmas de comunicación: orientado a la conexión, no orientado a la conexión, orientado a paquetes y commutación de circuitos.

Una red orientada a la conexión es similar a cuando se habla por teléfono, en este caso se debe configurar la comunicación con el receptor. Por su parte, el caso de una red no orientada a la conexión puede asemejarse al servicio postal cuando se envía una carta, ya que el receptor no conoce de su existencia hasta que llega. Orientado a la conexión involucra que hay una negociación de la velocidad de transferencia, del formato de la comunicación y de algunas configuraciones adicionales. Cuando la comunicación es no orientada a la conexión, simplemente no hay una retroalimentación por parte del receptor al emisor y no se puede conocer si la información llegó o si se perdió en el camino.

Los otros dos paradigmas están relacionados con el uso de la infraestructura, orientado a paquetes y commutado por circuitos. Orientado a paquetes significa que se puede tomar la infraestructura de Internet y

los paquetes de un mismo flujo o comunicación pueden viajar a través de diferentes enlaces indistintamente. Comutado por circuitos involucra un uso exclusivo de toda la infraestructura para el flujo de comunicación. Entre el emisor y el receptor se genera un camino directo que se mantiene mientras dura la comunicación, indiferente de si se ocupa o no toda la infraestructura.

Para reducir la complejidad, el trabajo de las redes de computadoras e Internet se definió a través de capas, el modelo que usa Internet es TCP/IP el cual tiene cuatro capas que serán analizadas más adelante.

Internet finalmente permite que redes de diferente tamaño, tecnología, software puedan intercambiar información en forma colaborativa.

Recursos:

En el REA Curso Abierto de Sistemas Distribuidos de la Universidad Carlos III de Madrid (López Fuentes, 2015), específicamente en el tema 2 ([Tema 2 - Redes e interconexión – \(uc3m.es\)](#)) existe un enfoque similar de esta unidad, invito a hacer un cuadro comparativo de lo analizado en este apartado de tal forma que se amplíen sus conocimientos.



Actividades de aprendizaje recomendadas

Es importante que en esta semana usted realice las siguientes actividades para consolidar su aprendizaje:

- Leer y familiarizarse con el material didáctico de la asignatura.
 - Lea nuevamente los contenidos de esta guía y cree un mapa conceptual de toda la unidad.
 - Consulte el plan docente donde se muestra la secuencia de estudio de los contenidos, así como las actividades organizadas por fechas que complementan el aprendizaje, con su respectiva valoración, rúbricas de evaluación y distribución de las calificaciones por bimestre.
 - Recuerde que el entorno virtual de aprendizaje es el medio de interacción permanente entre docente y estudiante, donde

encontrará orientaciones semanales, recursos, herramientas de interacción para con sus compañeros y docente.

- Ingrese al entorno virtual de aprendizaje para conocer las orientaciones de estudio y actividades a desarrollar de esta semana; así como para realizar consultas a su docente tutor e interactuar con sus compañeros.
- Lea los conceptos presentados en la segunda semana, complemente los conceptos que allí se presentan con los cursos abiertos de sistemas distribuidos, con estos insumos elabore su propio resumen.

Nota: conteste las actividades en un cuaderno de apuntes o en un documento Word.



Semana 3

2.5. Frontera o borde de la red

En la frontera de la red están los equipos finales, donde corren las aplicaciones, se implementa la fiabilidad de la red y por lo general es donde están los clientes y consumidores de un sistema distribuido.

Los equipos finales pueden usar dos tipos de comunicación cliente/servidor y P2P. En cliente/servidor, un cliente requiere un servicio, por ejemplo, ubicado en un servidor web. En el modelo P2P no existe una asignación específica de funciones, cualquier equipo puede ser cliente o servidor.

En la frontera de la red se pueden dar los dos tipos de conexiones: orientado y no orientado a la conexión. La selección del tipo de conexión está relacionada directamente con las características, la tolerancia al retardo y la pérdida que necesita garantizar la aplicación.

En Internet, un servicio orientado a la conexión se logra a través del protocolo TCP cuya meta es transferir datos entre sistemas finales en forma fiable. Para lograrlo se hace una negociación inicial denominada: saludo de tres vías. TCP también da confiabilidad a la red, eso significa que si un paquete o segmento se pierde en el camino se definen los

mecanismos y estrategias de recuperarlo, como por ejemplo, colocar un identificador del segmento para saber que ha llegado al receptor. De su parte, el receptor al determinar un segmento faltante informa al emisor indicando que el segmento con el identificador no ha arribado.

En una transmisión por el dinamismo de la red, los paquetes pueden llegar en diferente orden. Por ejemplo, imagine que está transfiriendo un video en vivo, si llegan los paquetes en desorden no es posible que tengamos visualización fluida o continua. En este caso, TCP antes de entregar información a la aplicación almacena los paquetes en una memoria temporal en el receptor de tal forma que pueda ordenarlos y luego pasárselos a la aplicación. De la misma manera, en el emisor existe una memoria temporal en la que se almacena los paquetes que no han sido confirmados por el receptor.

TCP también trabaja con dos conceptos importantes, el control de congestión y control de flujo. El control de flujo se hace en la memoria del receptor controlando que no se llene la memoria temporal, el receptor notifica al emisor el estado de uso de memoria de tal forma que el emisor pueda limitar la cantidad de información que envía. Como vemos, el control de flujo se da en los equipos finales.

La congestión, en cambio, se da en los equipos intermedios o encaminadores. Los mismos que pueden, a su vez, tener comprometidos sus recursos (memoria, procesamiento, etc.) lo que afecta directamente en la velocidad en la cual los paquetes puedan despacharse e inclusive pueden hacer que se pierdan o se eliminan. Para el control de congestión, TCP utiliza temporizadores y acuses de recibo de paquetes. Cuando el emisor despacha un paquete inicia un temporizador esperando un acuse de recibo por parte del receptor, si no es recibido en el tiempo definido en el temporizador, el emisor retransmite nuevamente el mensaje.

Los equipos finales ubicados en la frontera de la red también pueden usar un servicio no orientado a la conexión. Este tipo de servicio no cuenta con ningún control de congestión ni brinda una fiabilidad de la red, no le interesa si se pierden los paquetes o si se satura el receptor. En un servicio no orientado a la conexión los mecanismos de fiabilidad no se implementan, lo que hace que el procesamiento de la información sea más rápido. Por lo general, este tipo de servicio se utiliza en aplicaciones tolerantes a la pérdida y no tolerantes al retardo, como por ejemplo videoconferencias.

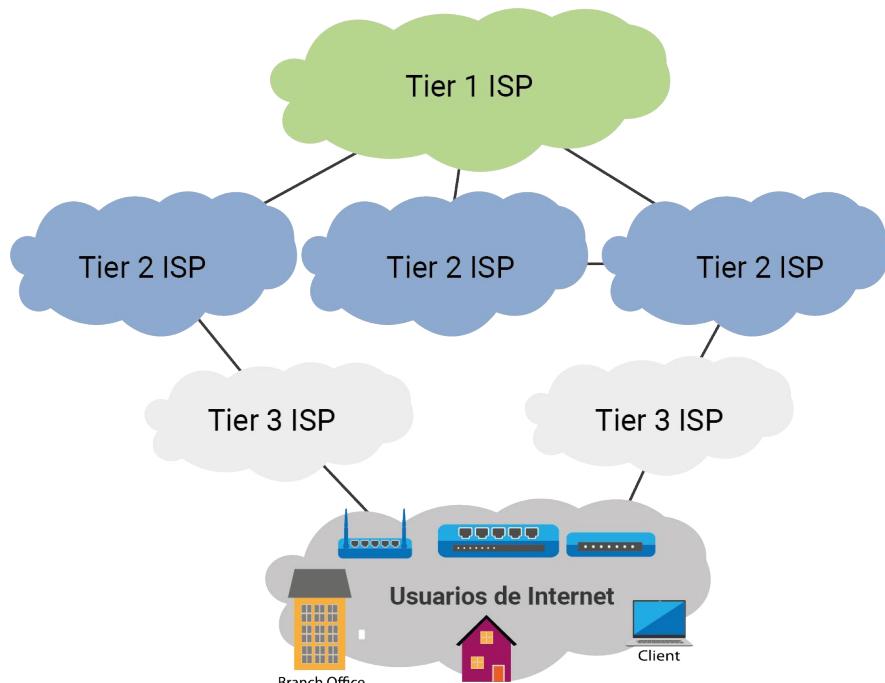
2.6. Núcleo de la red

Una vez que la información sale del receptor a través de su interfaz de red, y si el destino de la información está en otra red, empieza el trabajo del núcleo de la red. Está conformado por un conjunto de nodos y enlaces, los nodos son denominados enrutadores y los enlaces son las diferentes tecnologías que unen los enrutadores.

Ahora vamos a conocer la operatividad de los proveedores de servicios de internet (ISP):

Para tener acceso a Internet, las entidades u organizaciones utilizan un Proveedor de Servicios de Internet (ISP), los ISP se interconectan entre ellos formando el backbone de Internet, Figura 8. El núcleo de la red, desde el punto de vista de la organización, empieza desde la interfaz de salida de su enrutador de borde, el enrutador que conecta la entidad con su ISP. Los ISP por lo general dan el servicio a más de una entidad u organización.

Figura 8.
Jerarquía de ISP



Nota. Tomado de Govoni (2012).

En el núcleo de la red puede existir más de un camino desde un origen a un destino y cada paquete de un mismo flujo puede transitar por un camino diferente.

El trabajo de los enrutadores es despachar los paquetes entrantes lo más rápidamente posible, de acuerdo con las condiciones de la red. Considera cada paquete en forma individual. Un paquete puede ser parte de un flujo de paquetes. Imagine una carta de diez páginas, donde cada página es un paquete, el flujo es toda la carta y cada paquete es una página de la carta. Es decir, que cada enrutador no considera el orden de las páginas para enviarlas al siguiente salto, esto hace que en el destino los paquetes puedan llegar desordenados. En nuestro ejemplo de la carta, las páginas pueden no llegar en orden al destinatario.

El enrutador decide cual es el siguiente salto, por lo general, de acuerdo con la información del destino del paquete, utiliza su tabla de rutas donde está especificado una dirección de red de destino y la interfaz de salida correspondiente. Esta función se hace a través de dos operaciones importantes que a veces se suelen confundir: el reenvío y el enrutamiento.

El reenvío es el trabajo de transformar una trama de una interfaz de entrada a la trama correspondiente con la tecnología de la interfaz de salida. Por ejemplo, un paquete puede ingresar por una interfaz Ethernet con su trama correspondiente, y si se despacha a través de un enlace satelital la trama debe ser transformada a una trama propia de la tecnología, en este caso podría ser una trama PPP (Point to Point protocol).

El enrutamiento es el trabajo de seleccionar la interfaz de salida del paquete utilizando por lo general la tabla de rutas. Una vez que el paquete ha ingresado al enrutador, se obtiene la dirección de red de destino y se busca una coincidencia en la tabla de rutas, donde está especificada la interfaz de salida.

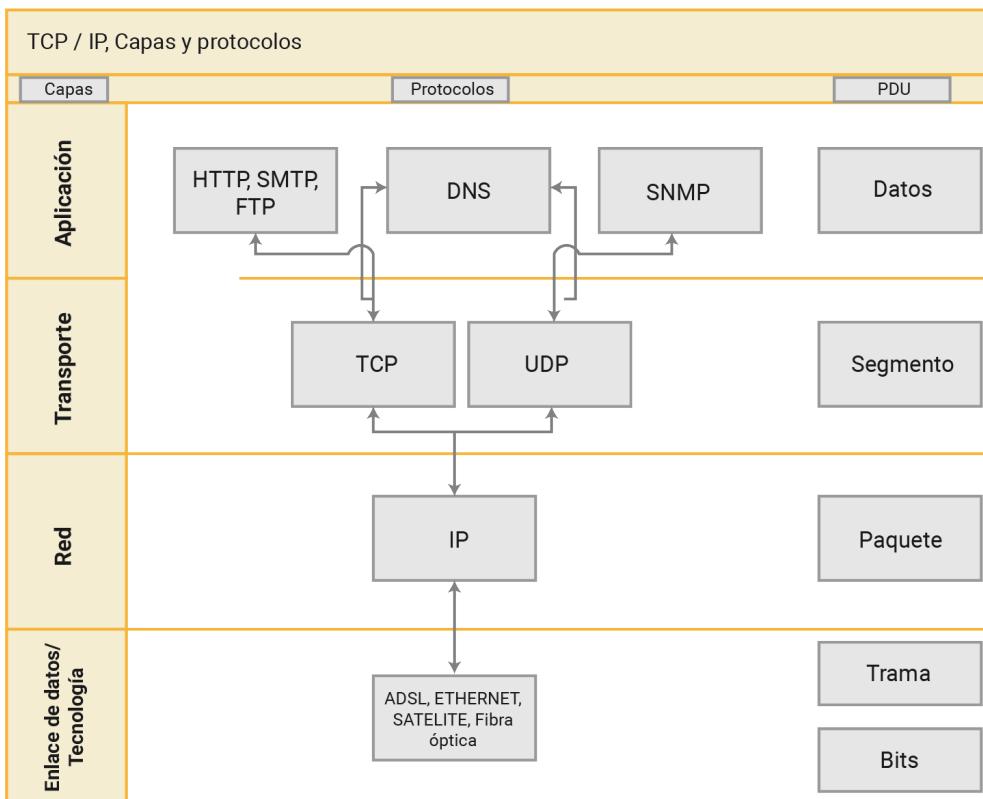
Un sistema distribuido va a utilizar toda esa infraestructura de las redes de computadores, del núcleo de la red, por lo que es necesario conocer su funcionamiento así como sus limitaciones.

2.7. Abstracción de la red por capas o niveles

Para abordar toda la complejidad de las redes de computadoras, la comunidad académica y empresarial usa una abstracción por capas que permite la definición de procesos e interfaces y la interacción de diferentes programadores, desarrolladores de hardware e investigadores.

Figura 9.

Principales protocolos en TCP/IP



Nota. Elaborado por el autor

En la Figura 9 se muestra las capas del modelo TCP/IP con los principales protocolos de cada capa. La capa de **aplicación** está conformada por todas las aplicaciones que generan o consumen información en los equipos finales. La siguiente capa es la de **transporte**, donde se definen los servicios orientados a la conexión con el uso del protocolo TCP y no orientado a la conexión con el uso del protocolo UDP. Los sistemas distribuidos, por lo general, se implementan de la capa de transporte hacia arriba. La siguiente capa la de **red** se define por lo general en el sistema operativo del

dispositivo, por lo que es más complejo modificar su funcionamiento. La última capa en la figura agrupa la capa de **enlace de datos** y la **capa física** del modelo TCP/IP. Esta última capa por lo general viene implementada en el hardware del dispositivo.

Las capas tienen una comunicación lógica con sus pares, es decir, la capa de aplicación del emisor se comunica con la capa de aplicación del receptor, la de transporte del emisor con la correspondiente capa de transporte del receptor, para hacerlo utiliza los servicios de la capa inferior y a su vez brinda servicios a la capa superior, por ejemplo, la capa de transporte usa los servicios de la capa de red y brinda servicios a la capa de aplicación.

La comunicación lógica entre capas usa su propia unidad de datos del protocolo (PDU), es así que: a nivel de aplicación el PDU son los datos, en la capa de transporte el PDU se llama segmento, en la de red se denomina paquete, en la capa de enlace de datos se denomina trama y a nivel físico son los bits.

2.8. Proceso de encapsulación

La técnica de encapsulación es muy utilizada en el área de sistemas distribuidos, por ejemplo, la aplicación Skype genera su propio dato y segmento que envía a la capa de red. Esto le permite implementar sus propios algoritmos para gestionar la calidad de servicio y el transporte de tráfico multimedia.

Le invito a revisar el siguiente recurso que explica el proceso de encapsulación.

[Proceso de encapsulación](#)

Es importante recalcar que las direcciones físicas tienen un significado local, en otras palabras, solo tienen un significado dentro de las conexiones físicas de la red, mientras las direcciones IP tienen un significado global, por ejemplo, la dirección IP del servidor web de Amazon es la misma para cualquier cliente que requiera una conexión. Cuando se establece una conexión, mientras el paquete está en tránsito, la dirección lógica o IP de origen y destino no cambian, no así las direcciones físicas de la trama

que cambian de acuerdo a la tecnología subyacente por la cual transita la información.

2.9. Identificación de componentes

La comunicación entre diferentes redes, por ejemplo, en Internet es necesario que cualquier equipo o dispositivo que requiera enviar información a otro equipo tenga una forma de identificación única en toda la red. Por lo general, se utiliza algún tipo de dirección, en Internet, por ejemplo, se utilizan direcciones Ipv4 o Ipv6. Esta dirección debe permitir identificar al dispositivo dentro de su propia red y al mismo tiempo identificar a la red a la que pertenece. Por ejemplo, en la Figura 10, que es una dirección IPv4, se puede ver que tiene dos partes, una parte para identificar la red y otra parte que identifica al dispositivo en la red. Esta división en partes permite que exista un direccionamiento jerárquico que logra que los paquetes puedan ser encaminados por los enrutadores y que puedan llegar desde una red a otra indiferentemente de su ubicación geográfica.

Figura 10.
Dirección IPv4

	Red		Host	
IP	192	168	10	2
Máscara de red	255	255	0	0

Nota. Elaborado por el autor

Recursos:

En el REA Curso Abierto de Sistemas Distribuidos de la Universidad Carlos III de Madrid (López Fuentes, 2015), específicamente en el tema 2 ([Tema 2 - Redes e interconexión – \(uc3m.es\)](#)) existe un enfoque similar de esta unidad, invito a hacer un cuadro comparativo de lo analizado en esta semana de tal forma que se amplíen sus conocimientos.



Actividades de aprendizaje recomendadas

Es importante que en esta semana usted realice las siguientes actividades para consolidar su aprendizaje:

- Leer y familiarizarse con el material didáctico de la asignatura.
 - Lea nuevamente los contenidos de esta guía y cree un mapa conceptual.
 - Consulte el plan docente donde se muestra la secuencia de estudio de los contenidos, así como las actividades que complementan el aprendizaje organizadas por fechas, con su respectiva valoración, rúbricas de evaluación y distribución de las calificaciones por bimestre.
 - El entorno virtual de aprendizaje es el medio de interacción permanente entre docente y estudiante, donde encontrará orientaciones semanales, recursos, herramientas de interacción para con sus compañeros y docente.
- Ingrese al entorno virtual de aprendizaje para conocer las orientaciones de estudio y actividades a desarrollar esta semana; así como para realizar consultas a su docente tutor e interactuar con sus compañeros.
- Lea los conceptos presentados en la tercera semana, complemente los conceptos que allí se presentan, los cursos abiertos mostrados en la sección de recursos, con estos insumos elabore su propio resumen.
- Le invito a completar la tabla: Ejemplo de tolerancia de aplicaciones en sistemas distribuidos, colocando en un valor del 1 al 10 para determinar cuál es la tolerancia a fallos, pérdidas y retardo de las aplicaciones y ambientes que hemos revisado. Complete con dos ejemplos para cada actividad. Cuando el valor es 10 es altamente tolerable y 1 no es tolerable.

[**ANEXO**](#) Ejemplo de tolerancia de aplicaciones en Sistemas Distribuidos

A continuación, se invita a que revise cómo van sus conocimientos haciendo la respectiva autoevaluación.



Autoevaluación 2

Seleccione las alternativas que considere correctas:

1. ¿Cuándo se desarrolló Internet que características no fueron consideradas?

- a. Seguridad.
- b. Comunicación extremo a extremo.
- c. Interconexión de redes.
- d. Intercambio de tráfico multimedia.

2. En las siguientes interrogantes, seleccione la alternativa correcta:

¿Cuál de los siguientes es un ejemplo de sistemas distribuidos?

- a. Email.
- b. DNS.
- c. WWW.
- d. Microsoft Word.

3. ¿Cuál de los siguientes equipos son más seguros?

- a. Mainframe.
- b. Celulares inteligentes.
- c. Teléfonos análogos.
- d. Computadores personales.

4. ¿En qué año la UTPL inauguró su primer centro de cómputo?

- a. 1979
- b. 1980
- c. 1974
- d. 1990

- 5. ¿Qué protocolo de Internet permite una abstracción de identificadores numéricos a nombres nemotécnicos?**
 - a. HTTP
 - b. DNS
 - c. SMTP
 - d. FTP
- 6. ¿Quién regula la cantidad de ancho de banda para conectarse a Internet que puede tener una organización?**
 - a. ISP
 - b. Organización
 - c. LACNIC
 - d. WWW consorcio
- 7. ¿Qué organización define los estándares para el trabajo de Internet?**
 - a. IETF
 - b. ITU
 - c. ISO
 - d. ISP
- 8. ¿Cuáles de las siguientes características son propias de Internet?**
 - a. Hay un establecimiento previo de la conexión.
 - b. Los enrutadores pueden enviar los paquetes de un mismo flujo en forma desordenada.
 - c. Los paquetes en el destino llegan con la misma cadencia.
 - d. Los paquetes de un mismo flujo llegan al receptor en forma ordenada.
- 9. Los enrutadores se encuentran en el núcleo de la red.**
 - a. Verdadero
 - b. Falso

10. Seleccione las alternativas correctas.

¿Qué acciones se llevan en la frontera de la red?

- a. Se crean las rutas o caminos de comunicación.
- b. Los enruteadores ejecutan las aplicaciones.
- c. Se implementa la fiabilidad de la conexión.
- d. Se brinda seguridad a la comunicación.

[Ir al solucionario](#)

Resultado de aprendizaje 2

- Realiza un análisis costo-beneficio para una solución de servidor propuesto.

Contenidos, recursos y actividades de aprendizaje recomendadas

Con este resultado de aprendizaje podrá determinar cuál es el costo computacional, la cantidad de pasos requeridos y el tiempo de cada uno de los algoritmos y técnicas de comunicación más conocidos. También podrá determinar el costo que involucra mantener la heterogeneidad, tanto en hardware como en software.

En esta unidad se hace un análisis de las primitivas de comunicación brindadas por las capas de red y de transporte.

En los recursos, proponemos comparar estos contenidos con los presentados por los cursos abiertos de prestigiosas universidades del mundo.

Se propone la implementación de los ejemplos mostrados en esta unidad y la revisión de escenarios diferentes para mejorar la asimilación de los conceptos y mejorar las habilidades de programación.

La recreación del código mostrado en el texto y el uso de software de análisis de tráfico para ver en tiempo real el intercambio de mensajes en la red permitirá tener una idea más precisa de los retos que deben asumir los sistemas distribuidos.

En esta unidad se hace un análisis del problema que presenta la heterogeneidad de los equipos, sistemas operativos, lenguajes y estilos de programación existentes y que deben ser considerados cuando los mensajes que generan las entidades o dispositivos son enviados por una red de computadoras.

Se analizará también la comunicación en grupo o multicast como una forma de ahorrar recursos tales como ancho de banda y procesamiento cuando se envía mensajes en un sistema distribuido.

En los recursos proponemos comparar estos contenidos con los presentados por los cursos abiertos de prestigiosas universidades del mundo.

En las actividades recomendadas se propone analizar los formatos de representación de datos más usados en la actualidad, de la misma forma se recomienda implementar el software multicast presentado como caso de análisis.



Semana 4

Unidad 3. Programación de redes de computadoras (sockets)

En esta unidad se analiza parte de las primitivas de comunicación que los sistemas distribuidos utilizan para el intercambio de mensajes. A través de ejemplos desarrollados en el lenguaje de programación Java, se muestra cómo los procesos pueden intercambiar información para poder coordinar y realizar actividades en conjunto.

3.1. Implementación de la comunicación en redes de computadoras

En esta unidad abordaremos los conceptos importantes de la comunicación entre computadores desde el punto de vista del desarrollo de un sistema. Veremos las interfaces para los sockets en los protocolos TCP y UDP. Utilizaremos como ejemplo las librerías proporcionadas por TCP y UDP.

Abordaremos también el problema y las soluciones de la estandarización de la representación de datos cuando se envía información de un equipo a otro a través de una red de computadores. El problema se presenta debido a la heterogeneidad que existe en los sistemas operativos, en el hardware y en la representación de los datos que cada sistema operativo brinda al programador.

3.2. Protocolos

Los protocolos de comunicaciones se refieren a un conjunto de reglas y formatos. Al decir reglas, estamos diciendo que hay una secuencia de pasos y acciones o mensajes que se deben intercambiar; y formatos, que son usados en la comunicación entre procesos para especificar el formato de los datos en los mensajes.

En la vida real también existen protocolos, por ejemplo, imaginen que están ingresando a un país extranjero a través de la oficina de migración, en estos espacios hay reglas que se deben cumplir para poder pasar todos los controles: filas para ciudadanos extranjeros, filas para ciudadanos residentes, filas rápidas; y también hay formatos: el uso del lenguaje común, los itinerarios y las reservas. Otro ejemplo es cuando conversamos con una persona, en primer lugar, existen unas reglas tácitas, un saludo, luego una identificación y luego un formato de lenguaje que depende de la cercanía con la persona, si es un familiar será un formato más relajado, si es un extraño será un formato más formal.

En Internet, por ejemplo, el protocolo IP tiene algunas funciones, entre ellas la de permitir que la información viaje desde un emisor por un receptor a través de la red de núcleo. IP define una identificación global tanto para el equipo inicial como para el equipo final denominada dirección IP, envía la información a través de una estructura denominada paquete, la misma que está conformada por una cabecera y la carga útil que es donde la información que viene desde las capas superiores se almacena.

Cuando se implementa un protocolo en software, se debe analizar cuáles son las reglas que se necesitan, definidas, como por ejemplo, en el orden en que se deben intercambiar los mensajes, con qué intervalo de tiempo se pueden intercambiar los mensajes; también se debe definir la cantidad de mensajes necesarios para su funcionamiento, el formato en el cual estos mensajes intercambian información, la cantidad de datos, su tipo de dato y la longitud de la información que tiene cada tipo de mensaje.

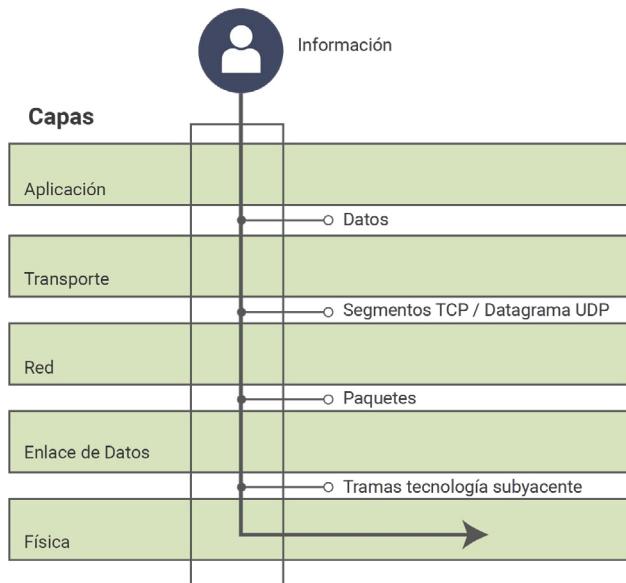
Continuemos con el aprendizaje mediante la revisión del tema: Capas de comunicaciones y el software.

3.3. Capas de comunicaciones y el software

En la Figura 11 se ve la relación de las capas con las unidades de datos de protocolo, entre cada capa se definen interfaces para que según la capa se puedan entregar datos, datagramas, paquetes o tramas respectivamente.

Figura 11.

Capas de comunicación de datos y software



Nota. Elaborado por el autor

Nos vamos a concentrar en el estudio de las capas para lo cual solicito concentrar su atención en las siguientes explicaciones:

Cada capa proporciona un servicio a la capa superior y utiliza los servicios de la capa inferior a través de la interfaz definida. Cuando la relacionamos en software, una interfaz es similar a una función o un procedimiento.

Una función o procedimiento realiza un proceso para lo cual debe definir qué tipos de datos necesita como entrada o como argumento, para luego de procesarla dar un resultado.

Desarrollar aplicaciones para las capas física, enlace de datos y de red no es muy común debido a que estas funcionalidades están definidas en el hardware de comunicación y en el sistema operativo. Sin embargo, para que las aplicaciones puedan funcionar, el sistema operativo debe brindar

a los programadores las interfaces necesarias para las operaciones de la capa de red e inclusive la capa de enlace de datos.

A nivel de la capa de aplicación, cuando se desarrolla un sistema informático, para poder utilizar la infraestructura de red se debe acceder a las interfaces de la capa de transporte.

Las interfaces en las capas de transporte, dependiendo de la aplicación, pueden ser orientadas a la conexión, como por ejemplo las definidas para el protocolo TCP y no orientadas a la conexión con el protocolo UDP. En Internet, indiferente del tipo de conexión requerido por lo general en la capa de transporte, se debe **identificar la aplicación** o el proceso que están usando los servicios y la aplicación o proceso al que se quiere conectar en el destino, se lo hace con un identificador denominado **número de puerto**. Por ejemplo, cuando se utiliza diferentes pestañas del navegador en el emisor cada pestaña se identifica a través de un número de puerto diferente. Cuando desde la pestaña del navegador se desea acceder a un servicio web entonces la aplicación de destino va a estar identificada por un puerto que generalmente será el puerto 80.

Existen sistemas distribuidos que han creado e implementado su propio software, sus propios procesos para la capa de transporte, colocando sus propios algoritmos para el transporte fiable y la gestión de errores.

Cuando la capa de transporte desea usar los servicios de la capa de red, entrega el segmento o datagrama a través de una interfaz declarada por la capa de red. En este caso, la capa de red debe identificar al emisor y/o al receptor. Lo hace a través de una dirección IP. En el caso de la capa de red el sistema operativo debe proporcionar al desarrollador las interfaces para la identificación del equipo y gestión del paquete cuando está transitando por el núcleo de la red.

3.4. Interfaz de programación de aplicaciones

Cuando se diseña y se implementa un sistema, en especial un sistema distribuido, se debe determinar qué interfaces se va a utilizar por cada una de las capas del modelo de telecomunicaciones. En la Figura 12 se ha considerado las interfaces para las capas de red hasta la capa de aplicación.

Usaremos el lenguaje de programación Java para colocar ejemplos de cómo se han implementado las interfaces para cada una de las capas desde la capa de red.

En las aplicaciones y servicios se definen las interfaces para la interacción del usuario con el sistema. Las aplicaciones o servicios para la comunicación entre procesos en la red pueden usar cualquiera de las siguientes opciones:

1. Directamente a través de la interfaz de la capa de transporte definida en los protocolos TCP o UDP.
2. Pueden implementar sus propios algoritmos para la capa de transporte, lo que se denomina **middleware**, Figura 12. La capa de Middleware está entre la capa de aplicación y la de red. Por ejemplo, Skype ha definido su propio middleware para su sistema de comunicación.
3. Se puede usar una invocación remota. En la invocación remota en programación se da cuando desde un equipo se usan los datos, las funciones o procedimientos ubicados en un equipo remoto. Para que la comunicación sea exitosa, el equipo local y remoto deben existir en el mismo tiempo y ser accesibles.
4. Cuando se usa comunicación indirecta, la responsabilidad de la comunicación entre el equipo local y remoto está asignada a un equipo o proceso en el medio. Un ejemplo de este tipo de comunicación es Twitter. Aquí los equipos local y remoto no necesariamente deben existir en el mismo tiempo.

Cualquiera de las opciones listadas anteriormente pueden utilizar las primitivas de comunicación de las capas inferiores, por ejemplo: los sockets para TCP y UDP, o las interfaces para una comunicación multicast, o el paso de mensajes que vamos a ver en las siguientes secciones.

Figura 12.

Interfaz de programación de aplicaciones



Nota. Elaborado por el autor

3.5. Interfaz de la capa de red

El tener una dirección IP única en toda la red es un requisito indispensable para que la comunicación se lleve adecuadamente. Existen dos tipos de direcciones IP, las definidas en el protocolo IPv4 con 32 bits, que se representan en forma decimal dividiendo los 32 bits en 4 octetos, transformando cada octeto a un número decimal y colocando un punto entre cada octeto. Por ejemplo 23.25.2.2. Las direcciones definidas en IPv6 tienen una longitud de 128 bits y se representan en forma hexadecimal dividiendo los 128 bits en grupos de 16 bits, cada grupo se separa con dos puntos. Cada grupo se representa por cuatro elementos hexadecimales. Por ejemplo, 2001:0db8:85a3:08d3:1319:8a2e:0370:7332

Para mostrar las interfaces para direccionamiento que brindan los sistemas operativos para la capa de red usaremos el lenguaje de programación Java. Para representar una dirección IP Java tiene una clase de alto nivel llamada *InetAddress*.

Figura 13.

Código de ejemplo para IP

```
1 package com.myjava.ip;
2 import java.net.InetAddress;
3 import java.net.UnknownHostException;
4 public class MyHostName {
5 public static void main(String a[]){
6 try {
7 InetAddress host = InetAddress.getByName("172.16.20.55");
8 System.out.println(host.getHostName());
9 } catch (UnknownHostException ex) {
10 ex.printStackTrace();
11 }
```

Nota. Elaborado por el autor

En el ejemplo de la Figura 13 tenemos un código en Java. Para poder utilizar el direccionamiento IP tenemos que importar la clase InetAddress línea 1. La siguiente clase que se importa en la línea 2 es UnknownHostException la que permite obtener errores del sistema operativo para enviarlos a una variable, de tal forma que el programa no se pare por un mal funcionamiento. Una vez importadas las clases necesarias para el funcionamiento se instancia en la línea 7 en una variable *host* la cual almacenará los datos del direccionamiento IP, en este caso para IPv4. Podemos ver el uso de dos métodos *getByName* y *getHostName*. El primer método obtiene la dirección IP del host dándole como argumento un nombre o una dirección IP. Cuando se da como argumento la dirección IP, el método solo verifica que el formato esté correcto. Cuando se da un nombre del host, como por ejemplo “WWW.utpl.edu.ec”, Java implícitamente utiliza los servicios de DNS del sistema operativo para poder obtener una dirección IP. El método *getHostName*, línea 8, obtiene el nombre del host para su dirección IP.

En las actividades complementarias existen más ejercicios para usar y verificar el uso de distintos métodos de la clase InetAddress.

Existen algunas direcciones especiales que pueden ser usadas, como *localhost* o *127.0.0.1*, que identifican al mismo equipo y que sirven para verificar que el programa está funcionando adecuadamente sin considerar el envío por una red de computadoras.

3.6. Comunicación entre procesos

En sistemas distribuidos, los entes de comunicación los vamos a denominar procesos, en el mundo real los procesos hacen referencia a equipos o software.

Le invito a profundizar sus conocimientos acerca de este importante tema.

Para simplificar el estudio empezaremos definiendo un proceso emisor y *un proceso receptor en una comunicación unidireccional de $p \rightarrow q$*

Figura 14.
Comunicación entre procesos



Nota. Elaborado por el autor

En la Figura 14 se puede analizar algunos componentes en general de la comunicación entre procesos. En primer lugar, el proceso p tiene una función o primitiva *enviar* que toma un mensaje m , le da formato y lo deposita en la memoria de mensajes de salida para que luego sea enviada a través de las funciones del canal de comunicación. El proceso q a través de la primitiva o función *recibir* toma de memoria de mensajes la entrada.

La memoria de mensajes de entrada es usada cuando existe una comunicación orientada a la conexión. El emisor guarda el mensaje hasta que él tenga una retroalimentación de que ha sido entregado con éxito y sin errores.

La memoria de mensajes de salida es usada en el receptor debido a que los mensajes pueden llegar desordenados por el dinamismo de la red de núcleo, una vez que se tenga los mensajes ordenados se envían a la primitiva recibir del proceso.

La gestión de la memoria de los mensajes de entrada y salida es un tema de estudio en el área de ciencias de la computación y tiene relación con conceptos como control de congestión, control de flujo, manejo de colas, calidad de servicio.

Existe un tiempo para que el mensaje m llegue del proceso p al proceso q . Este tiempo no puede ser infinito ni tampoco instantáneo. Para determinar el tiempo en el que las primitivas deben ejecutarse debemos adicionar algunos tiempos: el tiempo de procesamiento, el tiempo de almacenamiento en el buffer, el tiempo de despacho del mensaje, la latencia o el tiempo de viaje del mensaje en la red. La latencia va a depender directamente de la distancia entre los procesos que se comunican.

El tipo de interacción entre procesos se puede dar de dos formas, a través de una comunicación síncrona o asíncrona.

Comunicación síncrona significa que cuando se envía el mensaje m se puede conocer en forma determinista el tiempo en el que llega al destino y por lo tanto se puede decidir en qué momento la primitiva *recibir* se puede ejecutar.

En una comunicación síncrona, los procesos p y q usan las primitiva *enviar* y *recibir* respectivamente, las bloquean y no procesan más mensajes hasta que el tiempo determinado se cumpla. En otras palabras, las primitivas *enviar* y *recibir* son operaciones bloqueantes.

Comunicación asíncrona significa que las primitivas de *enviar* y *recibir* no pueden determinar el tiempo en el que deben procesar los mensajes.

En la comunicación asíncrona las primitivas *enviar* y *recibir* son operaciones no bloqueantes. Esto involucra que el emisor puede enviar mensajes sin esperar confirmación de que el mensaje ha arribado, de la misma forma el receptor puede entregar el mensaje al proceso sin considerar el orden en el que llegue.

Recursos:

En el REA Curso Abierto de Sistemas Distribuidos de la Universidad Carlos III de Madrid (López Fuentes, 2015), específicamente en el tema 5 ([Tema 5 - Comunicación con sockets – \(uc3m.es\)](#)) existe un enfoque similar de esta unidad, invito a hacer un cuadro comparativo de lo analizado de tal forma que se amplíen sus conocimientos.



Actividades de aprendizaje recomendadas

Es importante que en esta semana usted realice las siguientes actividades para consolidar su aprendizaje:

- Leer y familiarizarse con el material didáctico de la asignatura.
 - Lea nuevamente los contenidos de esta guía y cree un mapa conceptual.
 - Consulte el plan docente donde se muestra la secuencia de estudio de los contenidos, así como las actividades que complementan el aprendizaje organizadas por fechas, con su respectiva valoración, rúbricas de evaluación y distribución de las calificaciones por bimestre.
 - El entorno virtual de aprendizaje es el medio de interacción permanente entre docente y estudiante, donde encontrará orientaciones semanales, recursos, herramientas de interacción para con sus compañeros y docente.
- Ingrese al entorno virtual de aprendizaje para conocer las orientaciones de estudio y actividades a desarrollar esta semana; así como para realizar consultas a su docente tutor e interactuar con sus compañeros.
- Lea los conceptos presentados en la cuarta semana, complemente los conceptos que allí se presentan con los cursos abiertos de sistemas distribuidos, con estos insumos elabore su propio resumen.
- Para el ejemplo de la librería *InetAddress* de Java, use diferentes computadores y diferentes direcciones, capture el tráfico de red con un software de análisis como Wireshark (Foundation, 2016).

Nota: conteste las actividades en un cuaderno de apuntes o en un documento word.



3.7. Sockets de red

Un socket es un concepto abstracto que se usa en programación para definir la comunicación entre diferentes procesos ubicados, por lo general, en diferentes computadores unidos por una red. A través del socket, los procesos pueden enviar información de un proceso a otro.

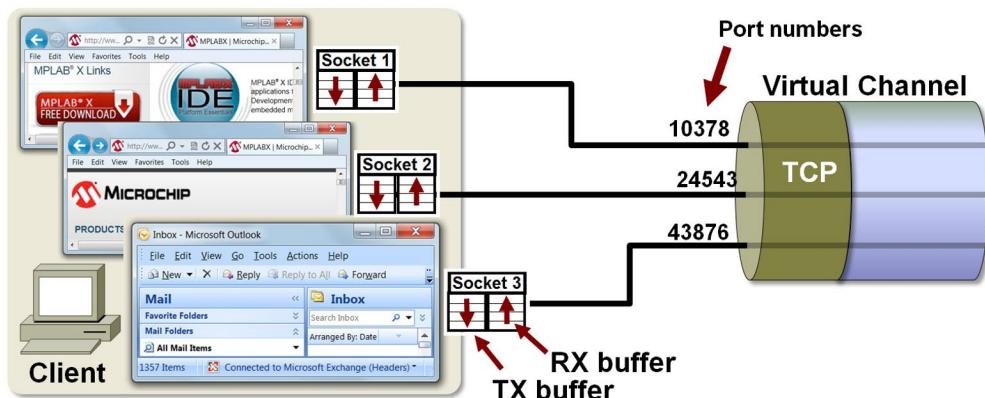
Las primitivas para acceder a los sockets son proporcionadas por el sistema operativo. La comunicación con sockets entre procesos es muy similar a las operaciones de entrada y salida de archivos, y es independiente del lenguaje de programación que se utilice.

Para funcionar, un socket necesita identificar cuatro valores:

- a. La identificación del origen, la dirección IP del origen.
- b. La identificación del destino, la dirección IP de destino.
- c. La identificación del proceso origen, número de puerto de origen.
- d. La identificación del proceso destino, número de puerto de destino.

El conjunto de estos cuatro valores identifica en forma única a la comunicación. Esta característica en la práctica permite que, por ejemplo, en un cliente podamos tener abiertas dos pestañas de un navegador (dos procesos diferentes) comunicándose con el mismo servidor web. En este ejemplo, al cambiar un valor de los cuatro valores del socket se crea una identificación única para cada comunicación.

Figura 15.
Sockets en un dispositivo



Nota. Tomado de *Introduction to TCP/IP (Part 4) - Sockets and Ports - Developer Help*, n.d.

En la Figura 15 se puede ver el ejemplo de los sockets para TCP/IP y los componentes de este. En TCP/IP un número de puerto para el socket tiene un tamaño de 16 bits. Esto significa que un mismo equipo puede tener abierto 65536 procesos comunicándose a través de la red. En otras palabras, se pueden comunicar 2^{16} aplicaciones en el equipo al mismo tiempo, un número bastante grande que permite que varias aplicaciones o procesos en el mismo equipo tengan una alta probabilidad de tener un número diferente de puerto.

En TCP/IP existe una distinción histórica de los números de puertos: los números de puerto bien conocidos van de 1 a 1023, se usan por lo general para aplicaciones y protocolos estándares como HTTP, SMTP, DNS. Tomemos, por ejemplo, el protocolo HTTP, cuando se configura un servidor web se crea un proceso que estará escuchando generalmente en el puerto 80, es decir, que cualquier cliente que quiera conectarse a este servidor usará, para crear un socket, la dirección IP del servidor y el número de puerto 80 como parámetros de dirección IP y puerto destino respectivamente. El cliente, a través de un proceso o aplicación, en este caso un navegador, con su propia dirección y un número de puerto completará toda la información para crear efectivamente el socket y comenzar el intercambio de información con el servidor. Los números de puerto en el cliente se generan en forma aleatoria de entre el rango de números enteros de 1024 a 65535.

3.8. Comunicación de datagramas con UDP

Uno de los servicios que puede usar una comunicación entre procesos en TCP/IP es el uso de las primitivas de comunicación dadas por el protocolo de datagrama de usuario (UDP) (Postel, 1980). UDP es una extensión en la capa de transporte de IP, significa que no es confiable, no permite entregar los datagramas en orden, pueden llegar con errores y se pueden perder en el camino.

Comparado con TCP, UDP entrega la información con mayor velocidad, lo que lo hace ideal para tráfico no pesado pero intenso como DNS, y para tráfico no tolerante al retardo como videoconferencias.

Figura 16.

Formato de cabecera UDP

32 bits	
Puerto de origen	Puerto de destino
Longitud	Suma de comprobación
Datos	

Nota. Elaborado por el autor

En la Figura 16 se muestra el formato de cabecera de UDP, en el cual se pueden apreciar los campos necesarios en la cabecera, puerto de origen y destino, longitud del mensaje y la suma de comprobación. El campo de longitud de mensaje es usado por el destino para determinar el tamaño de los datos y pasarlo al proceso. El campo de suma de comprobación utiliza un método de definición de integridad simple que permite determinar si el datagrama cambió en el viaje por la red de núcleo o fue alterado por una tercera entidad.

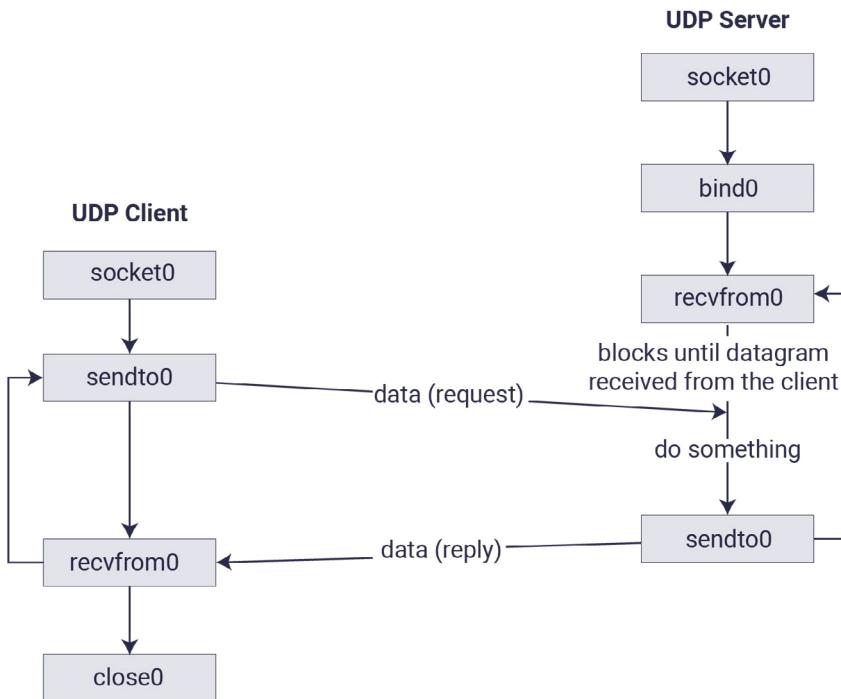
Cuando se usa UDP, la capa superior o de aplicación es la encargada de detectar y corregir errores. La corrección de errores en la capa de aplicación se hace utilizando mecanismos propios. Como ejemplo, si usamos una técnica de proyección o pronóstico, si se envía una foto por Internet usando UDP (una foto es un arreglo bidimensional de píxeles, mientras más píxeles mejor es la resolución y cada píxel representa un color) y para simplificar imagine que cada píxel se puede enviar en un datagrama UDP.

Si la aplicación en el destino detecta que un píxel se ha perdido, puede determinar con una alta probabilidad de acierto qué color tenía el píxel que se perdió basado en la información de los píxeles circundantes. Suponga con todos los píxeles que rodean un píxel faltante son negros, entonces se podría decir que el color del píxel faltante es negro. En este ejemplo es más eficiente y veloz determinar el color faltante que pedir al emisor que reenvíe un datagrama con la información del píxel que no llegó.

Una visión de diagrama de flujos de datos e interacción entre cliente y servidor de UDP se muestra en la Figura 17. Aquí se puede ver algunas primitivas que son utilizadas, tanto para el cliente como para el servidor. Hablaremos de cada una de ellas.

Figura 17.

Interacción UDP entre el cliente y servidor



Nota. Elaborado por el autor

El **servidor** define su servicio a través de la primitiva `socket()`, luego con la primitiva `bind()` se le asigna un número de puerto y un identificador de proceso para su uso en el sistema operativo. Una vez que toda la información está completa el servidor espera por una conexión desde el cliente con la primitiva `recvfrom()`. Esta primitiva es bloqueante, significa que no cambiará de estado hasta que se reciba un datagrama desde el

cliente. Una vez recibido el datagrama el servidor envía información al cliente a través de la primitiva `sendto()`. El servidor puede generar un ciclo repetitivo hasta que el cliente no tenga más datagramas que enviar.

En el **cliente**, los estados son similares al servidor. Define el servicio a través de la primitiva `socket()`. Con la información recibida de la capa de aplicación genera los datagramas y los envía al servidor a través de la primitiva `sendto()`. El cliente puede recibir datagramas del servidor a través de la primitiva `recvfrom()`. Una vez que ha enviado todos los datagramas libera los recursos ocupados por el socket a través de la primitiva `close()`.

Continuemos con el aprendizaje mediante la revisión del Código Java de un servidor y un cliente UDP.

3.8.1. Código Java de un servidor UDP

Estudiaremos cómo implementar una comunicación para UDP en lenguaje de programación Java. Haremos el análisis del programa tanto para el servidor como para el cliente. El objetivo de este programa es que el servidor reciba una cadena de caracteres, la transforme en mayúsculas y la devuelva al cliente.

En la Figura 18 vemos un ejemplo de un programa para el servidor. Las líneas 1 a 5 importan las clases necesarias para el manejo de excepciones (problemas informados por el sistema operativo), para generar datagramas UDP *DatagramPacket*, para crear un socket UDP *DatagramSocket* y también para generar un paquete IP con la librería *InetAddress*.

En la línea 7 se define el número de puerto en el cual el servidor escuchará y será usado para la creación del socket UDP. La línea 10 instancia un socket UDP para recibir respuestas del cliente, aquí es usado el número de puerto definido anteriormente. El cliente debe saber de antemano en qué puerto está escuchando el servidor para poder crear la comunicación. Las líneas 11 y 12 crean espacios de memoria para la recepción y envío de información respectivamente. La línea 13 instancia un datagrama UDP para recibir información del cliente y almacenarla en la memoria de recepción. La línea 15 implementa la primitiva de recepción para recibir datos del cliente y almacenarlos en el datagrama UDP. Las líneas 16 y 17 muestran la información que envió el cliente. La línea 18 transforma la información a mayúsculas y la almacena en la memoria de envío. Las líneas 19 y 20 obtienen la información del paquete IP que envío el cliente, esta información, dirección IP y número de puerto será usada para retornar

la información al cliente. La línea 21 crea un paquete IP y lo almacena en la memoria de envío para luego retornar la información al cliente, aquí se observa el uso de la dirección IP y el puerto obtenido en las líneas 19 y 20. La línea 22 envía el paquete IP al cliente para finalmente en la línea 23 cerrar el socket.

Figura 18.
Servidor Java UDP

```
1 import java.io.IOException;
2 import java.net.DatagramPacket;
3 import java.net.DatagramSocket;
4 import java.net.InetAddress;
5 import java.net.SocketException;

6 public class UDPServer{
7     public final static int SERVICE_PORT=65010;
8     public static void main(String[] args) throws IOException{
9         try{
10             DatagramSocket serverSocket = new DatagramSocket(SERVICE_PORT);
11             byte[] receivingDataBuffer = new byte[1024];
12             byte[] sendingDataBuffer = new byte[1024];
13             DatagramPacket inputPacket = new DatagramPacket(receivingDataBuffer,
14 receivingDataBuffer.length);
15             System.out.println("Waiting for a client to connect...");

16             serverSocket.receive(inputPacket);

17             String receivedData = new String(inputPacket.getData());
18             System.out.println("Sent from the client: "+receivedData);

19             InetAddress senderAddress = inputPacket.getAddress();
20             int senderPort = inputPacket.getPort();

21             DatagramPacket outputPacket = new DatagramPacket(
22                 sendingDataBuffer, sendingDataBuffer.length,
23                 senderAddress, senderPort
24             );
25             serverSocket.send(outputPacket);
26             serverSocket.close();
27         }
28     }
29 }
```

Nota. Elaborado por el autor

La programación de comunicación UDP tiene algunas limitaciones adicionales a la de la transferencia no confiable, una de ellas es el tamaño del mensaje que debe ser acordado entre el cliente y el servidor, si el tamaño del datagrama es más grande este será recortado. Las primitivas de envío son no bloqueantes mientras que la de recepción es una primitiva bloqueante. Para evitar que la primitiva de recepción espere de forma indefinida se puede implementar límites de tiempos a través del uso de temporizadores.

3.8.2. Código Java de un cliente UDP

En la Figura 19 se muestra el código para la comunicación del cliente con el código del servidor analizado anteriormente. Vamos a revisar las líneas de código relevantes.

Las líneas 1 a la 5 han sido explicadas anteriormente y tienen las mismas funciones. La línea 7 especifica el número de puerto en el cual el servidor está escuchando y que ha sido previamente informado al cliente. La línea 10 instancia el socket del cliente, al no colocar un número de puerto específico se generará uno aleatorio. En la línea 11 se especifica la dirección IP del servidor, como argumento se puede colocar un nombre canónico o una dirección IP. En este caso se ha colocado el nombre canónico especial "localhost", que se refiere a que el servidor y el cliente están ejecutándose en el mismo equipo. Las líneas 12 y 13 crean los espacios de memoria de envío y recepción. La línea 14 almacena una cadena de caracteres para que en la línea 15 se transforme a bytes y se almacene en la memoria de envío. En la línea 16 se crea el paquete IP con la identificación IP del servidor, puerto del servidor y la información a enviar. En la línea 17 se usa la primitiva `send()` del socket para enviar el paquete IP. En la línea 18 se espera y en la línea 19 se recibe la respuesta del servidor para finalmente en la línea 21 cerrar el socket.

Figura 19.

Cliente UDP Java

```
1 import java.io.IOException;
2 import java.net.DatagramPacket;
3 import java.net.DatagramSocket;
4 import java.net.InetAddress;
5 import java.net.SocketException;

6 public class UDPClient{
7     public final static int SERVICE_PORT = 65010;

8     public static void main(String[] args) throws IOException{
9         try{
10             DatagramSocket clientSocket = new DatagramSocket();

11             InetAddress IPAddress = InetAddress.getByName("localhost");

12             byte[] sendingDataBuffer = new byte[1024];
13             byte[] receivingDataBuffer = new byte[1024];

14             String sentence = "Hello from UDP client";
15             sendingDataBuffer = sentence.getBytes();

16             DatagramPacket sendingPacket = new
16 DatagramPacket(sendingDataBuffer,sendingDataBuffer.length,IPAddress,
16 SERVICE_PORT);

17             clientSocket.send(sendingPacket);

18             DatagramPacket receivingPacket = new
18 DatagramPacket(receivingDataBuffer,receivingDataBuffer.length);
19             clientSocket.receive(receivingPacket);

20             String receivedData = new String(receivingPacket.getData());
21             System.out.println("Sent from the server: "+receivedData);

22             clientSocket.close();
23         }
24         catch(SocketException e) {
25             e.printStackTrace();
26         }
27     }
}
```

Nota. Elaborado por el autor

Recursos:

En el REA Curso Abierto de Sistemas Distribuidos de la Universidad Carlos III de Madrid (López Fuentes, 2015), específicamente en el tema 5 ([Tema 5 - Comunicación con sockets – \(uc3m.es\)](#)) existe un enfoque similar de esta unidad, invito a hacer un cuadro comparativo de lo analizado de tal forma que se amplíen sus conocimientos.



Actividades de aprendizaje recomendadas

Es importante que en esta semana usted realice las siguientes actividades para consolidar su aprendizaje:

- Leer y familiarizarse con el material didáctico de la asignatura.
 - Lea nuevamente los contenidos de esta guía y cree un mapa conceptual.
 - Consulte el plan docente donde se muestra la secuencia de estudio de los contenidos, así como las actividades que complementan el aprendizaje organizadas por fechas, con su respectiva valoración, rúbricas de evaluación y distribución de las calificaciones por bimestre.
 - El entorno virtual de aprendizaje es el medio de interacción permanente entre docente y estudiante, donde encontrará orientaciones semanales, recursos, herramientas de interacción para con sus compañeros y docente.
- Ingrese al entorno virtual de aprendizaje para conocer las orientaciones de estudio y actividades a desarrollar esta semana; así como para realizar consultas a su docente tutor e interactuar con sus compañeros.
- Lea los conceptos presentados en la quinta semana, complemente los conceptos que allí se presentan con los cursos abiertos de sistemas distribuidos, con estos insumos elabore su propio resumen.
- Para el ejemplo de la librería *DatagramSocket* de Java, use diferentes computadores y diferentes direcciones, capture el tráfico de red

con un software de análisis como Wireshark (Foundation, 2016). Identifique los campos de dirección y puertos del origen y del destino.

- Cambie el programa de UDP para que siga recibiendo datagramas hasta recibir un comando salir.
- Consultar los números de puerto de 20 aplicaciones bien conocidas cuyos números de puertos sean menores a 1024.

Nota: conteste las actividades en un cuaderno de apuntes o en un documento Word.



Semana 6

3.9. Comunicación de segmentos con TCP

El Protocolo de Control de Transmisión (TCP) es otro servicio que se puede utilizar para la comunicación entre procesos (Goralski, 2009). TCP brinda una entrega confiable de los mensajes, es un servicio orientado a la conexión que, previo a enviar información, negocia parámetros de conexión como por ejemplo: tamaño del segmento, cantidad de segmentos a enviar por flujo y una identificación de los segmentos. TCP también implementa un espacio de memoria tanto en el emisor y en el receptor para el control de flujo y de congestión. A diferencia del protocolo UDP, la aplicación puede seleccionar el tamaño del mensaje.

Debido a que TCP utiliza la red de núcleo que es no confiable para el intercambio de paquetes, debe implementar un modelo de fallas de acuerdo a los siguientes escenarios:

1. Los segmentos no llegan en orden al receptor: en este caso es necesario que TCP identifique cada uno de los segmentos que son enviados. Además, en el receptor se almacenan los segmentos desordenados, para entregarlos a la capa de aplicación cuando la secuencia de segmentos sea la correcta y este completa.
2. Los segmentos se pierden en tránsito: para este escenario TCP implementa el uso de mensajes de retroalimentación del receptor al emisor informando que un mensaje llegó con o sin problemas.

También utiliza los identificadores para determinar qué segmento se perdió.

3. Los segmentos se corrompen o son modificados: para este escenario, además de los mensajes de retroalimentación, utiliza un campo de suma de verificación para determinar la integridad de los datos.
4. El emisor no recibe un mensaje de retroalimentación: en este escenario el emisor utiliza un espacio de memoria para almacenar los segmentos en tránsito, los segmentos serán eliminados de la memoria una vez que se tenga un mensaje de retroalimentación correspondiente positivo. El emisor usa temporizadores para no esperar indefinidamente el mensaje de retroalimentación positivo. En el caso de que el temporizador fenezca tomará el segmento de la memoria y lo reenviará nuevamente.
5. El espacio de memoria en el receptor se llena: en este caso el proceso de gestionar el espacio de memoria en el receptor también es denominado control de flujo. En este escenario el receptor informa a través de los mensajes de retroalimentación el espacio disponible en memoria, de tal forma que el emisor con esta información reduzca la cantidad de segmentos enviados. Con esto se logra que un emisor rápido no sobrecargue a un receptor lento.
6. Conexiones rotas: los procesos que usan la conexión no pueden distinguir si la falla se da en los procesos emisores o receptores o en la red de núcleo. Una falla en la red de núcleo, por ejemplo, es cuando un enrutador dejó de funcionar. Una falla del proceso es cuando un programa levantado, por ejemplo, para un servidor web dejó de responder, aunque el equipo físico siga funcionando y mostrando disponibilidad en la red. Aquí, el uso de temporizadores es requerido para cerrar conexiones que se encuentren abiertas en forma indefinida.

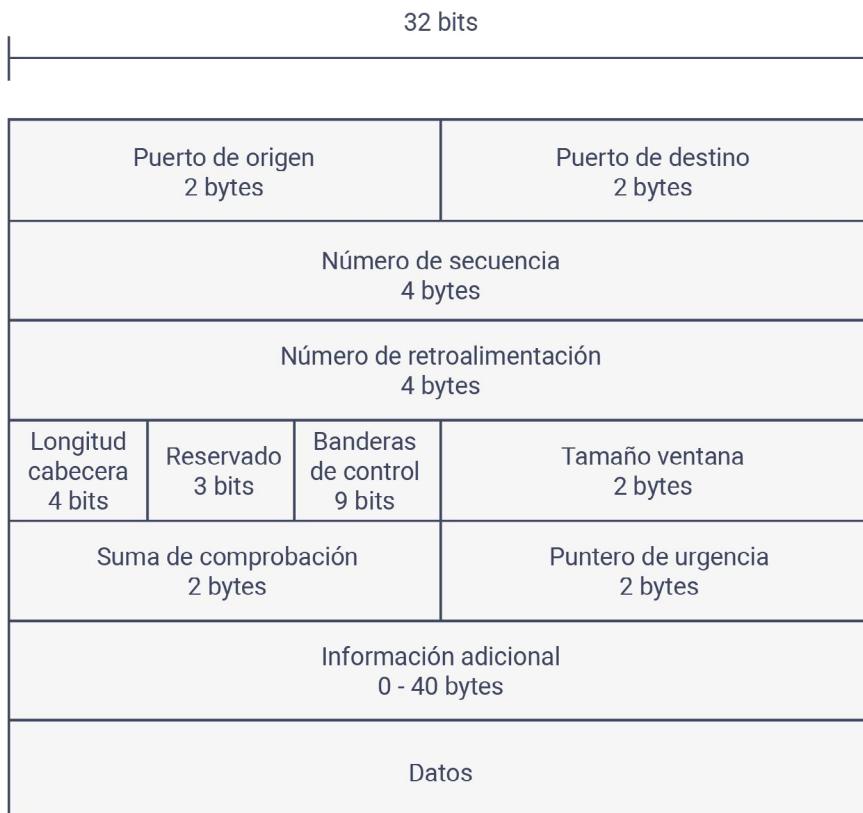
Para lograr implementar una comunicación confiable, TCP define un formato de segmento que se aprecia en la Figura 20. El segmento tiene dos partes bien definidas: la cabecera y los datos.

Los campos de la cabecera son utilizados para las siguientes tareas:

1. Negociación de los parámetros de comunicación, saludo de tres vías, cierre de la comunicación: banderas de control.
2. Segmentos en desorden: números de secuencia.
3. Gestión de fallas y/o pérdida de segmentos: números de secuencia, banderas de control, números de retroalimentación.
4. Integridad —los segmentos son modificados en la red de núcleo—: suma de comprobación de cabecera y número de secuencia.
5. Control de flujo o memoria de recepción llena: tamaño de la ventana.

Figura 20.

Formato de segmento TCP



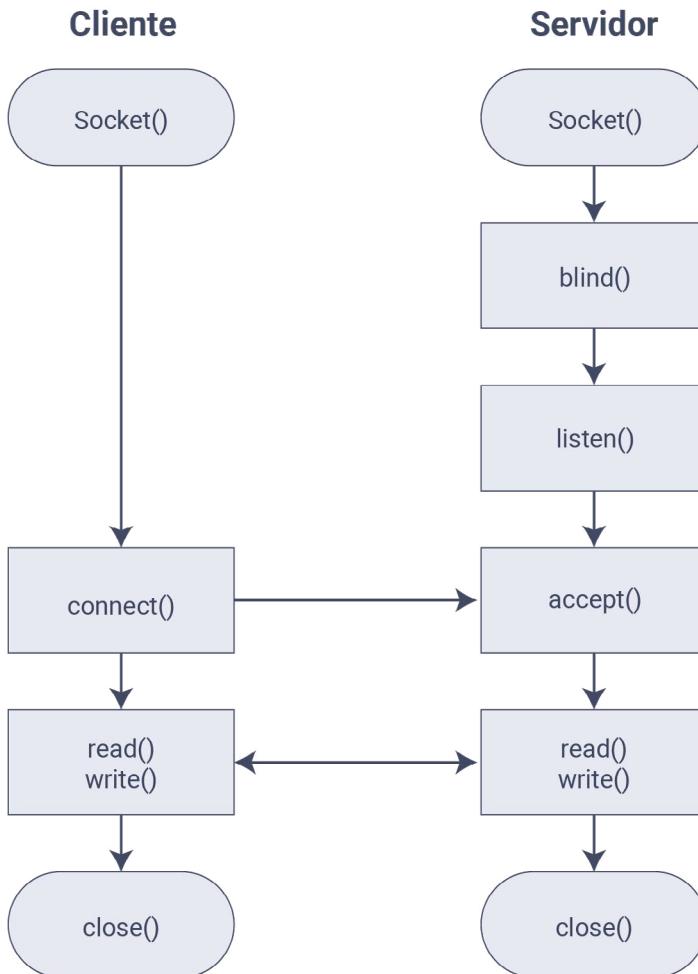
Nota. Elaborado por el autor

Una visión de diagrama de flujos de datos e interacción entre cliente y servidor para el protocolo TCP se muestra en la Figura 21. Aquí se puede

ver algunas primitivas que son utilizadas, tanto para el cliente como para el servidor. Hablaremos de cada una de ellas.

Figura 21.

Flujo de datos e interacción cliente / servidor TCP



Nota. Elaborado por el autor

El **servidor** define su servicio a través de la primitiva `socket()`, luego con la primitiva `bind()` se le asigna un número de puerto y un identificador de proceso para uso en el sistema operativo. Una vez que toda la información está completa, el servidor espera por una conexión desde el cliente con la primitiva `listen()`. Esta primitiva es bloqueante, significa que no cambiará de estado hasta que se reciba un segmento desde el cliente. Con la primitiva `accept()` negocia los parámetros de conexión con el cliente. Una vez recibido el segmento es procesado a través de la primitiva `read()` y envía

información al cliente a través de la primitiva `write()`. El servidor puede generar un ciclo repetitivo hasta que el cliente no tenga más datagramas que enviar. Con la primitiva `close()` se liberan los recursos en el servidor que fueron tomados por el socket y se cierra la conexión.

En el **cliente** los estados son similares al servidor. Define el servicio a través de la primitiva `socket()`. Empieza a negociar la conexión con el servidor con la primitiva `connect()`. Con la información recibida de la capa de aplicación genera los datagramas y los envía al servidor a través de la primitiva `write()`. El cliente puede recibir segmentos del servidor a través de la primitiva `read()`. Una vez que ha enviado todos los segmentos, cierra la conexión con el servidor y libera los recursos ocupados por el socket a través de la primitiva `close()`.

Continuemos con el aprendizaje mediante la revisión código Java de un servidor y un cliente TCP.

3.9.1. Código Java de un servidor TCP

Estudiaremos cómo implementar una comunicación para TCP en lenguaje de programación Java. Haremos el análisis del programa tanto para el servidor como para el cliente. El objetivo de este programa es que el servidor establezca la conexión con el cliente y le retorne un saludo en un mensaje de texto.

En la Figura 22 vemos un ejemplo de un programa para el servidor. Las líneas 1 a 5 importan las clases necesarias para el manejo de excepciones (problemas informados por el sistema operativo), para crear un socket TCP se utiliza la clase `ServerSocket` y para establecer la comunicación con el cliente, la clase `socket`.

Figura 22.

Servidor TCP en Java

```
1 import java.io.DataOutputStream;
2 import java.io.IOException;
3 import java.net.ServerSocket;
4 import java.net.Socket;

5 public class SocketServer {

6     public static final int SERVER_PORT = 65010;

7     public static void main (String[] args){

8         try {
9             ServerSocket server = new ServerSocket(SERVER_PORT);
10            Socket clientConn = server.accept();
11            DataOutputStream serverOutput = new DataOutputStream(clientConn.getOutputStream());
12            serverOutput.writeBytes("Sistemas Distribuidos UTPL\n");
13            clientConn.close();
14        }
15        catch (IOException e) {
16            e.printStackTrace();
17        }
18    }
19 }
```

Nota. Elaborado por el autor

La línea 6 define el número de puerto en el que el servidor levantará el servicio. La línea 9 enlaza y crea el socket. La línea 10 coloca el socket en un estado de espera para una conexión de un cliente. La línea 11 obtiene los parámetros de conexión de un cliente una vez que se ha conectado y lo almacena en variable de tipo *DataOutputStream*. La línea 12 retorna un mensaje de texto al cliente. Finalmente, la línea 13 cierra el socket.

3.9.2. Código Java de un cliente TCP

En la Figura 23 se muestra el código para la comunicación del cliente TCP con el código del servidor analizado anteriormente. Vamos a revisar las líneas de código relevantes.

Las líneas 1 a la 5 han sido explicadas anteriormente y tienen las mismas funciones. La línea 8 genera una conexión con el servidor. La línea 9 almacena la información enviada desde el servidor. Debido a que no se puede pasar directamente a la aplicación (consola) la información recibida,

esta es almacenada en la línea 11. La línea 12 transforma el flujo de datos a caracteres y en la línea 13 se presenta la información en consola.

Figura 23.
Cliente TCP en Java

```
1 import java.io.BufferedReader;
2 import java.io.InputStream;
3 import java.io.InputStreamReader;
4 import java.net.Socket;

5 public class SocketClient {
6     public static void main(String[] args){
7         try {
8             Socket clientSocket = new Socket ("localhost",65010);
9             InputStream is = clientSocket.getInputStream();
10            BufferedReader br = new BufferedReader(new InputStreamReader(is));
11            String receivedData = br.readLine();
12            System.out.println("Información recibida: "+receivedData);
13        }
14        catch (Exception e) {
15            e.printStackTrace();
16        }
17    }
18 }
```

Nota. Elaborado por el autor

Recursos:

En el REA Curso Abierto de Sistemas Distribuidos de la Universidad Carlos III de Madrid (López Fuentes, 2015), específicamente en el tema 5 ([Tema 5 - Comunicación con sockets – \(uc3m.es\)](#)) existe un enfoque similar de esta unidad, invito a hacer un cuadro comparativo de lo analizado de tal forma que se amplíen sus conocimientos.



Actividades de aprendizaje recomendadas

Es importante que en esta semana usted realice las siguientes actividades para consolidar su aprendizaje:

- Leer y familiarizarse con el material didáctico de la asignatura.
 - Lea nuevamente los contenidos de esta guía y cree un mapa conceptual.
 - Consulte el plan docente donde se muestra la secuencia de estudio de los contenidos, así como las actividades que complementan el aprendizaje organizadas por fechas, con su respectiva valoración, rúbricas de evaluación y distribución de las calificaciones por bimestre.
 - El entorno virtual de aprendizaje es el medio de interacción permanente entre docente y estudiante, donde encontrará orientaciones semanales, recursos, herramientas de interacción para con sus compañeros y docente.
- Ingrese al entorno virtual de aprendizaje para conocer las orientaciones de estudio y actividades a desarrollar esta semana; así como para realizar consultas a su docente tutor e interactuar con sus compañeros.
- Lea los conceptos presentados en la sexta semana, complemente los conceptos que allí se presentan con los cursos abiertos de sistemas distribuidos, con estos insumos elabore su propio resumen.
- Para el ejemplo de la librería *Socket de Java*, use diferentes computadores y diferentes direcciones, capture el tráfico de red con un software de análisis como *Wireshark (Foundation, 2016)* y haga un análisis de la diferencia con UDP.
- Consultar cuáles son los subcampos de las banderas de control
- Consultar qué protocolos bien conocidos usan TCP y por qué razón.

Nota: conteste las actividades en un cuaderno de apuntes o en un documento Word.

A continuación, se invita a que revise cómo van sus conocimientos haciendo la respectiva autoevaluación.



Autoevaluación 3

Seleccione la alternativa que considere correcta:

- 1. Los protocolos de comunicación se refieren a:**
 - a. Reglas y formatos.
 - b. Intercambio de mensajes.
 - c. Encapsulación de paquetes.
 - d. Gestión de la sobrecarga de la red.

- 2. ¿Cuál de las siguientes es un ejemplo de protocolo de comunicación?**
 - a. TCP
 - b. HDMI
 - c. Servidor web
 - d. Tarjeta de red

- 3. ¿En qué capas se implementan o desarrollan los sistemas distribuidos?**
 - a. De la capa de red hacia arriba.
 - b. De la capa de transporte hacia arriba.
 - c. En la capa de aplicación.
 - d. De la capa de enlace de datos hacia abajo.

- 4. ¿Cómo se identifica en TCP o UDP a una aplicación que está usando el sistema distribuido?**
 - a. Dirección IP.
 - b. Nombre de dominio.
 - c. Nombre de la aplicación.
 - d. Número de puerto.

- 5. En un sistema distribuido ¿cuál es la opción de comunicación que utiliza directamente las interfaces del sistema operativo?**
- a. Sockets.
 - b. Middleware.
 - c. Invocación remota.
 - d. Comunicación indirecta.
- 6. En un sistema distribuido ¿cuál es la opción de comunicación que desarrolla sus propias interfaces?**
- a. Sockets.
 - b. Middleware.
 - c. Invocación remota.
 - d. Comunicación indirecta.
- 7. En un sistema distribuido ¿cuál es la opción de comunicación que utiliza un equipo intermedio para gestionarla?**
- a. Sockets.
 - b. Middleware.
 - c. Invocación remota.
 - d. Comunicación indirecta.
- 8. ¿En qué capas se implementan los sistemas distribuidos que usan middleware?**
- a. Aplicación y transporte.
 - b. Transporte y red.
 - c. Aplicación y red.
 - d. Transporte, red y enlace de datos.
- 9. Seleccione las alternativas que considere correctas:**
- ¿En TCP / IP cuántos procesos se pueden comunicar al mismo tiempo en un mismo equipo?
- a. 65536
 - b. 12000
 - c. 1024
 - d. 65000

10. Seleccione la alternativa correcta.

¿En TCP / IP los números de puerto bien conocidos están en el rango de?

- a. 1 a 1023
- b. 1 a 1045
- c. 1 a 65000
- d. 1024 a 65000

[Ir al solucionario](#)



Unidad 4. Representación de datos y mensajes multicast

4.1. Representación, transformación y codificación de datos

En sistemas distribuidos, uno de los procesos más relevantes que se debe considerar es la representación adecuada de la información para su uso en un equipo diferente al del emisor. Esto involucra una representación estándar consensuada, una transformación del dato en el origen a la representación estándar y luego una codificación adecuada para enviarlo por la red.

Este proceso es necesario ya que la información recibida puede ser grabada en un almacenamiento secundario, por ejemplo, en el disco duro. Debido a que el hardware y el sistema operativo pueden tener representación para el almacenamiento local diferente al sistema de origen, es necesario que los datos guarden la integridad cuando sean requeridos nuevamente.

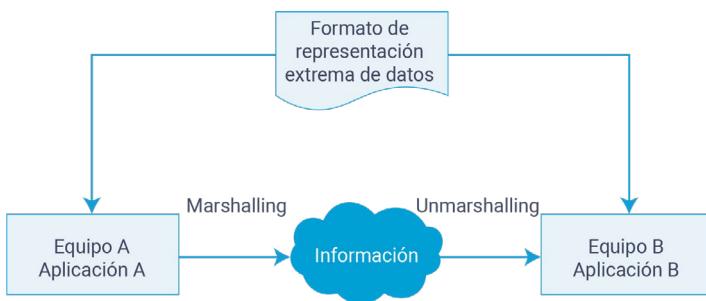
La representación de la información debe afrontar algunos problemas de interoperabilidad, entre los cuales tenemos: el orden en que los bits de los tipos de datos numéricos son almacenados en la memoria principal y secundaria. Algunos equipos pueden almacenar considerando el bit más representativo a la izquierda y otros a la derecha. Imagine que tiene el número 4 en decimal, un equipo puede almacenar en binario como 00000100, mientras que otro puede almacenarlo como 00100000, lo que hará que cuando se recupere la información esta no sea la correcta. Otro problema es la representación del punto flotante en los números reales. Y el último reto tiene que ver con la realidad de que los sistemas diferentes pueden usar diferentes tipos de codificación de caracteres, por ejemplo, ASCII, UTF-8.

Las posibles soluciones que se pueden dar son las siguientes: en primer lugar, los dos lados de la comunicación acuerdan en una representación externa para enviar la información; y, la segunda, es que el emisor transmita la información junto con el tipo de dato y su representación, el receptor toma esta información y la convierte de acuerdo con sus propias características.

En la Figura 24 se muestra cuál es la relación entre el emisor, el receptor y el formato de representación externa de datos. En este caso, los equipos acuerdan un formato de representación de estructuras de datos y primitivas. Luego, el equipo emisor toma los datos, los clasifica, los transforma en la representación externa (marshalling) y los envía a través de la red. El receptor hará con el mismo formato de representación la operación contraria (unmarshalling).

Figura 24.

Representación externa y transformación de datos



Nota. Elaborado por el autor

4.1.1. Iniciativas de representación externa de datos

Algunas compañías y desarrolladores han implementado algunas iniciativas para la representación externa de datos:

1. **CORBA:** relacionada con la representación de datos estructurados (clases) y tipos primitivos (entero, carácter, etc.) que pueden ser pasados como argumentos y resultados en una invocación remota (OMG Object Management Group, 2021).
2. **Java object serialization:** para almacenar y recuperar los objetos de la memoria secundaria en una forma serializada se debe representar el estado de los objetos de tal forma que puedan ser reconstruidos. Los objetos pueden ser almacenados de dos formas por, serialización y externalización. La interfaz de serialización permite recuperar los objetos y sus contenidos. La interfaz de externalización permite simplemente recuperar el formato de los contenidos del objeto (Oracle, 2004).
3. **XML:** el lenguaje de marcado extensible es un formato simple basado en texto de representación de información estructurada (W3C, 2008).

4. **Protocol buffer:** una plataforma propietaria de Google para serializar datos estructurados (Google, 2019). Utiliza un servicio eficiente y mínimo para invocación remota.
5. **JavaScript Object Notation:** JSON es un formato de texto independiente del lenguaje de programación (JSON, 2021).

Es importante aclarar que, al ser una representación externa, estas iniciativas pueden actualizar sus formatos por lo que es necesario que exista una compatibilidad entre versiones. Imagine que el emisor envía el formato con una representación de una estructura de datos con XML versión anterior a la actual, en este caso el emisor debe también enviar la versión del formato que está usando de tal forma que el receptor pueda usar la misma versión.

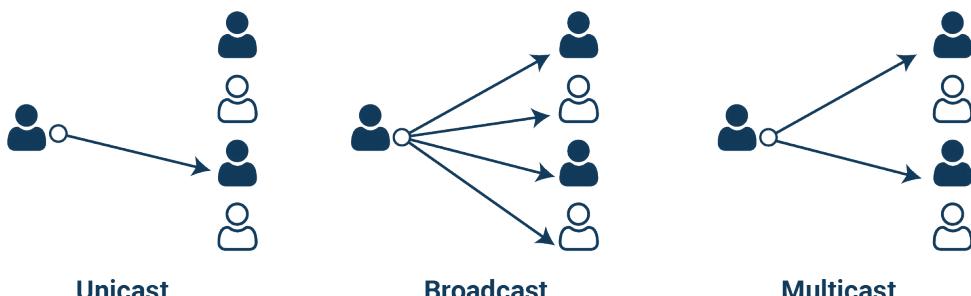
Algunas iniciativas son más completas y extensibles, mientras que otras son más rápidas. Otras usan más bytes para representar información que otras. Por ejemplo, XML al ser basada en texto envía más información que se tarda más en procesar.

4.2. Comunicación multicast

En la Figura 25 se puede determinar la diferencia entre una comunicación unicast, broadcast y multicast. La comunicación unicast es usada por TCP y UDP, por lo general, en un esquema cliente-servidor con una relación de *uno a uno*. La comunicación en broadcast, generalmente, es usada en la capa de enlace de datos y en la capa de red para enviar mensajes a todos los integrantes de la red, la información es enviada en una relación *uno a todos*. Mientras que la comunicación multicast permite que un emisor envíe la misma información a varios destinos que pertenecen a un mismo grupo, en una relación *uno a muchos*.

Figura 25.

Tipos de comunicación



Nota. Elaborado por el autor

Los mensajes multicast brindan una infraestructura altamente usada para la construcción de sistemas distribuidos, especialmente, para las siguientes tareas:

1. Tolerancia a fallos basado en servicios replicados.
2. Descubrimiento de servicios en redes de computadoras: usados para determinar, entre otras cosas, qué equipo o grupo de equipos en la red tiene asignada una función especial, un ejemplo es determinar cuál es el enrutador de último salto en una red.
3. Mejorar el rendimiento a través de servicios replicados.
4. Propagación de notificaciones de eventos.

Los mensajes multicast tienen una menor carga computacional en los equipos finales y generan un menor tráfico en la red que unicast, cuando los equipos finales emisor y receptor trabajan en la misma red. En multicast, el equipo emisor solo genera un paquete que entrega a la red y, como en la red solo circula un paquete, la carga en la red es menor.

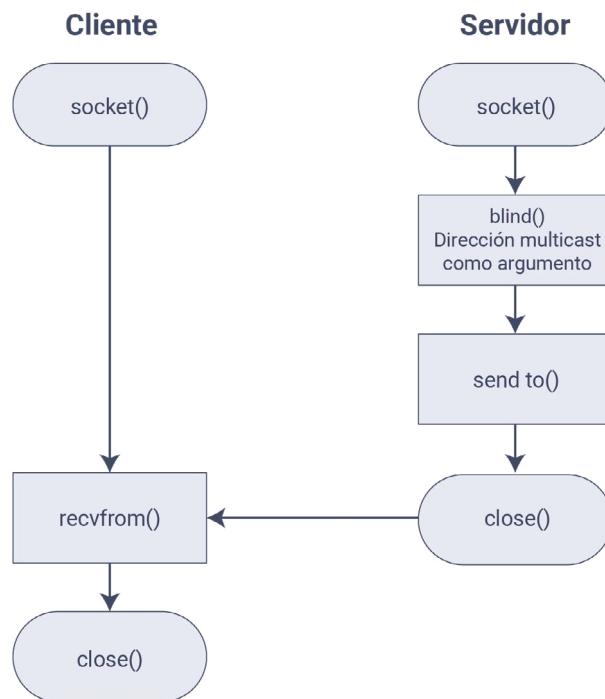
El uso de multicast en el núcleo de la red involucra que los enrutadores o equipos intermedios tengan la capacidad de reenviar mensajes multicast, por lo general, los fabricantes no habilitan esta función por defecto. En el núcleo de la red, el uso de multicast genera más tráfico debido a que el enrutador debe replicar el paquete original y enviarlo a todas las interfaces de salida seleccionadas. La aplicación puede limitar la distancia de operación o la cobertura de los mensajes de multicast configurando un valor bajo en el campo *Time To Live (TTL)* del paquete IP.

Hay que recalcar que multicast trabaja a nivel de la capa de red, es decir, que da los servicios a la capa de transporte, opera únicamente con UDP. Por lo general, los sistemas distribuidos que usan multicast deben implementar sus propias características de transporte confiable. Además, deben considerar que la retroalimentación de los clientes al sistema distribuido se la hace tanto con una comunicación unicast como multicast.

Para poder usar multicast la aplicación debe usar direcciones especiales para definir los grupos y el destino de la información. En TCP/IP para la versión de Ipv4, las direcciones multicast están definidas por la clase D de direcciones de Internet que van en el rango de 224.0.0.0 hasta 239.255.255.255. En IPv6, las direcciones en multicast empiezan con el valor FF en hexadecimal.

La Figura 26 muestra un diagrama de flujo e interacción entre un servidor y un cliente multicast. Se analiza las primitivas más importantes para la comunicación.

Figura 26.
Interacción multicast



Nota. Elaborado por el autor

En el **servidor** se debe generar información enviada hacia un grupo específico a través de la primitiva `bind()` en la cual se debe colocar la dirección IP multicast. Luego, con la primitiva `sendto()` se envía el mensaje multicast a todos los clientes que pertenezcan al grupo, los clientes tienen una aplicación definida en la cual están escuchando en la dirección multicast y el puerto preestablecido.

El **cliente**, una vez que levanta el socket, recibe información del servidor. Como se puede apreciar la comunicación multicast es unidireccional. En este ejemplo la interacción del cliente representa a un grupo.

A continuación, se analizará el código Java de un servidor multicast y para clientes multicast:

4.2.1. Código Java de un servidor multicast

En la Figura 27 se muestra el código para la comunicación de un equipo enviando mensajes a un grupo multicast. En el ejemplo, el servidor envía mensajes a la dirección multicast 231.0.0.0 y al puerto 4321 (Developer, 2021).

Se revisa las líneas de código relevantes.

Figura 27.

Servidor multicast en Java

```
1 import java.io.IOException;
2 import java.net DatagramPacket;
3 import java.net DatagramSocket;
4 import java.net InetSocketAddress;
5
6 public class UDPMulticastServer {
7
8     public static void sendUDPMessages(String message,
9             String ipAddress, int port) throws IOException {
10        DatagramSocket socket = new DatagramSocket();
11        InetSocketAddress group = InetSocketAddress.getByName(ipAddress);
12        byte[] msg = message.getBytes();
13        DatagramPacket packet = new DatagramPacket(msg, msg.length, group, port);
14        socket.send(packet);
15        socket.close();
16    }
17
18    public static void main(String[] args) throws IOException {
19        sendUDPMessages("Primer mensaje multicast", "230.0.0.0", 4321);
20        sendUDPMessages("Segundo mensaje multicast", "230.0.0.0", 4321);
21        sendUDPMessages("Final", "230.0.0.0", 4321);
22    }
23}
```

Nota. Elaborado por el autor

Las líneas 1 a la 4 importan las clases para el manejo de excepciones y direccionamiento IP y la creación de paquetes UDP y sockets UDP respectivamente. La línea 7 permite la instanciación del socket. La línea 9 instancia un objeto para el manejo de las direcciones multicast. La línea 10 genera un datagrama para enviar en la línea 11 a través del socket UDP. Como se puede ver, se envían tres paquetes multicast. La línea 11 finalmente cierra la conexión y libera los recursos.

4.2.2. Código Java para clientes multicast

En la Figura 28 se muestra el código para la recepción de información multicast en Java. El mismo código se ejecuta para todos los clientes que pertenezcan al grupo. En el ejemplo, toma los mensajes enviados por el servidor a la dirección multicast 231.0.0.0 y al puerto 4321, y los presenta en consola. A continuación, se revisa las líneas de código relevantes.

Figura 28.

Cliente multicast en Java

```
1 import java.io.IOException;
2 import java.net.DatagramPacket;
3 import java.net.InetAddress;
4 import java.net.MulticastSocket;
5
6 public class UDPMulticastClient implements Runnable {
7
8     public static void main(String[] args) {
9         Thread t = new Thread(new UDPMulticastClient());
10        t.start();
11    }
12
13    public void receiveUDPMessages(String ip, int port) throws
14        IOException {
15        byte[] buffer = new byte[1024];
16        MulticastSocket socket = new MulticastSocket(4321);
17        InetAddress group = InetAddress.getByName("230.0.0.0");
18        socket.joinGroup(group);
19        while(true) {
20            System.out.println("Esperando mensajes multicast");
21            DatagramPacket packet = new DatagramPacket(buffer, buffer.length);
22            socket.receive(packet);
23            String msg = new String(packet.getData(),
24                packet.getOffset(), packet.getLength());
25            System.out.println("Mensaje multicast UDP recibido: >> " +msg);
26            if("Final".equals(msg)) {
27                System.out.println("No más mensajes. Cerrando conexión : " +msg);
28                break;
29            }
30        }
31    public void run() {
32        try {
33            receiveUDPMessages("230.0.0.0", 4321);
34        }catch(IOException ex) {
35            ex.printStackTrace();
36        }
37    }
}
```

Nota. Elaborado por el autor

Las líneas 1 a la 4 importan las librerías para excepciones, el manejo de direcciones multicast y la generación de un socket multicast respectivamente.

La línea 5 implementa la clase multicast cliente con la interfaz Java *Runnable*. Esta interfaz es usada para ejecutar código en un hilo en forma concurrente, tiene un método no definido *run()*, líneas 31 a la 36, que no retorna ningún valor ni tampoco usa argumentos.

La línea 13 define un número de puertos para luego crear un socket multicast, y en la línea 14 se especifica la dirección multicast, el socket con los argumentos definidos anteriormente es instanciado en la línea 15. La línea 18 y 19 generan una estructura de datagrama para recibir los mensajes multicast enviados por el servidor. La línea 22 muestra en consola el mensaje recibido. La línea 28 libera a la aplicación de la pertenencia al grupo multicast y la línea 29 cierra el socket.

Recursos:

- Curso Abierto de Sistemas Distribuidos, en inglés, del Massachussets International Institute (Robert, 2006).
- Curso Abierto de Sistemas Distribuidos de la Universidad Carlos III de Madrid (López Fuentes, 2015).
- Curso Abierto de Sistemas Distribuidos de la Atlantic International University (Atlantic International University, 2021).

En el documentación de Google, protocolo buffer ([Protocol Buffers | Google Developers](#)), existe el ejemplo para la estructura person, invito a hacer un análisis de la técnica de marshaling propuesta en el sitio, de tal forma que se amplíen sus conocimientos.



Actividades de aprendizaje recomendadas

Es importante que en esta semana usted realice las siguientes actividades para consolidar su aprendizaje:

- Leer y familiarizarse con el material didáctico de la asignatura.
 - Lea nuevamente los contenidos de esta guía y cree un mapa conceptual.

- Consulte el plan docente donde se muestra la secuencia de estudio de los contenidos, así como las actividades que complementan el aprendizaje organizadas por fechas, con su respectiva valoración, rúbricas de evaluación y distribución de las calificaciones por bimestre.
- El entorno virtual de aprendizaje es el medio de interacción permanente entre docente y estudiante, donde encontrará orientaciones semanales, recursos, herramientas de interacción para con sus compañeros y docente.
- Ingrese al entorno virtual de aprendizaje para conocer las orientaciones de estudio y actividades a desarrollar esta semana; así como para realizar consultas a su docente tutor e interactuar con sus compañeros.
- Lea los conceptos presentados en la séptima semana, complemente los conceptos que allí se presentan con los cursos abiertos de sistemas distribuidos, con estos insumos elabore su propio resumen.
- Para el ejemplo de la librería *MulticastSocket* de Java, use al menos tres diferentes computadores y diferentes direcciones, capture el tráfico de red con un software de análisis como *Wireshark* (*Foundation, 2016*).

Nota: conteste las actividades en un cuaderno de apuntes o en un documento Word.

A continuación, se invita a que revise cómo van sus conocimientos haciendo la respectiva autoevaluación.



Autoevaluación 4

- 1. ¿Por qué es importante una representación adecuada de datos en sistemas distribuidos?**
 - a. Por la heterogeneidad de los sistemas operativos.
 - b. Porque la información puede viajar por diferentes ISP.
 - c. Porque la memoria del emisor puede ser diferente al del receptor.
 - d. Por la cantidad de procesos que se ejecutan en el sistema distribuido.
- 2. El orden de almacenamiento de los datos numéricos hace necesario que se requiera una representación estándar de datos.**
 - a. Verdadero
 - b. Falso
- 3. Si almaceno el número 12 en un byte con el bit más representativo a la izquierda ¿cuál es el valor en binario?**
 - a. 00001100
 - b. 00001010
 - c. 00110000
 - d. 01010000
- 4. ¿Por qué es importante una representación estándar de datos en sistemas distribuidos?**
 - a. Por la cantidad de procesos que se ejecutan en el sistema distribuido.
 - b. Porque la información puede viajar por diferentes ISP.
 - c. Diferentes sistemas pueden usar diferentes tipos de codificación de caracteres.
 - d. Porque la memoria del emisor puede ser diferente al del receptor.

- 5. Seleccione una de las posibles soluciones al problema de representación de datos en sistemas distribuidos.**
- a. Cambiar todos los datos a un tipo de dato único.
 - b. Eliminar los tipos de datos y usar solo caracteres.
 - c. El emisor transmite la información junto con el tipo de dato y su representación.
 - d. Enviar la información como un flujo de bits sin límites.
- 6. ¿Cuáles de las siguientes son iniciativas de representación externa de datos estructurados y tipos primitivos?**
- a. CORBA
 - b. XML
 - c. TCP
 - d. ASCII
- 7. ¿Cuáles de las siguientes son iniciativas de representación externa para el paradigma orientado a objetos?**
- a. CORBA.
 - b. XML.
 - c. Java Object serialization.
 - d. ASCII.
- 8. ¿Qué relación existe en una comunicación unicast?**
- a. Uno a uno.
 - b. Uno o todos.
 - c. Todos a todos.
 - d. Uno a muchos.
- 9. ¿Qué relación existe en una comunicación multicast?**
- a. Uno a uno.
 - b. Uno o todos.
 - c. Todos a todos.
 - d. Uno a muchos.

- 10. Los enrutadores en la red de núcleo deben tener la capacidad y ser configurados para permitir el paso de mensajes multicast.**
- a. Verdadero
 - b. Falso

[Ir al solucionario](#)



Actividades finales del bimestre



Semana 8

En esta última semana del primer bimestre, es importante que realice las siguientes actividades para consolidar su aprendizaje:

- Ingrese al entorno virtual de aprendizaje para conocer las orientaciones de estudio y actividades a desarrollar esta semana; así como para realizar consultas a su docente tutor e interactuar con sus compañeros.
- Revise los contenidos abordados en el primer bimestre de la asignatura y sus anotaciones, resúmenes, síntesis respecto a los mismos como preparación a la evaluación del primer bimestre.
- Participe de la actividad suplementaria.
- Revise las autoevaluaciones y cuestionarios de las unidades 1, 2, 3 y 4 para recordar y reforzar los contenidos que allí se abordaron.



Segundo bimestre

Resultado de aprendizaje 3 y 4

- Diseñar e implementar una red de varios equipos que ofrece servicios de computación en esa red.
- Diseñar un conjunto de servidores para una situación dada.

En esta unidad se abordará los conceptos de la representación y las características de los sistemas distribuidos, para luego determinar cuáles son las mejores arquitecturas para un diseño de aplicaciones con capacidad distribuida. La falta de un reloj global y algunas soluciones mayormente aceptadas son analizadas y explicadas en esta unidad.

Como actividades de aprendizaje recomendadas, se invita a que se implementen los algoritmos para soluciones de relojes lógicos que son parte de esta unidad.

En esta unidad se mostrarán los diferentes conceptos, algoritmos y componentes de los sistemas distribuidos, y cómo se relacionan entre sí con el ánimo de mostrar una forma ordenada y clara para modelar adecuadamente aplicaciones que usen sistemas distribuidos.

Con estos resultados de aprendizaje tendrá la capacidad de reconocer los componentes de los sistemas distribuidos y los algoritmos necesarios para poder tener una noción de tiempo conjunta que permite la interacción de los diferentes componentes en un sistema distribuido. Con esta información podrá diseñar un sistema distribuido considerando las posibles fallas que puedan ocurrir.

Contenidos, recursos y actividades de aprendizaje recomendadas



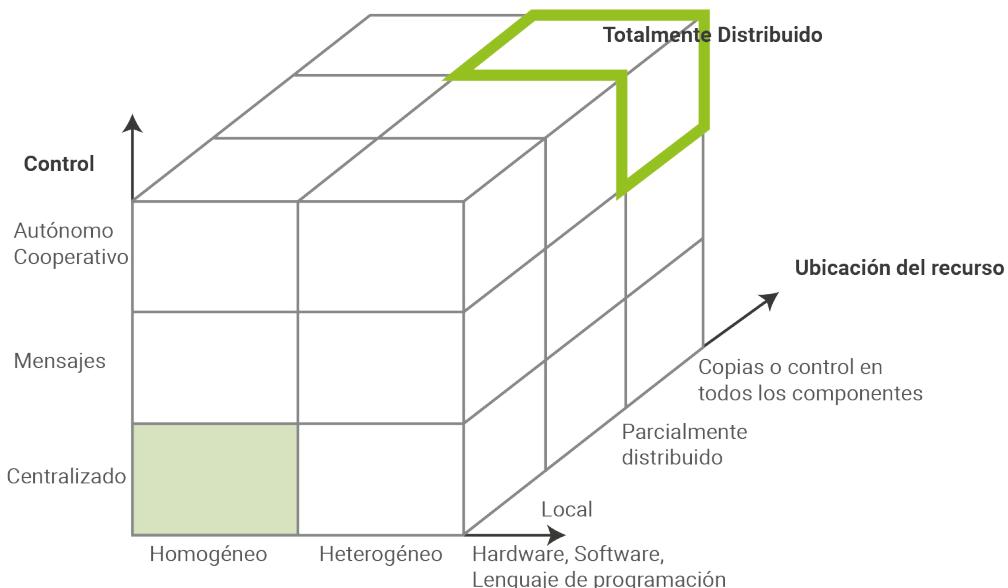
Unidad 5. Diseño y representación de sistemas distribuidos

5.1. Sistemas distribuidos y sistemas centralizados

Un sistema distribuido es donde los procesos ubicados en dispositivos electrónicos intercambian información a través de una red computadores. Los procesos se comunican entre sí a través del intercambio de mensajes. Los sistemas distribuidos carecen de un reloj global que permita sincronizar sus tareas y los componentes pueden fallar de forma independiente, afectando a la salud de todo el sistema.

Figura 29.

Heterogeneidad, control y localización en sistemas distribuidos



Nota. Elaborado por el autor

En la Figura 29 se puede ver la relación entre el control del sistema, la homogeneidad del hardware y software, y la ubicación del recurso compartido. Un sistema centralizado en la figura está ubicado en la esquina inferior, los recursos están en un solo componente, el control es centralizado y se tendrá una homogeneidad total porque se usará el mismo

sistema operativo, el mismo hardware y, posiblemente, el mismo lenguaje de programación para desarrollar las aplicaciones. En la esquina superior se encuentran los sistemas totalmente distribuidos en los cuales el control puede ser autónomo y los componentes son totalmente cooperativos entre sí, los recursos compartidos se encuentran repartidos en los componentes del sistema y es posible que exista una alta heterogeneidad tanto en hardware, sistema operativo, lenguajes de programación y estilos de programación.

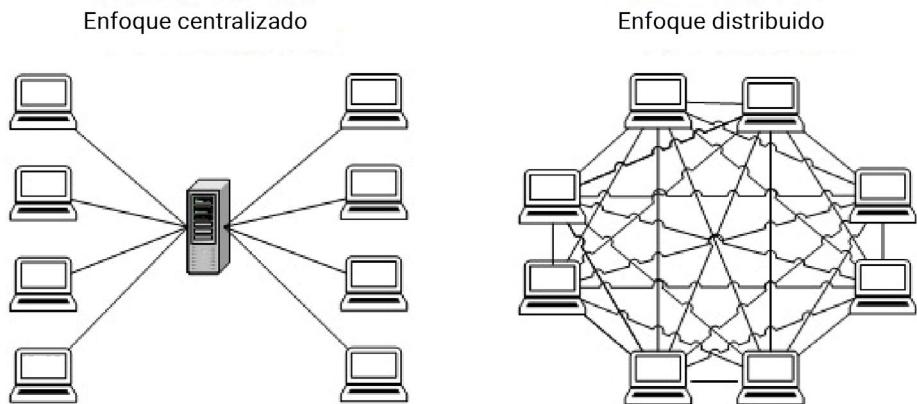
En un sistema distribuido existen múltiples componentes que trabajan en forma autónoma, los componentes son compartidos, pero no necesariamente con todos los usuarios. El software se ejecuta en procesos concurrentes en diferentes procesos. Por su naturaleza existen varios puntos de control y múltiples puntos de falla. La gestión de un sistema distribuido es más compleja que en un sistema centralizado.

A su vez, en un sistema centralizado existe un componente (sistema central o un equipo) que permite gestionar todo el sistema. Este componente es compartido por todos los usuarios, una de las ventajas es que existe un único punto de control lo que facilita su gestión. El componente central maneja el tiempo de las actividades del sistema porque se puede sincronizar de mejor manera. Sin embargo, este sistema presenta un simple punto de falla de tal forma que, si este equipo es vulnerado, todo el sistema falla.

En la Figura 30, enfoques distribuidos y centralizados, se puede ver la diferencia entre un sistema distribuido y un sistema centralizado, especialmente útil para determinar la responsabilidad centralizada y la poca escalabilidad del sistema centralizado.

Figura 30.

Enfoques distribuidos y centralizados



Nota. Elaborado por el autor

En la Tabla 2, sistemas distribuidos y centralizados, se puede observar las principales diferencias entre un sistema distribuido y un sistema centralizado.

Tabla 2.

Sistemas distribuidos y centralizados

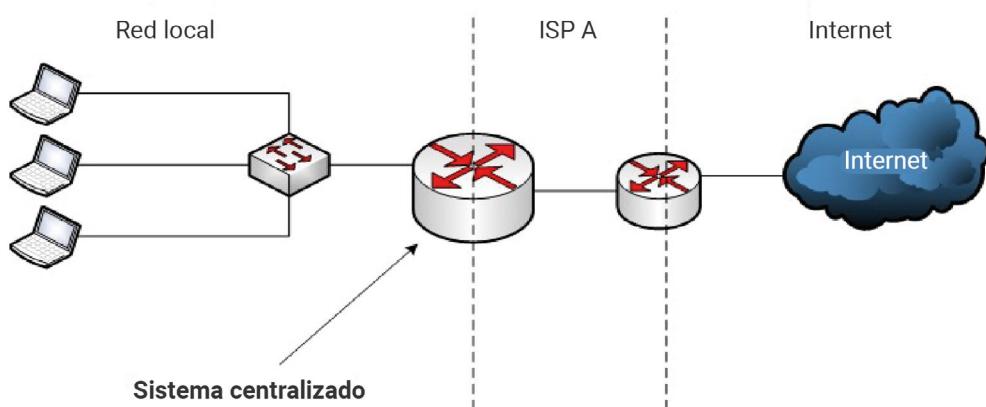
	Distribuido	Centralizado
Escalabilidad	Alta	Baja
Seguridad	Baja	Alta
Gestión de fallas	Compleja	Fácil
Reloj global	Gestionado por el componente central.	Los componentes se deben sincronizar a través del intercambio de mensajes.
Fallas	Dispersas entre los componentes.	Un punto único de fallos en el componente central.

Nota. Elaborado por el autor

La Figura 31 muestra un ejemplo de un sistema centralizado de acceso a Internet a una red local, el servicio en la red local está dado por un solo enrutador que se conecta a un solo ISP, la información de rutas para acceder a otras redes se concentra en el enrutador. En este caso, es relativamente fácil colocar en el enrutador políticas de acceso a la red determinando qué equipos pueden tener servicio y cuáles no. Sin embargo,

si este enrutador falla, toda la organización no puede tener el servicio de Internet.

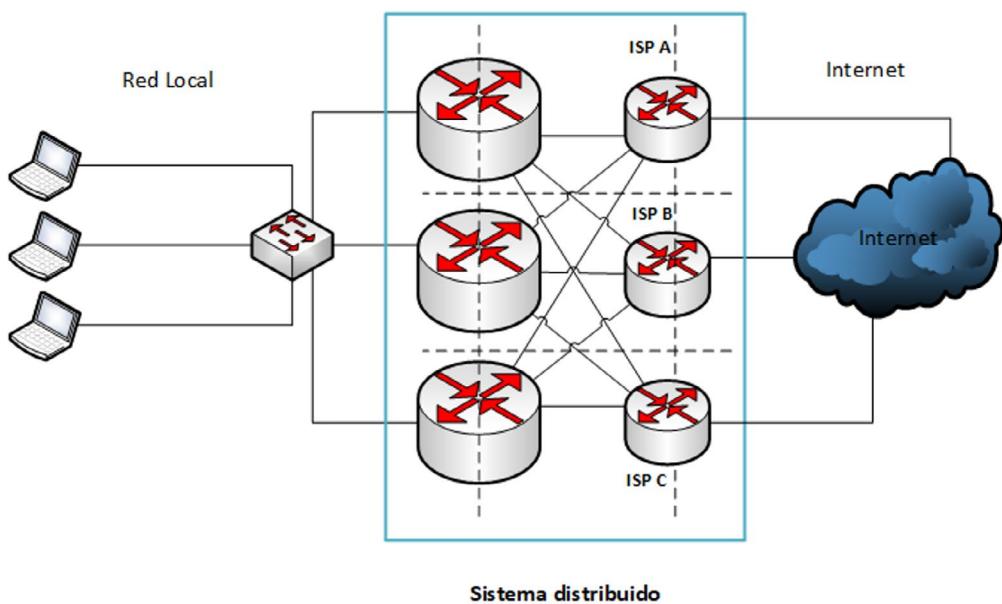
Figura 31.
Acceso a Internet centralizado



Nota. Elaborado por el autor

Por otro lado, en la Figura 32 consta un ejemplo de un sistema distribuido de acceso a Internet. El sistema distribuido está compuesto por 6 enrutadores, 3 de ellos pertenecen a la organización y podrían tener el mismo hardware y software. Los enrutadores del ISP posiblemente serán diferentes en características que los de la organización, por lo que mantener el servicio requerirá un mayor conocimiento técnico y un acuerdo entre los ISP y la organización. La información y las tablas de rutas, para brindar el servicio de Internet, estarán almacenados en forma local en cada enrutador. Si un enrutador o enlace falla, el sistema distribuido a través del intercambio de mensajes debe garantizar el servicio de Internet. Si un enrutador falla, el servicio de Internet se mantiene. La información de rutas requiere un protocolo de enrutamiento para mantenerla actualizada, este protocolo intercambia información del estado del servicio (sistema distribuido) a través de mensajes.

Figura 32.
Sistema distribuido para acceso a Internet



Nota. Elaborado por el autor

Del ejemplo anterior se puede apreciar que un sistema distribuido tiene más variables a considerar que un sistema centralizado, lo que lo hace más complejo. Además, el costo de implementación, configuración y el costo computacional será mayor. Sin embargo, el servicio de acceso a Internet está garantizado y con mejores prestaciones.

Otro ejemplo de sistema distribuido es el de un banco, donde el recurso compartido es una cuenta de ahorro cuyo contenido está replicado (existen copias) en varias bases de datos distribuidos. Para hacer una transacción existen varios canales de comunicación, cajero automático, banca virtual, celular, etc. Por cualquiera de estos canales se puede depositar o retirar. Imagine que tiene una cuenta con 10 dólares de saldo y van a realizar dos transacciones, un depósito de 15 dólares desde la banca virtual y un retiro de 20 dólares desde un cajero automático. Como se puede ver, aquí existe una relación causal obligatoria, primero se debe depositar y luego retirar. Suponga que a las 12H00 hace el depósito, un segundo después realiza la transacción del retiro desde el mismo equipo, los cambios en la cuenta de ahorros deberían ser reflejados en forma consistente en todas las bases de datos. Sin embargo, mientras se hace el depósito un enrutador en la red de núcleo falla y causa que la transacción se demore dos segundos.

En este caso el retiro llegó antes. El sistema distribuido debe considerar todos estos posibles escenarios para manejar la información en forma consistente y confiable, por lo tanto, debe implementar estrategias como, por ejemplo, un reloj global, un modelo de gestión de fallos, un modelo sincronización y consenso, etc.

5.2. Requerimientos de diseño para sistemas distribuidos

Los principios generales de Ingeniería de software incluyen rigor y formalidad, modularidad, abstracción, anticipación al cambio, entre otros. Cuando se diseña sistemas distribuidos, además los anteriores, se debe considerar los siguientes:

- a. Denominación de componentes.
- b. Comunicación de componentes.
- c. Estructura del software.
- d. Arquitectura del sistema.

Ahora, analizaremos cada uno de ellos.

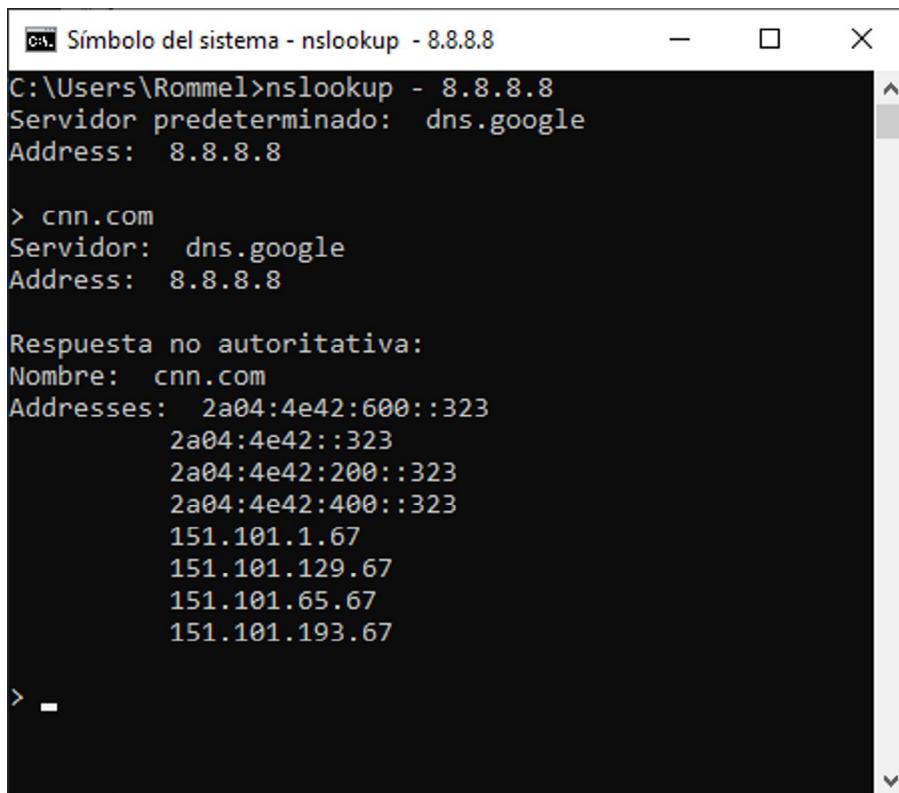
5.2.1. Denominación de componentes

La denominación de componentes, en inglés *naming*, es el proceso de traducir una referencia de un objeto o recurso a una forma interpretable. Permite, a los usuarios o programas, asignar nombres en formato de cadena de caracteres a los objetos o recursos y usar estos nombres a continuación para referirse a estos objetos. La denominación debe permitir también que el objeto sea localizable en el sistema distribuido. Por lo tanto, son necesarias dos instancias para tener un sistema de denominación funcional: la localización y la denominación.

Un sistema de denominación debe proveer un nivel de abstracción para las réplicas (copias de recursos), para su uso en el sistema distribuido. Cuando se solicita un recurso, el sistema de denominación debe retornar la localización de las réplicas.

Uno de los sistemas de denominación más conocidos es DNS que permite, dado un nombre, obtener las direcciones IP de donde se encuentra el recurso.

Figura 33.
Consulta de DNS



The screenshot shows a terminal window titled "Símbolo del sistema - nslookup - 8.8.8.8". The command entered is "nslookup - 8.8.8.8". The output shows the server is dns.google and the address is 8.8.8.8. A query is then made for "cnn.com", which also uses dns.google as the server and has an address of 8.8.8.8. The response is non-authoritative, listing eight IP addresses: 2a04:4e42:600::323, 2a04:4e42::323, 2a04:4e42:200::323, 2a04:4e42:400::323, 151.101.1.67, 151.101.129.67, 151.101.65.67, and 151.101.193.67. The command ">> -" is shown at the bottom.

Nota. Elaborado por el autor

En la Figura 33 se puede ver una consulta de denominación de DNS. Con la herramienta *nslookup* se hace una consulta sobre el nombre [cnn.com](#) al servidor 8.8.8.8 el cual responde con las direcciones IP en la cual se puede encontrar el recurso. En este caso, tenemos ocho servidores en los cuales está alojado el dominio.

DNS utiliza una denominación de alto nivel, orientado a los usuarios, significa que los nombres tienen significado y son fáciles entender. En sistemas distribuidos se usa también un nivel bajo de denominación, orientado a los sistemas, donde los nombres de los dispositivos o entidades pueden ser de longitud fija para que puedan ser usados y almacenados fácilmente por los sistemas. Los nombres son generados automáticamente por el sistema en forma distribuida para no depender de un único dispositivo centralizado. Debe existir una correspondencia entre el nombre de alto con el de bajo nivel, y la correspondencia adecuada entre

el nombre de bajo nivel y la localización y denominación de los objetos o recursos.

5.2.2. Comunicación de componentes

En un sistema distribuido, los componentes separados se comunican con procesos que envían y reciben información para realizar dos tareas fundamentales, transferencia de información y sincronización.

Este intercambio de mensajes se lo realiza a través de primitivas de enviar y recibir. Sin embargo, estas primitivas pueden bloquearse de acuerdo a la forma de comunicación. Cuando los procesos se comunican en forma síncrona, por lo general, las primitivas son bloqueantes, mientras que si es totalmente asíncrono será no-bloqueante.

Este intercambio se lo hace a través de patrones de comunicación cliente-servidor y comunicación en grupo a través de multicast, aquí se incluye la comunicación peer-to-peer.

Cuando se diseña sistemas distribuidos, se debe considerar cómo lograr que las réplicas tengan consistencia posiblemente emulando un sistema síncrono, considerando las características de los patrones de comunicación.

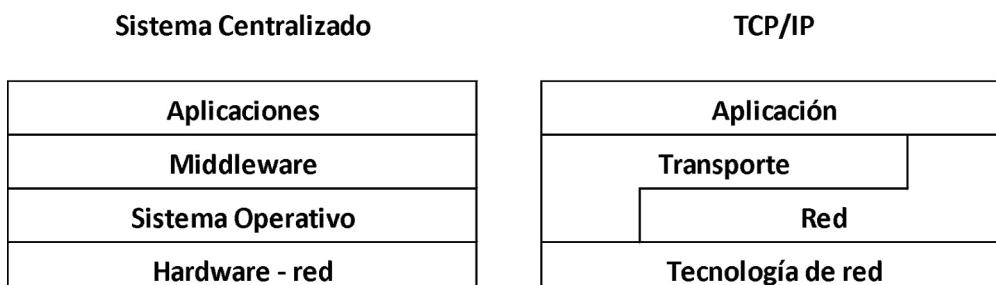
5.2.3. Estructura del software

Cuando se diseña sistemas distribuidos, se debe considerar cómo estructurar el software; una abstracción por capas es utilizada para diferenciar las responsabilidades del hardware, sistema operativo, aplicación y sus respectivas interfaces.

En la Figura 34 se ve cuáles son las capas en un sistema centralizado y la relación con el modelo de referencia TCP/IP. Una capa de middleware en un sistema centralizado usa las interfaces que debe dar un sistema operativo y, en relación con el modelo de capas de TCP/IP, están en la capa de aplicación y la capa de transporte.

Figura 34.

Relación TCP/IP y un sistema centralizado



Nota. Elaborado por el autor

A su vez, puede ver en la Figura 35 que la capacidad de generar un sistema distribuido está en la capa middleware, sin embargo, depende al mismo tiempo de los servicios que brinda el sistema operativo. Para las aplicaciones es transparente el trabajo del sistema distribuido que se da en la capa de middleware.

Figura 35.

Capas en un sistema distribuido



Nota. Elaborado por el autor

5.2.4. Arquitectura del sistema

La arquitectura del sistema se refiere a dónde están localizados los procesos, cómo interactúan entre sí en todo el sistema, cómo se comunican, cuáles son las responsabilidades de cada uno. Se puede definir las siguientes arquitecturas del sistema:

1. Ubicación de recursos: cliente-servidor, peer to peer.
2. Granja de servidores.
3. Comunicación indirecta: Web proxy.

4. Movilidad de recursos: código móvil o agentes móviles, clientes móviles.
5. Poca capacidad de computación.

Cuando se diseña un sistema distribuido, se debe tener en cuenta qué tipo de arquitectura podemos usar o cuál está disponible. Por ejemplo, un sistema cliente-servidor es mejor cuando tenemos pocos usuarios en el sistema y la posibilidad de crecimiento del sistema es limitada. Cuando queremos mejorar los tiempos de respuesta, por ejemplo, y los datos son de un tamaño considerable, sería recomendable tomar en cuenta las características de un sistema *peer to peer*. Si por otro lado los recursos compartidos son relativamente estáticos y no cambian en el tiempo una opción para mejorar los tiempos de respuesta, podría ser usar un sistema de comunicación indirecta, como por ejemplo, un web proxy. Si, a su vez, estos recursos son estáticos y el acceso requerido viene desde diferentes usuarios geográficamente separados, se puede considerar el uso de un sistema de nombres de dominio y una granja de servidores localizados de acuerdo a la ubicación de los clientes.

Un caso especial se da cuando se considera un código móvil o un programa que se almacena en el cliente para ser ejecutado cuando se interactúa con el sistema. Inclusive el código móvil puede ejecutarse en forma autónoma.

Otro caso especial es cuando la capacidad de movimiento está asociado al recurso, por ejemplo, un dispositivo móvil puede desplazarse y cambiar su ubicación, y ,por lo tanto, su acceso a la red puede verse afectado impactando directamente en la disponibilidad del recurso. Inclusive el recurso como tal puede migrar de un equipo a otro, en este caso el diseño debería incluir estrategias para mantener la integridad de los recursos.

Finalmente, están los sistemas distribuidos en el cual todo el procesamiento se encuentra centralizado debido a que los clientes tienen muy poca capacidad de procesamiento y únicamente son usados como medios de entrada y salida de información.



Actividades de aprendizaje recomendadas

Es importante que en esta semana usted realice las siguientes actividades para consolidar su aprendizaje:

- Leer y familiarizarse con el material didáctico de la asignatura.
 - Lea nuevamente los contenidos de esta guía y cree un mapa conceptual.
 - Consulte el plan docente donde se muestra la secuencia de estudio de los contenidos, así como las actividades que complementan el aprendizaje organizadas por fechas, con su respectiva valoración, rúbricas de evaluación y distribución de las calificaciones por bimestre.
 - El entorno virtual de aprendizaje es el medio de interacción permanente entre docente y estudiante, donde encontrará orientaciones semanales, recursos, herramientas de interacción para con sus compañeros y docente.
- Ingrese al entorno virtual de aprendizaje para conocer las orientaciones de estudio y actividades a desarrollar esta semana; así como para realizar consultas a su docente tutor e interactuar con sus compañeros.
- Busque al menos un sistema distribuido y un centralizado que se desarrollen en la vida real y, de acuerdo a los criterios de la tabla 2, haga su análisis y valoración.
- Práctica de *nslookup* desde su computador.

Nota: conteste las actividades en un cuaderno de apuntes o en un documento Word.



5.3. Tiempo y estado global en sistemas distribuidos

El tiempo global significa que todo el sistema distribuido debe tener una conceptualización del momento en que los procesos deben ejecutarse. Esto es un reto para los sistemas distribuidos asíncronos que no pueden determinar exactamente en qué momento los procesos se pueden llevar a cabo, sin embargo, se pueden aproximar usando rangos de tiempo mínimo y máximo. El tiempo mínimo se puede calcular ya que es el tiempo para mejor escenario (tiempo de generación, tiempo de despacho y tiempo de transferencia en la red). El tiempo máximo es aquel que se puede aproximar o proyectar, por ejemplo, el protocolo TCP utiliza como base el RTT, tiempo de ida y vuelta del último segmento, más un tiempo de guarda que está basado en el histórico del RTT. Los sistemas distribuidos asíncronos son un conjunto de procesos que no tienen una memoria de estado ni de tiempo compartida y se comunican entre sí a través de mensajes los cuales tienen diferentes tiempos de latencia.

Esto significa que para tener un estado global y un tiempo global deben intercambiar mensajes, y deben sincronizar su estado y tiempo considerando los tiempos de latencia.

El tiempo y el estado global es difícil de acordar en un sistema distribuido, más aún porque la tasa de eventos ejecutándose es muy alta y el tiempo de procesamiento de cada evento es muy pequeño. Los tiempos de transferencia suelen ser más altos que la ejecución del evento.

Lo que se debe hacer es aproximar una vista global (estado y tiempo del sistema distribuido) a través de tres estrategias:

- Simular un sistema asíncrono como síncrono, tratar de determinar el tiempo en el que un evento debe ser procesado.
- Simular un tiempo global, a través de mensajes para sincronizar los relojes físicos de los dispositivos y generar relojes lógicos para todo el sistema distribuido.

- Simular un estado global del sistema, a través del intercambio de mensajes determinar cuál es el estado de todo el sistema distribuido en cierto intervalo de tiempo. Es similar a tomar una foto de cómo están los procesos del sistema distribuido en un momento dado, cuántos procesos han fallado o cuántos procesos están funcionando, cuál es el estado de un recurso compartido y qué proceso, en un momento dado, está modificando o leyéndolo.

5.4. Tiempo global en sistemas distribuidos

Una noción precisa del tiempo en un sistema distribuido es una tarea muy difícil de lograr, sin embargo, una aproximación de tiempo es necesaria, especialmente en sistemas distribuidos de aplicación crítica, por ejemplo, una planta nuclear o juegos en línea.

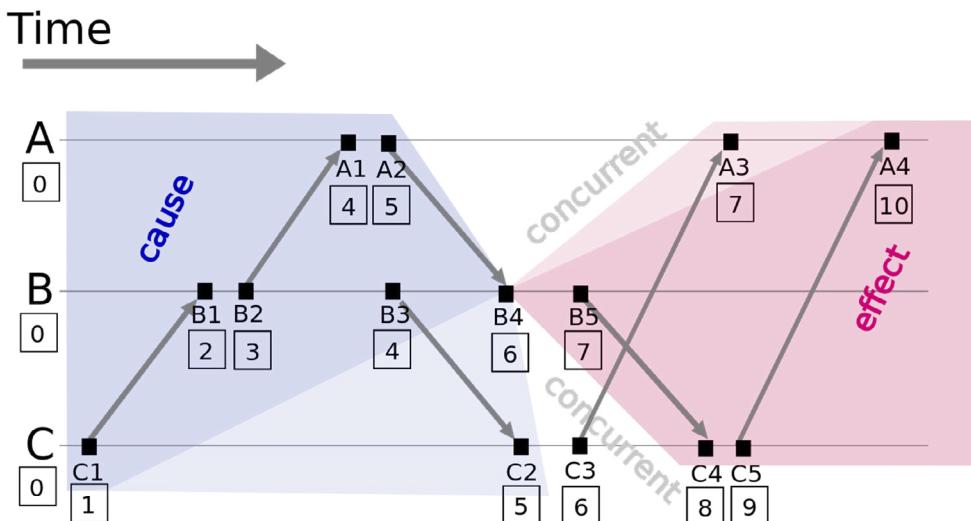
Los relojes en un sistema se desfasan o se desigualan, de un componente a otro, o de un componente con respecto a un reloj global; tratar de sincronizar con un reloj global es una de las tareas más demandantes de un sistema y podría ser un reto en su implementación.

Es, por lo tanto, requerido que exista una sincronización de relojes para simular un reloj global del sistema. Debe existir un mecanismo de sincronización entre un reloj físico y el reloj lógico. La sincronización de relojes lógicos deriva en un concepto importante en el sistema distribuido denominado causalidad.

La causalidad de eventos permite tener integridad en un sistema distribuido. Piense en un sistema multijugador en línea, cada jugador es un componente del sistema que maneja sus propios procesos, sin embargo, todo el sistema debe estar sincronizado. No es posible que el avatar de un jugador sea impactado por una bala antes de que otro jugador genere el proceso apuntar-disparar.

Figura 36.

Reloj lógico de Lamport



Nota. Tomado de Leslie Lamport (2021).

En la Figura 36 está un ejemplo clásico de la literatura para definir un reloj lógico y la relación de causalidad (*happened – before*). Existen tres procesos A, B, y C, cada proceso tiene eventos o actividades A1, A2, B5, C5, etc. Las flechas son paso de mensajes que se envían entre procesos. Por ejemplo, el evento B1 no puede suceder antes de que el evento C1 suceda. También se puede ver eventos concurrentes, A2 y B3 son concurrentes, sin embargo, A2 debe suceder luego que A1, por su parte B3 debe suceder luego de B2.

5.5. Sincronización de relojes físicos

Haga el siguiente ejercicio, tome varios relojes de su casa y consulte la hora, se dará cuenta que no todos los relojes tienen la misma hora, algunos estarán desigual inclusive en segundos. La siguiente actividad sería sincronizar los relojes, pero cuál reloj tomo para igualarlos, seguramente el reloj que parece más confiable o el más cercano. Una vez que estén sincronizados, consulte nuevamente la hora luego de una semana. Es posible que algunos relojes nuevamente estén desfasados. Y si los dejamos todo un año inclusive pueden estar desfasados en minutos u horas.

En cualquier sistema, cada dispositivo tiene un reloj físico posiblemente proveniente de diferentes fabricantes, inclusive la medida de un segundo entre un dispositivo y otro es diferente, por lo tanto, es necesario realizar un ajuste del reloj cada cierto intervalo de tiempo, y en un sistema distribuido una sincronización del reloj en general es muy importante.

La sincronización se da a través del intercambio de mensajes, se puede usar un dispositivo que pertenece al grupo del sistema distribuido como referencia e igualar el resto de los dispositivos de acuerdo a su reloj. En este caso, puede ser que el dispositivo de referencia se desfase. Se puede usar también un reloj externo al sistema e igualar los relojes de acuerdo a este, donde la responsabilidad de estar igualado es del reloj externo.

También se puede usar un reloj lógico en el cual los eventos o actividades de los procesos no van a depender de un reloj físico del sistema.

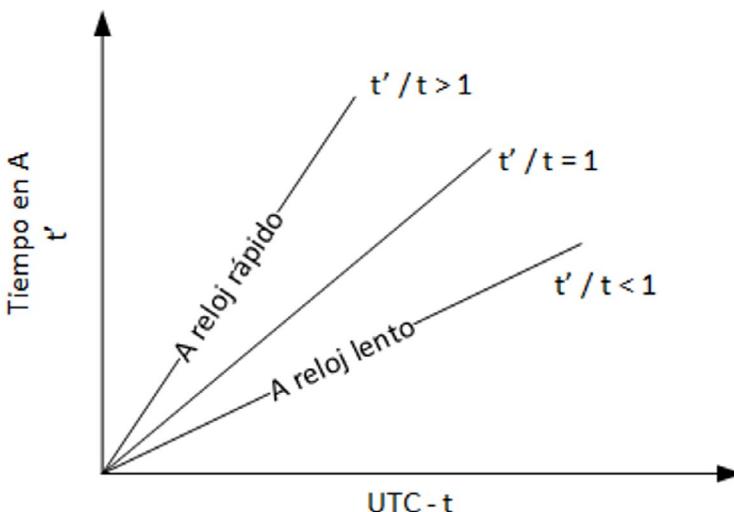
¿Cómo los dispositivos electrónicos miden el tiempo? Si hacemos un poco de historia se podrá ver que se lo hace acercándolo a los eventos astronómicos, traslación de la tierra al sol en años, estaciones con los meses y rotación de la tierra con los días, horas, minutos y segundos. Con esta información se crearon los primeros relojes mecánicos. Sin embargo, se debe considerar que los tiempos de traslación de la tierra se desfasan, por lo que se debe agregar un día cada cuatro años. Los tiempos de rotación de la tierra varían por lo que algunos días duran más que otros.

Para mejorar la precisión se crearon los relojes atómicos, que utilizan un patrón de frecuencia de resonancia atómica normal. En 1955 se construyó el primer reloj atómico de cesio 133 cuyo cristal oscila a una frecuencia bien conocida. La precisión es tan alta que solo se desfasa un segundo cada 30 millones de años (Essen & Parry, 1955).

Para coordinar el tiempo en todo el mundo se utiliza el estándar de tiempo UTC (Tiempo universal coordinado). El UTC utiliza el tiempo atómico internacional que es el promedio de la medición de tiempo atómico internacional basado en el átomo de cesio 133 de diferentes laboratorios distribuidos en todo el mundo.

Para sincronizar los relojes físicos en un sistema, se puede utilizar como referencia el tiempo definido por UTC, o una referencia propia del sistema sin el uso de UTC.

Figura 37.
Relación entre UTC y el reloj local



Nota. Elaborado por el autor

Como se puede ver la Figura 37 muestra la relación que existe para sincronizar el reloj, eje de las Y, usando como ejemplo el tiempo UTC que está representado en el eje de las X. Lo ideal es que el tiempo t' , el reloj del proceso del equipo A, para ejecutar el proceso i , sea el mismo que el t tiempo en UTC. Si el tiempo en A es mayor que el UTC, entonces tenemos un reloj muy rápido, y si es menor, es muy lento. En la práctica se permite una variable de tolerancia z de tal forma que exista un rango para mantener un límite inferior y un límite superior:

$$1 - z < \frac{A(t')}{UTC(t)} < 1 + z$$

Ahora analizaremos algunos algoritmos que intervienen en la sincronización del tiempo:

Sincronización relojes físicos

Recursos:

En el REA Curso Abierto de Sistemas Distribuidos de la Universidad Carlos III de Madrid (López Fuentes, 2015), específicamente en el tema 7 ([Tema 7 - Coordinación y sincronización distribuida – \(uc3m.es\)](#)) existe un enfoque similar de esta unidad, invito a hacer un cuadro comparativo de lo analizado de tal forma que se amplíen sus conocimientos.



Actividades de aprendizaje recomendadas

Es importante que en esta semana usted realice las siguientes actividades para consolidar su aprendizaje:

- Leer y familiarizarse con el material didáctico de la asignatura.
 - Lea nuevamente los contenidos de esta guía y cree un mapa conceptual.
 - Consulte el plan docente donde se muestra la secuencia de estudio de los contenidos, así como las actividades que complementan el aprendizaje organizadas por fechas, con su respectiva valoración, rúbricas de evaluación y distribución de las calificaciones por bimestre.
 - El entorno virtual de aprendizaje es el medio de interacción permanente entre docente y estudiante, donde encontrará orientaciones semanales, recursos, herramientas de interacción para con sus compañeros y docente.
- Ingrese al entorno virtual de aprendizaje para conocer las orientaciones de estudio y actividades a desarrollar esta semana; así como para realizar consultas a su docente tutor e interactuar con sus compañeros.
- Codificar el algoritmo de Cristian en Java.
- Codificar el algoritmo de Berkeley en Java.
- Configurar el protocolo NTP en su computador.

Nota: conteste las actividades en un cuaderno de apuntes o en un documento Word.



5.6. Sincronización a través de relojes lógicos

En aplicaciones distribuidas, los eventos o actividades que generan los procesos también son distribuidos lo que induce a una relación causal para generar un orden que debe ser sincronizado.

Esta relación causal de eventos es útil para analizar las propiedades de la computación distribuida, por ejemplo:

1. Equilibrio y justicia en eventos de exclusión mutua, por ejemplo, una primitiva `write()` lanzada por dos eventos al mismo tiempo, estos eventos deberían excluirse mutuamente, solo uno puede modificar el recurso y el otro debe esperar. Sin embargo, el segundo evento no debe esperar infinitamente, eventualmente debería cambiar el recurso.
2. Consistencia en bases de datos replicadas, si un proceso cambia un registro de una base de datos, todas las copias del registro deben actualizarse antes de que otra operación se haga sobre el registro.
3. Depuración en ambientes distribuidos, un ejemplo claro es cuando se trabaja con sistemas distribuidos basados en objetos, en el cual existen objetos que por su naturaleza son temporales, estos objetos deben borrarse en forma consistente en todo el sistema distribuido para liberar recursos y mejorar el funcionamiento del sistema.

Por lo tanto, esta relación de causalidad es importante gestionarla en todo el sistema distribuido y se la puede realizar con el uso de relojes lógicos.

5.6.1. Reloj lógico

Un reloj lógico es una instancia del sistema distribuido que le permite crear una forma de gestión de tiempo en todo el sistema distribuido, se utilizan para ordenar los eventos y mantener una relación de causalidad. Un reloj lógico en un sistema distribuido es necesario debido a que los procesos solo se comunican entre sí a través del intercambio de mensajes.

Las estructuras de los eventos se representan en forma abstracta a la computación distribuida, de tal forma que un proceso puede ser visto como una secuencia de eventos donde cada evento permite una transición atómica del estado local de un recurso, y las acciones de los eventos puede ser modelados utilizando por lo general 3 tipos de eventos: enviar mensaje, recibir mensaje, interno o un cambio de estado. Por ejemplo, definamos dos estados para un proceso: *listen* y *wait*. Cuando se inicia el proceso su estado estará en *listen*, esperando información para poder procesar. Una vez que recibe información el proceso a través del evento "enviar mensaje" o *send()*, cambia su estado a *wait*. Una vez que un mensaje llega al proceso este utiliza el evento "recibir mensaje" *recvto()* para procesarlo y cambia su estado nuevamente a *listen* para esperar información de la aplicación.

Como se mencionó, un reloj lógico C es un mecanismo abstracto que asigna a cualquier evento $e \in E$ que pertenece al grupo de eventos E , un valor de tiempo $C(e)$ en un dominio de tiempo T de tal forma que las siguientes condiciones son requeridas:

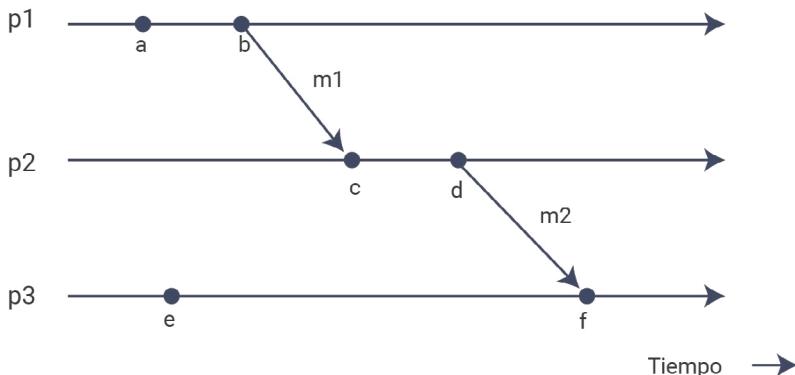
$$C: E \rightarrow T \text{ Donde } T \text{ conjunto de eventos parcialmente ordenado } e < e' \rightarrow C(e) < C(e')$$

Si se cumple lo anterior, entonces podemos mencionar que tenemos dos relaciones:

- a. Los eventos que ocurren en un proceso particular son totalmente ordenados por su secuencia de ocurrencia local, en otras palabras, si un evento e' ocurre antes de un evento e el tiempo local asignado será anterior al tiempo local asignado al evento e' .
- b. Para cualquier mensaje enviado de un proceso a otro, el tiempo lógico del evento *enviar* siempre será menor que el tiempo lógico de la recepción del mensaje. Cada evento de recepción tiene un evento correspondiente de envío y los eventos futuros no tienen influencia sobre los pasados, definiéndose una relación de causalidad.

Figura 38.

Procesos y eventos en un sistema distribuido



Nota. Tomado de Coulouris et al. (2012).

En la Figura 38 tenemos tres procesos que conforman un sistema distribuido, cada proceso tiene un conjunto de eventos, en este caso el tiene el conjunto de eventos conformado α y β , $Ep1=\{\alpha,\beta\}$. El proceso $p2$ a su vez $Ep2=\{c,d\}$ y el proceso $p3$ tiene el conjunto de eventos $Ep3=\{e,f\}$. Estos son eventos locales en cada uno de los procesos.

El sistema distribuido tiene un conjunto de eventos E conformado por todos los eventos de los procesos $p1$, $p2$, y $p3$, es decir $Ep1, Ep2$ y $Ep3$ son subconjuntos de $E=\{a,b,c,d,e,f\}$.

Lamport denominó a este orden parcial obtenido de generalizar estas dos relaciones (a) y (b), la relación *happened-before* (L. Lamport, 1978). En la literatura, esta relación también es denominada *ordenamiento causal* o *potencial ordenamiento causal*.

Coulouris, en su libro, denota la relación *happened-before* con el símbolo \rightarrow y agrega una tercera relación denominada transitividad (Coulouris et al., 2012).

- c. Si e, e' y e'' son eventos en el sistema distribuido de tal forma que $e \rightarrow e'$ y $e' \rightarrow e''$, entonces $e \rightarrow e''$ (Coulouris et al., 2012).

Como se puede ver en la Figura 39, los eventos α y β cumplen con la relación (a) porque están en el mismo proceso $p1$, lo mismo con los eventos $c \rightarrow d$ y $e \rightarrow f$ en los procesos $p2$ y $p3$ respectivamente.

Cuando hay envío y recepción de mensajes m_1 y m_2 , se cumple la condición (b) con los eventos $b \rightarrow c$ y $d \rightarrow f$ respectivamente. La relación (c) se cumple con los eventos $a \rightarrow f$.

También se puede apreciar que los eventos concurrentes en el ejemplo a y e son concurrentes, ya que están en diferentes procesos, y no hay cadena de eventos entre los dos. Por lo tanto, los eventos a y e no están ordenados. El sistema distribuido está parcialmente ordenado.

5.6.2. Implementación de reloj lógico

La implementación de un reloj lógico en software requiere:

1. Estructuras de datos para cada proceso para representar el tiempo lógico.
2. Un protocolo para actualizar la estructura de datos y asegurar la consistencia del sistema.

Cada proceso P_i debe mantener estructuras de datos para que permitan tener:

- Un reloj lógico local, RL_i , que permita que el proceso P_i mida su propio progreso.
- Un reloj lógico global, RG_i , que es una representación en cada proceso que permite tener una vista local del tiempo global. RL_i es parte del RG_i .

El protocolo debe asegurar la consistencia en cada proceso del reloj lógico local y del reloj lógico global. Por lo general, se usan dos reglas:

R1. Gestionar cómo el reloj lógico local es actualizado por un proceso cuando un evento se ejecuta.

R2. Gestionar cómo un proceso actualiza su reloj lógico global en concordancia con la vista global del tiempo y el progreso global.

5.6.3. Reloj escalar lógico-ordenamiento parcial

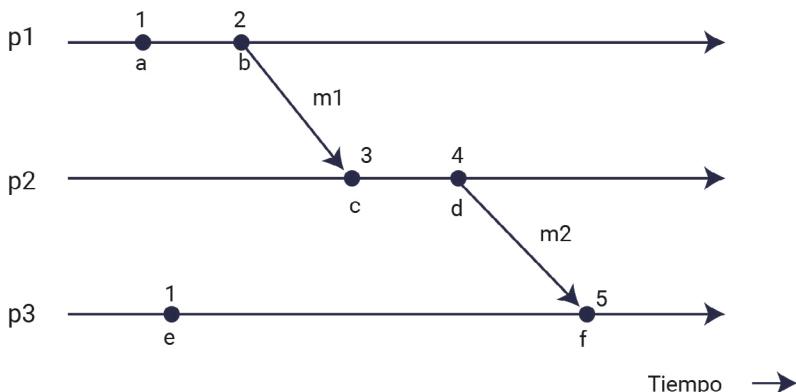
Lamport inventó un mecanismo simple para la relación *happened-before* a la que denominó reloj lógico a través de la implementación de un contador simple que no necesariamente tiene relación con el reloj físico de ningún componente. En otras palabras: es un contador de eventos en

todo el sistema distribuido. El reloj escalar de Lamport tiene las siguientes características:

- El dominio del tiempo es un conjunto de enteros no negativos $\{1,2,3,\dots,n\}$.
- El reloj lógico local RL_i y la vista local del tiempo global RG_i de un proceso P_i son colocados en un contador, una variable entera C_i .
- El contador incrementa su valor en forma lineal. El contador no tiene relación con el reloj físico.
- Cada proceso mantiene su propio reloj lógico para colocar estampa de tiempo a los eventos.

Figura 39.

Reloj lógico de Lamport



Nota. Tomado de Coulouris et al. (2012).

En la Figura 39 se puede apreciar el resultado de la implementación del reloj lógico de Lamport para el ejercicio de la Figura 39. Cuando se inician los procesos, la variable C_i o el contador se inicializa. Al ejecutarse un evento interno o un evento de enviar, cada proceso incrementa su contador C , por lo tanto, $C_i = C_i + u$ ($u > 0$), por ejemplo, los eventos a,b,d,e. Cuando el proceso P_1 envía un mensaje m , envía su variable C_i junto a la estampa de tiempo t correspondiente, los eventos d y e.

Cuando se ejecuta un evento de recepción en un proceso donde un mensaje con la estampa de tiempo (t) respectiva es recibido, eventos c y f, el reloj es actualizado de la siguiente forma:

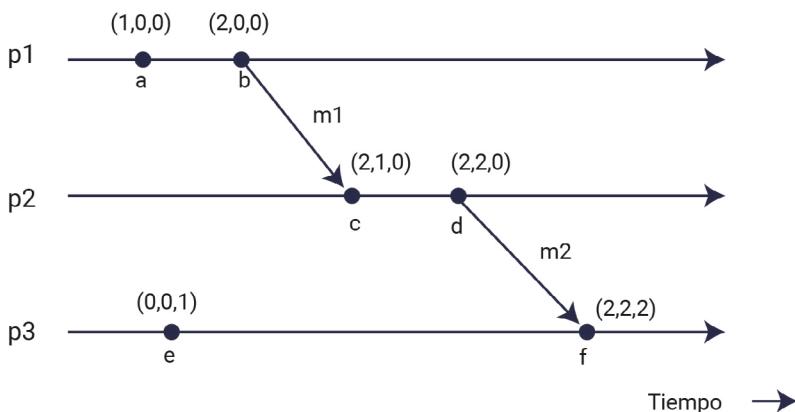
$$C_i = C_i + u \quad (u > 0)$$

El resultado es un orden parcial de eventos, porque como se puede ver en nuestro ejemplo, el proceso p_3 el evento e no sube en forma lineal.

5.6.4. Reloj vectorial lógico – ordenamiento total

En la Figura 40 se puede ver que los eventos no están totalmente ordenados debido a que distintos procesos, en nuestro caso p_1 y p_3 , han generado estampas de tiempo iguales para los eventos a y e . Para extender el orden parcial se puede incluir en la estampa de tiempo global la identificación del proceso.

Figura 40.
Reloj vectorial lógico



Nota. Tomado de Coulouris et al. (2012).

Las estampas de tiempo global pueden representarse como $(T\alpha, P\alpha)$ donde $T\alpha$ es la estampa de tiempo local y $P\alpha$ es la identificación del proceso. Como se puede ver en la Figura 40, existen tres procesos, en cada proceso se lleva un registro de la estampa de tiempo de cada uno de los procesos, por ejemplo, el evento a tiene como estampa de tiempo global $(1,0,0)$, donde la primera estampa de tiempo 1 es del proceso p_1 , el 0 es para el proceso p_2 y el último 0 es para el proceso p_3 . Como se puede dar cuenta, si existieran más procesos el tamaño de la estampa de tiempo global sería directamente proporcional al número de procesos. Esto puede ser una carga adicional para el tráfico de la red.

Es importante resaltar que el orden total es consistente con el orden parcial. Una estampa de tiempo global es menor que otra siempre y cuando

las estampas de tiempo local sean menores o iguales, y el número de proceso sea menor, es decir:

$$(Ta, Pa) < (Tb, Pb) \text{ si } ((Ta \leq Tb) \text{ && } (Pa < Pb))$$

El ordenamiento total no es adecuado para todas las aplicaciones distribuidas porque se pierden la independencia de algunos eventos provocando que la información se borre. Se obliga a ordenar eventos que no necesariamente necesitan ser ordenados, por ejemplo, el evento *b* en la Figura 41.

Mattern (Mattern, 1989) y Fidge (Fidge, 1991) desarrollaron, en forma independiente, una aproximación para un reloj vectorial con el objetivo de construir un mecanismo para que cada proceso tenga una aproximación óptima del tiempo global:

- Cada proceso tiene un reloj C_1 que consiste en un vector (un arreglo unidimensional) de longitud n , donde n es el cantidad total de procesos, cada celda de este arreglo $vt[j]$ es el reloj lógico local del proceso p_j y describe el progreso del tiempo lógico en p_j .

$$C_1 = vt [1 \dots n]$$

- Un proceso p_i incrementa en u el propio componente de su reloj cada vez que hay un evento.

$$C_1[i] += u$$

- Cada mensaje enviado o recibido contiene una retroalimentación para actualizar el vector del reloj local. Se actualiza cada celda del vector.

5.7. Estado global de un sistema distribuido

El estado global de un sistema distribuido está dado por el estado de cada uno de los eventos que se dan en los procesos. Cada proceso puede determinar su estado, pero lograr conocer un estado global de todo el sistema distribuido es una tarea más compleja de lograr.

El principal problema es la ausencia de un reloj global, si todos los procesos estuvieran perfectamente sincronizados se podría tener una foto global en un instante de tiempo con la información actualizada de todos los procesos y el estado de sus eventos.

El estado global de un sistema distribuido es necesario para detectar las siguientes relaciones que se dan en los procesos o la interacción entre ellos:

1. En sistemas distribuidos orientados a objetos es necesaria la recolección de objetos que han dejado de ser usados en el sistema distribuido. Los objetos ocupan memoria y recursos en cada sistema distribuido, por lo que es esencial que una vez que los objetos no sean usados estos recursos se liberen. Un objeto puede ser desecharido si no existe ninguna referencia al mismo en todo el sistema distribuido.
2. Procesos en espera indefinida o interbloqueos (deadlocks) se produce cuando, por ejemplo, un par de procesos están esperando indefinidamente que el otro proceso envíe un mensaje, provocando que el sistema distribuido no progrese en las acciones, el sistema distribuido queda esperando.
3. Cómo detectar que algoritmo distribuido ha finalizado. Este problema es muy parecido al anterior, sin embargo, los deadlocks afectan a un subconjunto de procesos.
4. Depuración distribuida: la depuración distribuida involucra que los eventos en un sistema distribuido guarden su estado durante su ejecución.

El estado global de un sistema distribuido es muy útil para detectar interbloqueos, poder determinar un punto de recuperación y realizar una recuperación del sistema distribuido, y hacer una finalización de los algoritmos distribuidos.

El estado global se logra a través del intercambio de la historia de eventos de cada proceso en el sistema. La cantidad de eventos que forman la historia de cada proceso debe ser prefijada. Cada evento, a su vez, cambia el estado del proceso por lo que se puede tener una historia de los estados de cada proceso.

Con esta información se puede generar una historia global para todo el sistema distribuido, que es la unión de la historia de cada proceso. De la misma forma, se puede crear una historia del estado de cada proceso para definir un estado global del sistema distribuido.

Un corte es similar a tomar una fotografía del estado del sistema distribuido en un momento determinado, por lo tanto es un subconjunto de la historia global y del estado global.

Recursos:

En el REA Curso Abierto de Sistemas Distribuidos de la Universidad Carlos III de Madrid (López Fuentes, 2015), específicamente en el tema 7 ([Tema 7 - Coordinación y sincronización distribuida – \(uc3m.es\)](#)) existe un enfoque similar de esta unidad relacionado con relojes lógicos, invito a hacer un cuadro comparativo de lo analizado de tal forma que se amplíen sus conocimientos.



Actividades de aprendizaje recomendadas

Es importante que en esta semana usted realice las siguientes actividades para consolidar su aprendizaje:

- Leer y familiarizarse con el material didáctico de la asignatura.
 - Lea nuevamente los contenidos de esta guía y cree un mapa conceptual.
 - Consulte el plan docente donde se muestra la secuencia de estudio de los contenidos, así como las actividades que complementan el aprendizaje organizadas por fechas, con su respectiva valoración, rúbricas de evaluación y distribución de las calificaciones por bimestre.
 - El entorno virtual de aprendizaje es el medio de interacción permanente entre docente y estudiante, donde encontrará orientaciones semanales, recursos, herramientas de interacción para con sus compañeros y docente.
- Ingrese al entorno virtual de aprendizaje para conocer las orientaciones de estudio y actividades a desarrollar esta semana; así como para realizar consultas a su docente tutor e interactuar con sus compañeros.

- Revise y profundice en los conceptos de la presente semana a través de la lectura de los cursos abiertos presentados como recursos.
- Implementar el reloj lógico de Lamport en Java.
- Implementar el reloj lógico vectorial en Java.

Nota: conteste las actividades en un cuaderno de apuntes o en un documento Word.

A continuación, se invita a que revise cómo van sus conocimientos haciendo la respectiva autoevaluación.



Autoevaluación 5

1. Seleccione la alternativa correcta:

En los sistemas centralizados el hardware es:

- a. Heterogéneo.
- b. Homogéneo.
- c. Compacto.

2. La gestión de un sistema distribuido es más compleja que un sistema centralizado.

- a. Verdadero
- b. Falso

3. Un sistema centralizado tiene una gestión de fallas más compleja.

- a. Verdadero
- b. Falso

4. ¿Cómo es la gestión del reloj global en un sistema distribuido?

- a. Por un componente central.
- b. A través del intercambio de mensajes.
- c. A través de interrupciones en el procesador.
- d. En cada componente del sistema.

5. Seleccione un ejemplo de una relación causal

- a. En Twitter el orden de las respuestas en un mismo anuncio.
- b. El orden de llegada en mensajes multicast.
- c. El almacenamiento de las réplicas en un sistema P2P.
- d. La descarga de las réplicas en un sistema P2P.

- 6. ¿Cuál de los siguientes define el proceso de comunicación de componentes de un sistema distribuido?**
- a. Traducir una referencia de un objeto a una forma interpretable.
 - b. Las entidades envían y reciben información de una forma sincronizada.
 - c. Considerar una abstracción de capas que se utiliza para diferenciar responsabilidades.
 - d. Donde están localizados los procesos, como interactúan entre sí en todo el sistema, como se comunican, cuáles son las responsabilidades de cada uno.
- 7. ¿En un sistema distribuido cómo se mantiene un estado global?**
- a. A través del intercambio de mensajes.
 - b. Usando las interrupciones del procesador local.
 - c. Definiendo un intercambio de estado con conocimiento previo.
 - d. El estado global no requiere una gestión.
- 8. Seleccione las alternativas correctas.**
- Seleccione las estrategias para definir un estado y tiempo global en un sistema distribuido.**
- a. Simular un tiempo local con el manejo de interrupciones en el procesador.
 - b. Simular un sistema asíncrono como síncrono.
 - c. Simular un tiempo global a través del intercambio de mensajes.
 - d. Forzar a los procesos a ejecutar los eventos al mismo tiempo predefinido.
- 9. Los relojes del sistema se pueden igualar tomando como referencia un reloj físico de un componente del sistema. Sin embargo, este último se puede desfasar.**
- a. Verdadero
 - b. Falso

10. Seleccione la alternativa correcta:

Para coordinar el tiempo en todo el mundo se utiliza el estándar:

- a. UTC
- b. IETF
- c. ANSI
- d. ITU

[Ir al solucionario](#)



Unidad 6. Modelamiento de sistemas distribuidos

En esta unidad, utilizando una estrategia divide y vencerás, se ha dividido la abstracción de sistemas distribuidos en componentes más pequeños, para luego analizar a profundidad sus entidades y relaciones existentes dentro de los componentes como la relación entre componentes.

6.1. Configuración, definición y gestión de un sistema distribuido

Para abordar esta unidad, usaremos la organización propuesta por Coulouris et al. (2012) para abstraer todos los componentes, posibles dimensiones, que puedan tener un sistema distribuido y las relaciones entre ellos para poder diseñarlo.

Un sistema distribuido se lo puede ver desde tres diferentes dimensiones:
a) de acuerdo al modelo físico; b) de acuerdo a su modelo arquitectónico; y,
c) desde el punto de vista de un modelo fundamental.

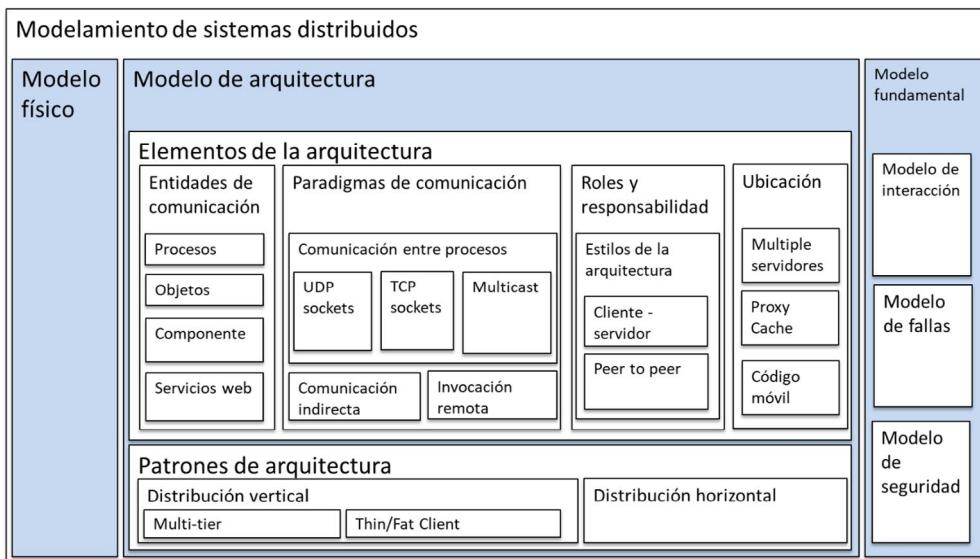
En la Figura 41, modelamiento de sistemas distribuidos, se puede ver la relación entre los diferentes modelos junto con sus componentes. El modelo físico tiene relación en mostrar cómo se conectan los procesos, los nodos y cómo se pueden nombrar cada uno de estos en el sistema distribuido. El modelo físico permite una descripción explícita de un sistema, se representa a través de abstracciones que capturan la composición del hardware en términos de computadores y sus redes interconectadas.

El modelo arquitectónico es el que está compuesto por los elementos de la arquitectura y los patrones de la arquitectura, menciona cuáles son las entidades que se comunican, qué paradigma de comunicación es utilizado, los roles y responsabilidades que tiene cada componente y cómo están ubicados dentro del sistema. Los modelos utilizados describen el sistema en términos de tareas computacionales y de comunicación realizados por las entidades del sistema.

El modelo fundamental trata de abstraer cómo interactúan los componentes, cómo pueden fallar los elementos y cómo el sistema distribuido se puede levantar de un fallo parcial o discernir si existe un fallo total. Está compuesto de tres modelos: el de interacción, de fallos y de seguridad.

Hemos preparado un recurso digital para que pueda interactuar con los contenidos de la presente unidad.

Figura 41.
Modelamiento de sistemas distribuidos



Nota. Elaborado por el autor

El modelo de interacción permite caracterizar cómo interactúan las entidades entre, por ejemplo, la relación happening-before que se discutió en el tema del reloj distribuido donde se analizó las relaciones de eventos causales y concurrentes.

El modelo fallas permite abstraer los errores que puede tener un sistema distribuido desde aquellos errores fáciles de gestionar como la caída total del sistema. O aquellos problemas que son difíciles de controlar, manejar o inclusive intuir, como por ejemplo, una falla de acceso a memoria que se da cuando se ejecuta.

El modelo de seguridad tiene que ver con las estrategias que debe considerar un sistema distribuido para que sea seguro y confiable, para

que no permita que una tercera entidad o un atacante vulnere el sistema distribuido y cuáles son las posibles cosas que deberíamos considerar para que este sistema sea más seguro.

Es importante considerar estas dimensiones o puntos de vista para el diseño de los sistemas distribuidos como una estrategia de dividir el concepto e identificar a qué modelo afecta alguna de las siguientes dificultades y poder definir la mejor estrategia para enfrentarlo y resolverlo, como por ejemplo:

- Los componentes del sistema están expuestos a variaciones en su carga de trabajo. Algunos recursos como servidores web son accedidos millones de veces en un día.
- Los componentes pueden desconectarse del sistema y volverse a conectar, por ejemplo, los dispositivos móviles como celulares.
- Algunas aplicaciones requieren requerimientos especiales, por ejemplo, aplicaciones multimedia que requieren alto ancho de banda y una baja latencia.
- Los componentes de los sistemas distribuidos son heterogéneos, diferente hardware, software y redes de computadores.
- Las redes difieren en rendimiento, es diferente el rendimiento de una red inalámbrica comparado con una red LAN.
- Los sistemas distribuidos están expuestos a amenazas externas o internas que pueden producir ataques a la integridad de los datos o ataques de negación de servicio.

6.2. Modelo físico

Es la representación de los equipos, hardware y cómo se conectan entre ellos. Un sistema distribuido debe coordinar todo este hardware en un ambiente de red donde los componentes se comunican únicamente por el paso de mensajes.

De acuerdo a la evolución del hardware y las redes de computadores, los sistemas distribuidos han evolucionado definiendo tres generaciones:

La primera generación apareció con el nacimiento de las redes de computadores, a finales de los 70 y principios de los 80. Los sistemas distribuidos de esta generación varían en tamaño de 10 a 100 nodos conectados y, por lo general, la cobertura era para redes LAN.

Con el aparecimiento de Internet emerge la segunda generación de sistemas distribuidos, ahora la cobertura de esta infraestructura es global. Y se empezaron a ver algunos sistemas distribuidos de uso mundial, uno de los más famosos era el proyecto SETI (Manu, 2021) cuyo objetivo era buscar vida inteligente en otros planetas utilizando el procesamiento local de cada uno de los clientes que se conectan al proyecto. Un programa recibe información ayuda con su procesamiento local y envía los resultados de regreso al proyecto.

La tercera generación define a los sistemas distribuidos contemporáneos, los cuales tienen que gestionar propiedades actuales como movilidad y ubicuidad. Aquellos nodos móviles son independientes de la ubicación, sin embargo, por esta misma característica tienen que manejar capacidades de ancho de banda diferente por cada red que visiten. Estos sistemas distribuidos deben trabajar con diferentes y nuevos tipos de tráfico con sus propios requerimientos. Por ejemplo, el tráfico multimedia que requiere una latencia baja, además un gran ancho de banda. Esta generación debe también trabajar con las nuevas formas de gestionar los recursos, como la computación en la nube.

Con el avance de la tecnología, la disminución del tamaño del hardware, incluyendo los procesadores y dispositivos de almacenamiento, las mejoras constantes en desarrollo de software y las comunicaciones con una mejor capacidad, han aparecido sistemas distribuidos de sistemas. Son sistemas más complejos que consisten una serie de subsistemas con autonomía propia, los cuales en conjunto realizan alguna tarea específica o tareas en corresponsabilidad. Por ejemplo, un sistema de alerta de tsunamis, o sistemas de control de congestión vehicular.

Existen algunas diferencias importantes entre las distintas generaciones de sistemas distribuidos, una de ellas es el área de cobertura o el tamaño del sistema. Otra diferencia es el soporte para la seguridad, la primera generación de los sistemas distribuidos no consideró a la seguridad como un requisito inicial para su implementación y desarrollo, mientras que ahora el tema de seguridad, debido entre otras cosas a la cobertura global de los sistemas, es una consideración obligatoria cuando se diseña

sistemas distribuidos. En el mismo sentido está el soporte para la calidad de servicio. Finalmente, la heterogeneidad, la transparencia y apertura, ahora los sistemas distribuidos están compuestos por componentes que provienen de diferentes creadores de hardware y software, por lo tanto, para poder interactuar los sistemas distribuidos deben ser transparentes, deben documentar el uso de sus interfaces para que puedan interactuar entre sí.

Un ejemplo de la evolución de los sistemas distribuidos y su relación con las generaciones se puede ver en la infografía donde se aprecia el avance de la tecnología de Internet desde su nacimiento: un sistema distribuido de segunda generación con sistemas propios, como el correo electrónico, y hasta una proyección a la década del 2030, toda esta evolución está relacionada con los avances en la tecnología para la interacción hombre/máquina o máquina/máquina.

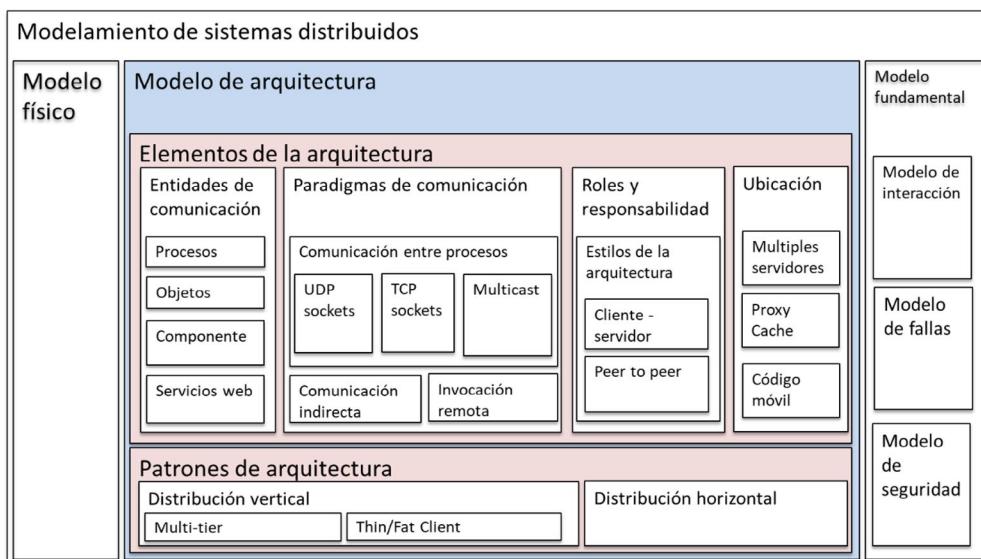
[La expansion de la WEB](#)

6.3. Modelo de arquitectura

Los modelos arquitectónicos, Figura 42, se componen de dos conjuntos: los elementos de la arquitectura y patrones de la arquitectura.

Figura 42.

Modelo de arquitectura y sus componentes



Nota. Elaborado por el autor

6.3.1. Elementos de la arquitectura

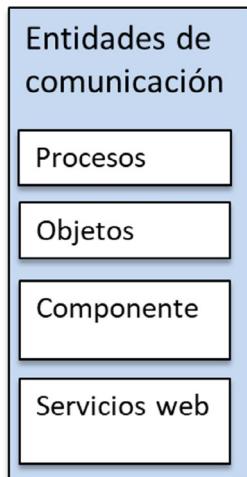
Los elementos de la arquitectura tienen relación con las entidades de comunicación, qué tipo de comunicación utilizan, qué rol y responsabilidad va a tener cada uno y su ubicación en el sistema distribuido. Los elementos de la arquitectura sirven para entender cómo interactúan los bloques fundamentales de un sistema distribuido, enfocándolos en la respuesta a las siguientes preguntas: ¿cuáles son las entidades que se están comunicando? ¿Cómo estas entidades se comunican o qué paradigma de comunicación están utilizando? ¿Qué función cumplen, qué responsabilidad tienen en el sistema distribuido? ¿Cómo están ubicados en la topología física del sistema?

6.3.1.1. Entidades de comunicación

Las entidades de comunicación, Figura 43, se las puede analizar desde dos puntos de vista diferentes: desde una perspectiva del sistema y desde una de orientación al problema.

Figura 43.

Entidades de comunicación de un sistema distribuido



Nota. Elaborado por el autor

Desde una perspectiva del sistema distribuido las entidades que se comunican son los procesos que se asocian con dispositivos o software. Sin embargo, hay casos especiales como los sensores en donde el software y el hardware están tan unidos que no se puede diferenciar, o también puede ser que un dispositivo sea tan potente en hardware como en el

software que el mismo proceso puede atender varias peticiones a la vez a través del uso de técnicas de programación como los hilos, thread en inglés.

Cuando nos referimos a las entidades de comunicación desde una perspectiva orientada al problema, se las puede referir como objetos, componentes o servicios web. Un objeto es una entidad de programación que tiene sus propios atributos y los métodos que permiten interactuar con dichos atributos. Los objetos solo pueden interactuar con otros objetos a través de la definición de interfaces. Los objetos pueden representar a entidades del dominio del problema.

Los componentes son similares a los objetos, se pueden definir como una unidad de composición que define específicamente sus interfaces. Se pueden considerar como una caja negra.

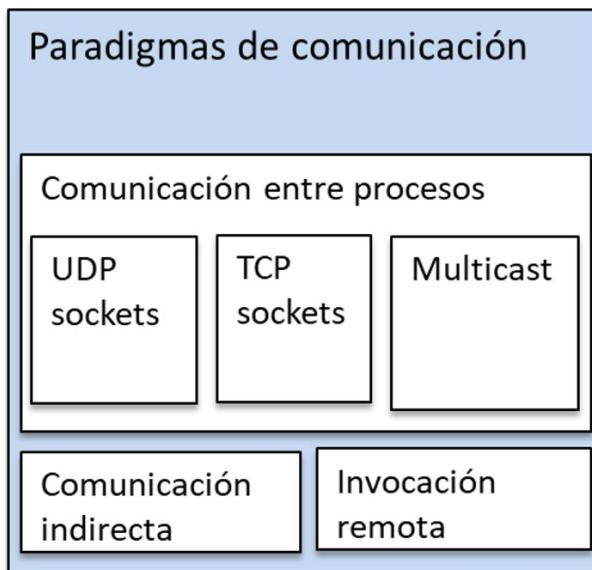
Los servicios web son otra de las entidades de comunicación, son aplicaciones de software que se identifican con un URI (Identificador de recursos uniforme), no deben ser descargados para ser usados con los clientes. Los clientes los pueden descubrir y acceder a su funcionalidad usando protocolos estándares basados en XML. Son más accesibles que los componentes por usar protocolos estándares.

6.3.1.2. Paradigmas de comunicación

En sistemas distribuidos, las entidades pueden usar cualquiera o una mezcla de los siguientes paradigmas de comunicación (Figura 44): comunicación entre procesos, invocación remota y comunicación indirecta.

Figura 44.

Paradigmas de comunicación



Nota. Elaborado por el autor

La **comunicación entre procesos**, ya se ha analizado anteriormente, se basa en una comunicación directa entre dos entidades de comunicación. En otras palabras, es necesario que los procesos o nodos comunicándose estén acoplados en tiempo y espacio. Como se analizó, cuando se usa sockets TCP o UDP, para que exista comunicación, las dos entidades deben existir en el momento de que se realice la envío de mensajes. La comunicación multicast es un caso especial porque, aunque un equipo esté transmitiendo a un grupo, no es requisito que los clientes estén presentes.

La **invocación remota** significa que la aplicación puede usar un servicio o un proceso a una función de un proceso que se encuentra en otra entidad dentro del sistema distribuido, al cual se comunica a través de una infraestructura de red. La utilización de esta función o proceso debe ser transparente, es decir, puede llamar a una interfaz del proceso remoto desde el computador local.

La invocación remota tiene como resultado la llamada de una operación remota, un procedimiento o un método. El emisor explícitamente invoca, a través de mensajes, operaciones en los receptores asociados. Los receptores deben estar pendientes del emisor, es decir, un servicio debe estar activo. Como se puede ver deben estar acoplados en tiempo.

La invocación remota difiere de acuerdo al tipo de paradigma de programación con el que se desarrolla las aplicaciones:

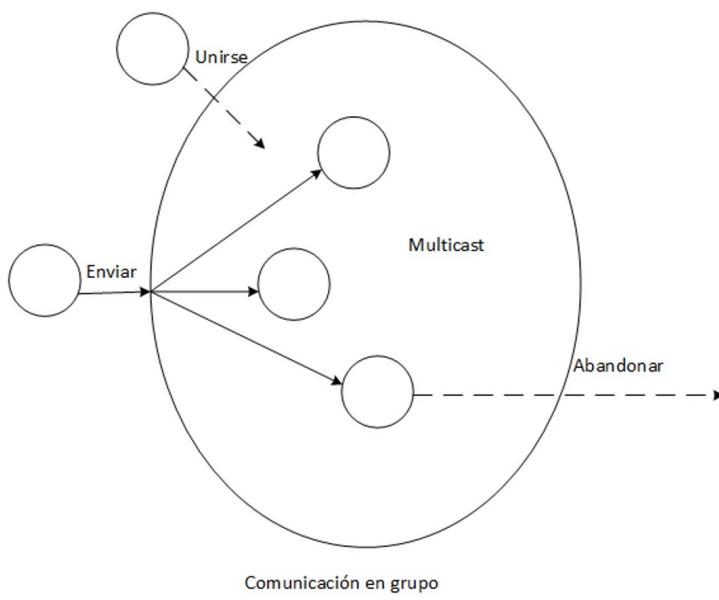
- Llamado de procesos remotos, RPC: cuando se usa una orientación basada en procedimientos y funciones. Los procedimientos en computadores remotos pueden ser llamados como si ellos estuvieran disponibles en el espacio local.
- Invocación remota de métodos: RMI; cuando las aplicaciones tienen un paradigma de orientación a objetos. Un objeto puede invocar un método en un objeto remoto.

La **comunicación indirecta** involucra un dispositivo entre los procesos que se comunican y que gestiona toda la interacción, ejemplos claros son las aplicaciones de Facebook o Twitter, donde el usuario deposita información para que luego pueda ser obtenida en cualquier momento por otro usuario.

En la comunicación indirecta, el emisor a veces no está pendiente de si el receptor está activo o no, y posiblemente tampoco tiene un conocimiento previo de que la entidad va a recibir esa información. Tanto el emisor como el receptor están desacoplados en tiempo y espacio. Algunos mecanismos que utilizan comunicación indirecta son la comunicación en grupo, los sistemas publicación-suscripción, las colas de mensajes, los espacios de tupla y la memoria compartida distribuida.

La comunicación en grupo, Figura 45, se da cuando la comunicación se envía a un grupo de recipientes. Existe la noción de grupo a través de un identificador de grupo. Los participantes que requieran pertenecer al grupo lo pueden hacer a través del identificador. El proceso que envía no conoce la cantidad de usuarios que pueden recibir la información, por esta razón es desacoplado en espacio. La comunicación en grupo es compleja de gestionar para un sistema distribuido, especialmente para el problema de consenso, que por lo general requiere de un conocimiento de la cantidad de nodos o procesos que pertenecen al sistema.

Figura 45.
Comunicación en grupo

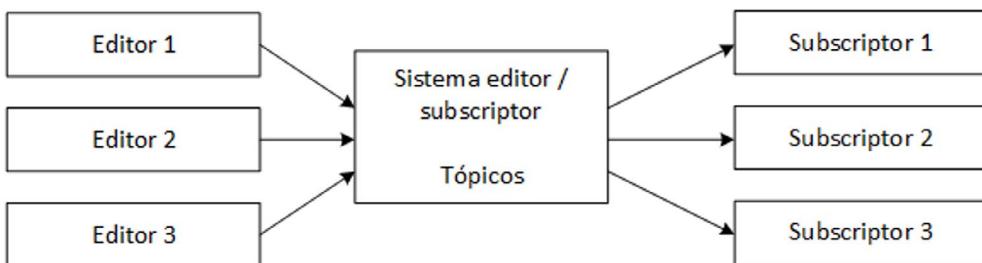


Nota. Elaborado por el autor

En la comunicación en grupo los participantes pueden unirse al grupo, a través del identificador del grupo, y abandonarlo en cualquier momento. Al usar multicast generan un ahorro en recursos de la red de núcleo, como ancho de banda y procesamiento.

Los sistemas de edición-subscripción (publish/subscribe), Figura 46, donde los productores de información o contenido colocan información y los subscriptores en cualquier momento la pueden consumir previa una suscripción. Un ejemplo de estos sistemas es Twitter.

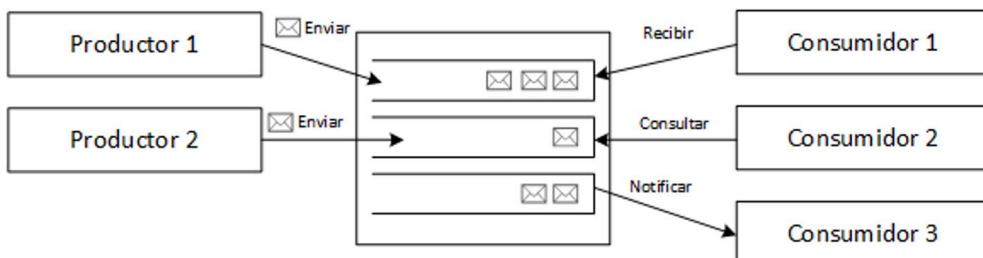
Figura 46.
Patrón editor / subscriptor



Nota. Elaborado por el autor

Las colas de mensajes, Figura 47, son muy similares a los sistemas de edición/ suscripción, sin embargo, aquí los productores de información envían mensajes a una cola específica donde el consumidor puede recibir y consultar por contenido. La cola de mensajes también puede notificar que existe información en la cola. Por lo general, las colas usan un mecanismo FIFO para almacenar los mensajes. Los colas de mensajes no son acopladas en tiempo y, al contrario de los sistemas edición/publicación, manejan una relación de uno a uno. El ejemplo es un sistema para la bolsa de valores donde el receptor requiere conocer la variación de los valores acciones cuando estas cambien.

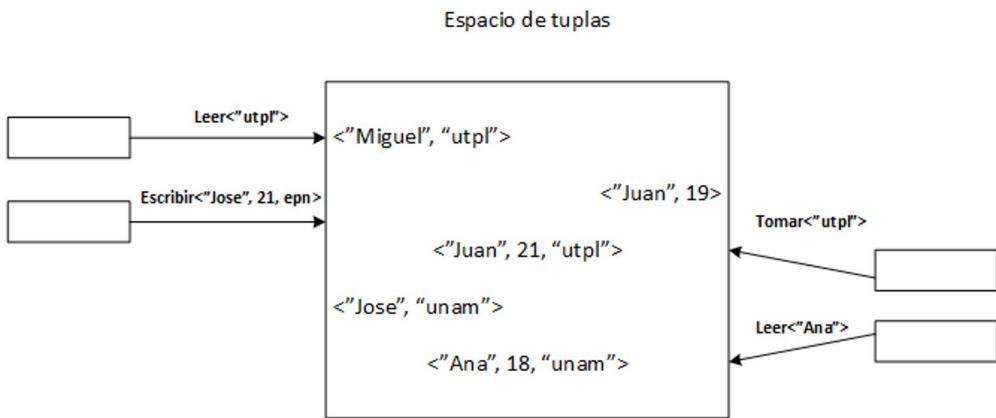
Figura 47.
Sistema de cola de mensajes



Nota. Elaborado por el autor

Los espacios de tuplas, Figura 48, almacenan secuencias de valores semiestructurados agrupados y se los trata como un valor único. Los procesos pueden depositar tuplas en espacios de memoria y luego los receptores pueden leerlas. Las tuplas son desacopladas en espacio porque los procesos que envían y reciben pueden venir de cualquier parte. También son desacopladas en tiempo porque las tuplas pueden residir indefinidamente en el espacio de tuplas. Una tupla por lo general tiene la forma $\langle \text{var1}, \text{var2}, \dots, \text{var n} \rangle$ por ejemplo: $\langle \text{"Miguel"}, 19, \text{"utpl"} \rangle$.

Figura 48.
Espacio de tuplas

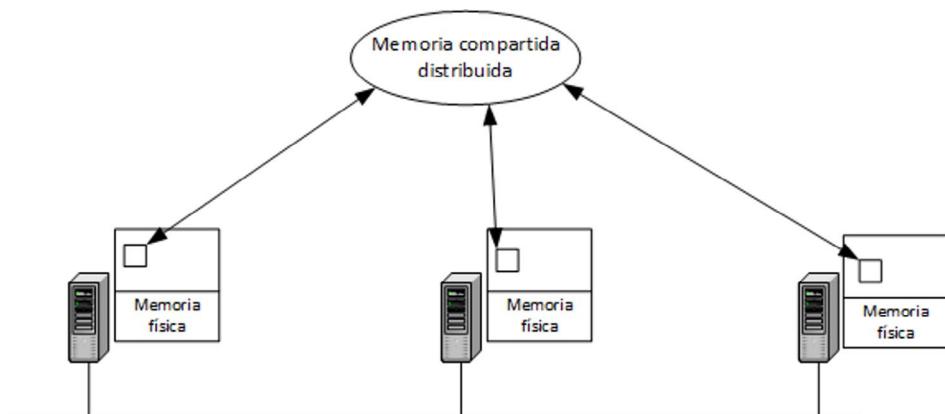


Nota. Elaborado por el autor

En la Figura 48 se pueden ver tres de las principales operaciones que se pueden hacer en un espacio de tuplas. Por ejemplo, Leer <"utpl"> obtendrá sin borrar el espacio todas las tuplas que cumplan la condición, en este caso sería dos tuplas. La operación Escribir almacena una tupla en el espacio. Finalmente, la operación Tomar, lee la tupla y la borra del espacio.

La memoria compartida distribuida, Figura 49, permite a los desarrolladores de software tener una alta abstracción de la ubicación de los recursos para que puedan hacer operaciones de lectura y escritura. Este tipo de sistemas requiere de una alta coordinación y sincronización para mantener la consistencia del sistema.

Figura 49.
Memoria compartida distribuida



Nota. Elaborado por el autor

La memoria compartida distribuida (DSM) es una abstracción utilizada para compartir datos entre computadores que no comparten memoria física. Los procesos acceden a DSM para leer y actualizar dentro de sus espacios de direcciones, sobre lo que aparenta ser la memoria interna normal asignada a un proceso. Sin embargo, existe un sistema subyacente en tiempo de ejecución que asegura de forma transparente qué procesos diferentes ejecutándose en distintos computadores observen las actualizaciones realizadas entre ellas.



Actividades de aprendizaje recomendadas

Es importante que en esta semana usted realice las siguientes actividades para consolidar su aprendizaje:

- Leer y familiarizarse con el material didáctico de la asignatura.
 - Lea nuevamente los contenidos de esta guía y cree un mapa conceptual.
 - Consulte el plan docente donde se muestra la secuencia de estudio de los contenidos, así como las actividades que complementan el aprendizaje organizadas por fechas, con su respectiva valoración, rúbricas de evaluación y distribución de las calificaciones por bimestre.
 - El entorno virtual de aprendizaje es el medio de interacción permanente entre docente y estudiante, donde encontrará orientaciones semanales, recursos, herramientas de interacción para con sus compañeros y docente.
- Con la información de los cursos abiertos, haga un análisis del acoplamiento en tiempo y espacio de los paradigmas de comunicación descritos en esta semana.

Nota: conteste las actividades en un cuaderno de apuntes o en un documento Word.

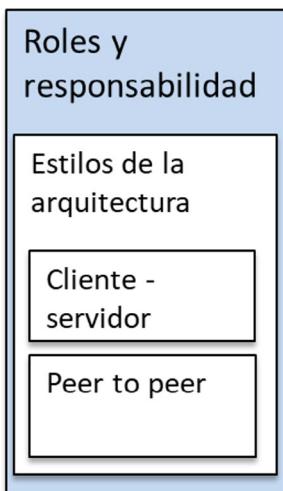


En la presente semana continuamos con el estudio de elementos de la arquitectura, enfocando algunos temas relacionados con roles y responsabilidades de las entidades de un sistema distribuido y su ubicación:

6.3.1.3. Roles y responsabilidades de las entidades de un sistema distribuido

Los roles y responsabilidades, Figura 50, tratan de afrontar la pregunta, ¿qué es lo que hace cada entidad dentro del sistema distribuido? Desde una dimensión del estilo de la arquitectura se puede responder desde dos puntos de vista, cliente/servidor y la comunicación peer to peer.

Figura 50.
Roles y responsabilidades

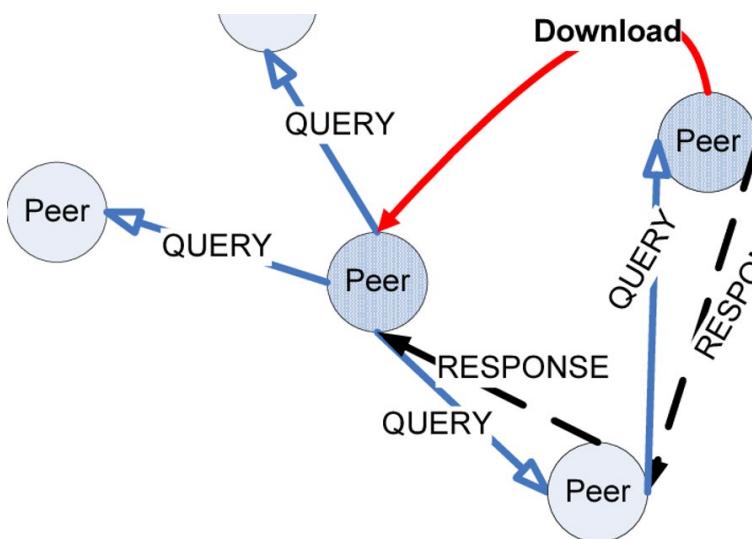


Nota. Elaborado por el autor

Cuando analizamos los sockets TCP y UDP, implícitamente se introdujo la idea del estilo de comunicación cliente/servidor, en donde cliente invoca un servicio alojado en un servidor y este, de acuerdo al requerimiento y a su capacidad, responde con el resultado deseado al cliente. El servidor puede ser considerado un cuello de botella si no tiene la capacidad suficiente para atender a todos los clientes. La seguridad es baja al ser el servidor un equipo primordial al que los atacantes o las fallas pueden comprometer.

Con un paradigma peer-to-peer, Figura 51, los roles están distribuidos de forma que un nodo, al mismo tiempo puede ser cliente y servidor, lo que permite mejorar la confiabilidad porque un recurso compartido puede estar replicado en todos o en la mayoría de los equipos. Si un nodo en la red abandona el sistema, el acceso al recurso aún está disponible.

Figura 51.
Comunicación peer-to-peer



Nota. Tomado de Lua et al. (2005).

Los sistemas peer-to-peer son más complejos que los cliente/servidor para gestionar la incorporación y retiro de nodos.

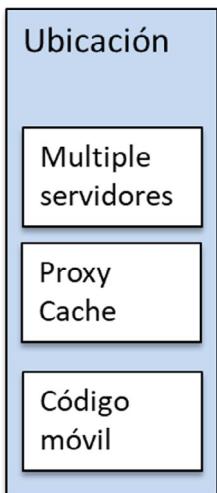
6.3.1.4. Ubicación

La ubicación de los procesos y servicios de un sistema distribuido, Figura 52, en la infraestructura física distribuida del sistema es un tema muy importante que permite diseñar un sistema con los requisitos de distribución de carga, seguridad, calidad de servicio y confiabilidad. Un ejemplo claro es la red Akamai, donde el servicio debe ser geográficamente distribuido para mejorar la carga a los servidores y mejorar la experiencia de usuario, incluso si existe un error en un servicio no afecta a toda la aplicación, lo que permite mejorar la confiabilidad del sistema.

Los procesos o servicios pueden estar ubicados en múltiples servidores, o estar en un servicio indirecto de proxy cache, o pueden ser un programa

de software que puede descargarse desde un equipo y ejecutarse en otro dispositivo.

Figura 52.
Ubicación entidades



Nota. Elaborado por el autor

Los servicios de un sistema distribuido pueden ser proporcionados por múltiples servidores, un cliente puede ser atendido por un servidor y otro cliente recibir el mismo servicio por otro servidor. Existen algunas formas en las cuales se puede lograr, por ejemplo, geolocalización o replicación. De cualquier forma, el servicio debe ser coordinado para, por ejemplo, si usa replicación determinar y actualizar la información con la copia más reciente.

Un servicio proxy cache, es un tipo de comunicación indirecta en el cual un equipo intermedio accede a un recurso y lo almacena en su memoria física. Los clientes, cuando quieren acceder al recurso, en primer lugar consultan al proxy cache por el recurso. Si el proxy cache tiene la copia más reciente lo envía al cliente, caso contrario solicita la copia más reciente del recurso antes de enviarlo al cliente.

Un código móvil puede ser descargado y ejecutarse en la máquina del cliente. Por ejemplo, un applet. Otro tipo de código móvil son los agentes móviles, en el cual código e información viajan por la red y realizan tareas específicas en el computador anfitrión. El código móvil tiene algunas ventajas: mejora el procesamiento, el uso de la red, porque ocupa menos ancho de banda, y la distribución de carga es repartida entre los clientes.

6.3.2. Patrones de la arquitectura

Los patrones de la arquitectura usan los elementos de la arquitectura para darles una estructura y organización que suele ser bien usada cuando se diseñan sistemas distribuidos. Se utilizan en conjunto con otro tipo de patrones, representaciones y modelos.

Figura 53.

Patrones de la arquitectura

Patrones de arquitectura	
Distribución vertical	Distribución horizontal
Multinivel	Cliente Thin/Fat

Nota. Elaborado por el autor

En este apartado mostramos algunos patrones importantes para sistemas distribuidos, Figura 53, incluyendo una distribución horizontal o por capas, así como una distribución vertical. Les invito a revisar este interesante apartado.

6.3.2.1. Distribución horizontal

La distribución vertical, que ya la hemos estudiado en el apartado 2.6 Abstracción de la red por capas o niveles, tiene que ver con las capas o niveles, por ejemplo, las capas del modelo TPC/IP. Las capas son utilizadas por sistemas complejos con el objetivo de dividirlo en conceptos más pequeños que puedan ser mejor entendidos y manejados. Cada capa brinda su servicio a las capas inferior y superior a través de interfaces bien definidas. Las interfaces en sistemas distribuidos pueden ser llamadas a procesos, funciones, objetos o componentes.

En sistemas distribuidos hay que diferenciar aquellas capas de las cuales se sirve el sistema distribuido de aquellas en las que se construye. Por lo general, un sistema distribuido puede tener las siguientes capas, desde la más cercana al usuario:

- Aplicación o servicio: donde se lleva el proceso de negocio del sistema distribuido.
- Middleware: en donde los procesos de gestión de procesos, de sincronización necesarios se llevan a cabo. Abstacta a la capa superior

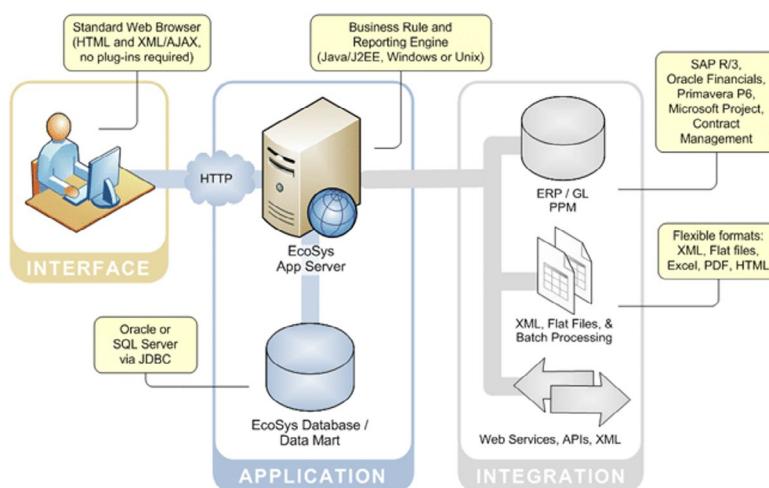
toda la posible heterogeneidad de las capas inferiores, sistema operativo y red.

- Sistema Operativo: el sistema operativo es parte de la plataforma o la infraestructura, por lo general los servicios que brinda no pueden ser modificados, pero son estándares especialmente para procesos de comunicación. Aquí están las primitivas para sockets TCP y UDP que analizamos anteriormente.
- Computadores y el hardware de red: esta capa es parte de la plataforma y debe ser considerada cuando se diseñan sistemas distribuidos.

6.3.2.2. Distribución vertical

La distribución vertical es una distribución física de dónde van los componentes, trata de agruparlos de acuerdo a alguna funcionalidad del sistema distribuido. Se puede considerar como una extensión de la comunicación cliente-servidor. Es usada para mostrar la distribución de funciones entre diferentes servicios. Podemos mencionar dos grandes grupos: la distribución multinivel y aquellos sistemas distribuidos donde los clientes tienen características diferentes, son robustos (fat) o no tienen mayor capacidad (thin).

Figura 54.
Distribución multinivel



Nota. Tomado de Enterprise Project Performance Platform Architecture | EcoSys (2021).

En la Figura 54 se muestra un ejemplo de la arquitectura de una aplicación comercial, este diagrama utiliza una representación multinivel. A la derecha está la interfaz de usuario, en el centro están los equipos o servidores frontales y de procesamiento, y en la capa a la izquierda está el nivel de almacenamiento. Esta distribución mejora la confiabilidad del sistema al distribuir la responsabilidad. Una aplicación cliente-servidor por lo general tiene tres niveles: presentación, procesamiento y almacenamiento.

Se puede también discutir un patrón de la arquitectura basado en las capacidades de los clientes. Por un lado, tenemos equipos de altas prestaciones para procesamiento y almacenamiento, y por otro lado tenemos equipos que no tienen esa capacidad, como por ejemplo los equipos en las redes de sensores. Esta capacidad de los clientes es un tema importante que considerar cuando se diseña sistemas distribuidos.

Cuando tenemos equipos robustos, se puede aprovechar la computación distribuida para usar las capacidades de los clientes y poder distribuir la carga del sistema distribuido tanto para memoria o procesamiento. Sin embargo, aumenta la complejidad para la sincronización de actividades. En el cliente pueden funcionar en forma sincronizada y distribuida los tres niveles de aplicación, presentación, procesamiento y almacenamiento.

Los servicios en la nube o las aplicaciones o los nuevos paradigmas, como por ejemplo, la red como servicio, optan por diseñar un sistema distribuido basado en no dar la responsabilidad de procesamiento, sincronización, etc., a los clientes. Los clientes son usados únicamente para presentación de información. Sin lugar a dudas, las mejores tecnologías de red en almacenamiento y procesamiento han logrado que esta sea una opción que considerar. El nivel de presentación se da a los clientes, mientras que el procesamiento y almacenamiento lo realiza el sistema distribuido.

6.4. Modelo fundamental

El modelo fundamental, Figura 55, hace referencia a cómo las entidades interactúan, en qué forma lo hacen, qué tipos aspectos se debe considerar cuando interactúan, qué fallas pueden tener los elementos del sistema y qué acciones se puede considerar para sobreponer la fallas, y ,finalmente, permite analizar cómo la seguridad se da en el sistema distribuido.

Figura 55.
Modelo fundamental



Nota. Elaborado por el autor

Ahora estudiaremos cada uno de estos modelos:

6.4.1. Modelo de interacción

Los nodos, procesos, objetos, que son las entidades de comunicación, interactúan entre sí por el intercambio de mensajes en una red de computadores a través de un canal de comunicación, como se puede observar hemos usado la misma figura de la comunicación entre procesos, la Figura 14, para ver cómo los elementos de la comunicación pueden afectar la interacción entre ellos, Figura 56. Esta interacción puede ser afectada por algunos factores de entre los cuales podemos mencionar:

Figura 56.

Interacción básica de entidades



Nota. Elaborado por el autor

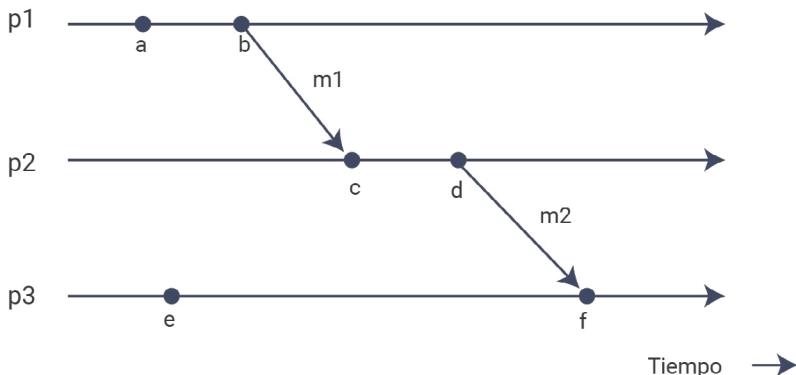
- El canal de comunicación tiene un rendimiento asociado que puede afectar a la interacción de los procesos, este rendimiento se puede medir de acuerdo a la latencia, el ancho de banda disponible y la variación de llegada de mensajes (jitter).
- Los nodos, procesos u objetos, son afectados principalmente en temas de coordinación debido a que los relojes físicos pueden no ser exactos.
- Los procesos pueden almacenar sus mensajes de entrada o salida en memoria local, la gestión de esta memoria también afecta a la interacción de los procesos, como se pudo ver en el apartado 3.8 Comunicación de segmentos con TCP.

Las entidades pueden interactuar en dos formas, síncrono y asíncrono. La comunicación síncrona define tiempos para el procesamiento y el envío de mensajes dentro de un intervalo de tiempo bien conocido. En la interacción asíncrona es más complicado poder determinar estos tiempos. Sin embargo, cuando se diseña sistemas distribuidos asíncronos se utiliza como base o se simula el sistema con un modelo de interacción síncrono.

Otra característica que se debe considerar en el modelo de interacción, es cómo los procesos en un sistema distribuido ordenan los eventos, por ejemplo, queremos que en el sistema distribuido el evento *recibir* en un proceso exista *luego de un evento enviar* en el proceso emisor. La orden real de eventos, Figura 57, de un sistema distribuido se puede lograr con los relojes lógicos de Lamport y los relojes lógicos vectoriales.

Figura 57.

Orden real de eventos



Nota. Elaborado por el autor

Determinar el orden de los eventos es un tema complejo de asumir en un sistema distribuido, especialmente para los procesos de sincronización y recuperación de fallas.

6.4.2. Modelo de fallas

Un sistema distribuido, como cualquier sistema, no está exento de fallas, las mismas que se pueden producir en cualquiera de los elementos del sistema distribuido. Un modelo de fallas define la forma en que las fallas pueden ocurrir y cuáles son los efectos en el desempeño del sistema.

Las fallas, por lo tanto, pueden darse en los canales de comunicación, en los procesos o nodos, y en la memoria de entrada y salida de los mensajes. Las fallas se pueden clasificar en:

- Fallas por omisión: los procesos o canales dejan de funcionar.
- Fallas arbitrarias: error no esperado en canales o procesos.
- Fallas en la sincronización o de tiempo: errores en la sincronización de los mensajes cuando no se realizó una actividad o evento en el tiempo determinado.

A continuación, en el listado de fallas que pueden existir, analice cada falla y determine si son fallas por omisión, arbitrarias o de tiempo:

- Falló la red, el sistema no tiene acceso a enviar y recibir información.
- Falló la tarjeta de red del equipo y no se puede recibir o enviar mensajes.

- Se sobrecargó la aplicación de un servidor web, sin embargo, el equipo sigue funcionando.
- Un mensaje fue alterado antes de llegar al receptor.
- El equipo dejó de funcionar por exceso de calor.
- El equipo accidentalmente fue desconectado de la red o del suministro eléctrico.
- Un equipo que pertenece a la red ha sido vulnerado y envía información falsa o alterada al sistema, afectando los procesos de consenso.

Fallas de omisión

Las fallas de omisión son “benignas” cuando se diseña un sistema distribuido porque este tipo de fallas se pueden anticipar y, por lo tanto, modelar estrategias para mantener el sistema operando, como por ejemplo, un detector de fallos. Las fallas de omisión se pueden clasificar en: fallo-parada, crash, omisión, omisión de envío y omisión de recepción.

El tipo de fallo-parada significa que un proceso se detiene y permanece en ese estado. Esta falla es manejable porque cualquier otro proceso puede detectar al proceso detenido. Un ejemplo claro se da en los protocolos de enrutamiento dinámicos para redes de computadoras que determinan si un enrutador falló y pueden actuar en consecuencia determinando una nueva ruta. Para detectar este tipo de fallos, los enrutadores envían mensajes cortos a sus vecinos para determinar si siguen funcionando.

En fallo de tipo crash el proceso se detuvo y se mantiene detenido, sin embargo, el resto de los procesos no pueden determinar que el proceso fallido se ha detenido. Un ejemplo de este tipo de falla es el servidor web saturado que no puede responder a más usuarios, pero el equipo está encendido y conectado a la red.

Las fallas de omisión afectan al canal, un proceso envía un mensaje, pero nunca arriba al receptor. El canal puede haber eliminado el mensaje, puede ser que la red esté saturada y que un enrutador lo eliminó porque sus recursos están al límite. O puede ser que cuando el mensaje estaba en tránsito se cayó un enlace y la red está en un ciclo infinito tratando de recuperarse causando que ese mensaje no llegue al destino. Para manejar las fallas de omisión del canal, por lo general, se utilizan temporizadores y/o cronómetros para determinar el tiempo de arribo del mensaje y generar acciones cuando el tiempo estimado finalice.

Las fallas de omisión en el envío afectan al proceso y son más complicadas de detectar, se dan cuando un proceso completó un evento envío y lo entrega a la capa inferior, o la memoria de salida que no responde adecuadamente porque la memoria está llena o los procedimientos de la capa inferior están detenidos. En este caso el proceso funciona y el canal funciona.

Una falla de omisión de recepción afecta al proceso, el mensaje llega a la memoria de recepción, pero el proceso no lo recibe. Una forma de gestionar este tipo de fallos es almacenar los mensajes que llegan en desorden para que el proceso sea atendido hasta que los mensajes faltantes previos arriben a la memoria.

Fallos arbitrarios o bizantinos

Hay algunas fallas que no se pueden determinar con antelación, cuándo y cómo ocurrirán. Por ejemplo, cuando estoy conectado a Internet y por un lapso muy pequeño de tiempo se detecta una desconexión, sin embargo, no es posible determinar si falló la red o falló el proceso. Las fallas arbitrarias afectan al canal, al proceso o a la memoria de entrada o salida. Estas fallas presentan un comportamiento arbitrario, aleatorio. En el canal se pueden presentar como mensajes arbitrarios de envío y recepción en tiempos aleatorios, o un proceso puede detenerse o realizar una actividad que no estaba definido hacerse. Otro ejemplo de este tipo de falla es la que existe cuando el software funciona bien, pero luego en producción existe una violación de acceso a memoria, este tipo de falla es muy complicado de depurar y de anticipar.

Las fallas arbitrarias en los canales de comunicación pueden darse porque el mensaje puede estar corrupto o los mensajes son enviados más de una vez por el proceso. En este caso, una de las soluciones para gestionar la integridad de los mensajes es generar soluciones de verificación de contenidos de los mensajes, por ejemplo, el campo de verificación de cabecera de TCP. La solución para determinar si existen mensajes repetidos o mensajes generados por terceros es agregar un identificación a los mensajes, por ejemplo, un número de secuencia.

Fallas en la sincronización o de tiempo

Las fallas de sincronización o de tiempo afectan principalmente a aquellos sistemas distribuidos cuyo objetivo es el intercambio de tráfico multimedia,

debido a los requisitos de latencia que deben ser logrados para tener una calidad de servicio adecuada.

Las fallas de tiempo pueden ser del reloj y de rendimiento. De reloj, cuando el reloj físico local se desacopla del reloj del proceso. Y las de rendimiento, que afectan al canal cuando un mensaje se demora más de lo previsto.

6.4.3. Modelo de seguridad

Los modelos de seguridad tienen que ver con tres conceptos relacionados con la seguridad en una red de computadores: autenticación, integridad y confidencialidad. La autenticación define las estrategias para que el emisor y el receptor tengan una alta confianza de que son las entidades que deben ser. La integridad está relacionada con que los mensajes no sean alterados cuando están en tránsito, y la confidencialidad, con la capacidad de la comunicación que los mensajes solo puedan ser entendidos por el emisor y el receptor.

Los modelos de seguridad tratan de asegurar los procesos, los mensajes y los canales de comunicación, y cómo se pueden lograr en un sistema distribuido. Cuando hablamos de autenticación en un sistema distribuido, involucra proteger al acceso a los objetos o recursos a través del permiso de acceso y la autenticación de nodos. Por lo general, un cliente tiene un acceso garantizado a ciertos objetos o recursos y, a su vez, el sistema distribuido debe garantizar los derechos de lectura y escritura relacionados con el nivel de acceso del cliente. Un ejemplo de permisos y derechos de acceso lo tenemos con los sistemas operativos donde el superusuario del sistema tiene derecho a modificar todos los archivos y procesos del sistema, mientras que un usuario normal tendrá un acceso más restringido limitándolo a sus recursos y a aquellos recursos de otros usuarios que permitan un acceso compartido.

Cuando hablamos de integridad en sistemas distribuidos, se debe proteger a los procesos e interacciones de amenazas, por ejemplo, de mensajes de solicitud y respuesta enviados a través de un ataque de hombre en el medio. Y también de mensajes que vienen de orígenes autorizados,

pero que han sido modificados en el canal de comunicación mientras transitaban por la red.

Cuando hablamos de confidencialidad, el sistema puede usar técnicas de criptografía para cifrar los mensajes desde el mismo evento de autenticación. Para lograr autenticación puede usar la criptografía dada por los canales de comunicación usando las interfaces de las capas inferiores, o puede implementar sus propias técnicas de encriptación.

Los modelos de seguridad dependerán del equilibrio entre costo beneficio, las técnicas de seguridad involucran el uso de recursos en los equipos. Algunos sistemas distribuidos, como por ejemplo aquellos que usan equipos con pocos recursos como una red de sensores, tendrán que analizar si los datos que envían deben ser encriptados y si los nodos deben ser autenticados. Es posible que el hardware y software, en los sensores, no tengan la capacidad de memoria y procesamiento para soportar los algoritmos de seguridad. Mientras que otros sistemas distribuidos, como por ejemplo el de la bolsa de valores, deben, en forma obligada, implementar las técnicas y algoritmos para la autenticación, integridad y confidencialidad que esta aplicación requiere.

Recursos:

Le invitamos a ver el siguiente video del profesor Ernesto Jímenez ([Cedia.tv - Model Process](#)), donde explica los tipos de fallas existentes en sistemas distribuidos. Con esta información y la de esta unidad, realice un mapa conceptual sobre las fallas existentes en sistemas distribuidos.



Actividades de aprendizaje recomendadas

Es importante que en esta semana usted realice las siguientes actividades para consolidar su aprendizaje:

- Leer y familiarizarse con el material didáctico de la asignatura.
 - Lea nuevamente los contenidos en esta guía y cree un mapa conceptual.
 - Consulte el plan docente donde se muestra la secuencia de estudio de los contenidos, así como las actividades que

complementan el aprendizaje organizadas por fechas, con su respectiva valoración, rúbricas de evaluación y distribución de las calificaciones por bimestre.

- El entorno virtual de aprendizaje es el medio de interacción permanente entre docente y estudiante, donde encontrará orientaciones semanales, recursos, herramientas de interacción para con sus compañeros y docente.
- Busque sistemas distribuidos comerciales que usen un modelo de arquitectura basado en la capacidad de los clientes y marque sus diferencias y ventajas.
- Revise y profundice en los conceptos de la presente semana a través de la lectura de los cursos abiertos presentados como recursos.
- Hacer un listado de 5 fallas que puedan ocurrir en cada elemento del sistema distribuido e identifique si son catalogadas como fallas de omisión, de tiempo o arbitrarias.
- Haga un listado de sistemas distribuidos que requieran implementar diferentes niveles de seguridad.

Nota: conteste las actividades en un cuaderno de apuntes o en un documento Word.

A continuación, se invita a que revise cómo van sus conocimientos haciendo la respectiva autoevaluación.



Autoevaluación 6

- 1. El modelo de la arquitectura de un sistema distribuido tiene relación con:**
 - a. Muestra cómo se conectan los procesos, los nodos y cómo se pueden nombrar cada uno de estos en el sistema distribuido.
 - b. Muestra la composición de los elementos de la arquitectura y los patrones de la arquitectura.
 - c. Permite abstraer cómo interactúan los componentes, cómo pueden fallar los elementos y cómo el sistema distribuido puede resolverlos.
- 2. ¿Qué modelo representa a través de abstracciones que capturan la composición del hardware en términos de computadores y sus redes interconectadas?**
 - a. Físico.
 - b. Fundamental.
 - c. De la arquitectura.
 - d. Modelo entidad-relación.
- 3. ¿Cuál de los siguientes NO es parte del modelo fundamental?**
 - a. Modelo de interacción.
 - b. Distribución vertical.
 - c. Modelo de fallas.
 - d. Modelo de seguridad.
- 4. ¿Cuáles son las fallas más complejas de gestionar?**
 - a. Caída total.
 - b. Bizantina.
 - c. Fallas de sincronización.
 - d. Fallos integridad.

- 5. Seleccione las características de la primera generación de los sistemas distribuidos:**
- a. Es una red mundial.
 - b. Apareció en la década de los 70 y 80.
 - c. La cantidad de equipos conectados están en la magnitud de decena.
 - d. Apareció en la década de los 60 y 70.
- 6. Seleccione las características de la segunda generación de los sistemas distribuidos:**
- a. Es una red mundial.
 - b. Apareció con el Internet
 - c. La cantidad de equipos conectados están en la magnitud de decena.
 - d. Apareció en la década de los 60 y 70.
- 7. Seleccione cuál de los siguientes NO tiene relación con los patrones de la arquitectura:**
- a. Entidades de comunicación.
 - b. Roles y responsabilidades.
 - c. Distribución horizontal.
 - d. Ubicación.
- 8. Seleccione cuál de los siguientes es usado para la comunicación entre procesos:**
- a. Socket IP.
 - b. Socket TCP.
 - c. Comunicación indirecta.
 - d. Invocación remota.
- 9. Las entidades vistas desde una perspectiva orientada al problema pueden ser:**
- a. Procesos.
 - b. Objetos.
 - c. Componentes.
 - d. Servicios web.

10. ¿Los servicios web son más accesibles que los componentes por que usan protocolos estándares?

- a. Verdadero
- b. Falso

[Ir al solucionario](#)

Resultado de aprendizaje 5

- Describe tres de los principales problemas asociados con la práctica de almacenamiento de datos actual.

Con este resultado de aprendizaje podrá entender claramente los conceptos de sincronización, exclusión mutua y consenso, procesos necesarios y obligatorios para la gestión de recursos en un sistema distribuido, incluyendo el almacenamiento distribuido y centralizado.

En esta unidad se mostrará cómo los diferentes conceptos, algoritmos y componentes se relacionan entre sí para mostrar una forma ordenada y clara con el objetivo de modelar sistemas distribuidos.

Se recomienda implementar los algoritmos presentados para el consenso y la sincronización.

Contenidos, recursos y actividades de aprendizaje recomendadas



Semana 14

Unidad 7. Sincronización, coordinación y consenso

En esta unidad se analizan tres conceptos importantes en sistemas distribuidos, como son la sincronización, la coordinación y el consenso.

Los procesos en sistemas distribuidos tienen que consensuar una acción o el valor de un recurso, y lo tienen que hacer únicamente a través del intercambio de mensajes. Para hacerlo deben sincronizar sus acciones, deben coordinarlas para que cumplan un orden que puede ser parcial o total.

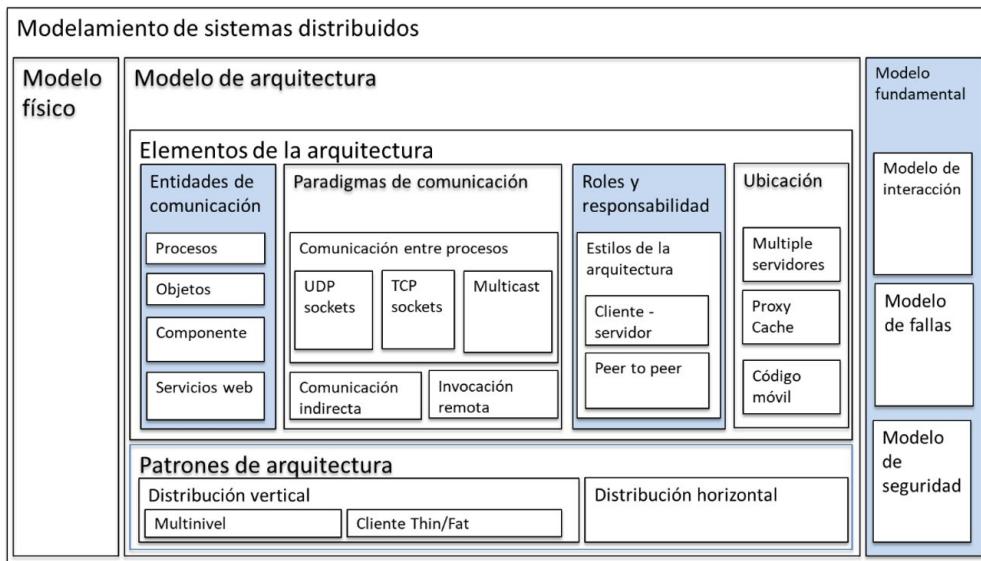
7.1. Contexto de la coordinación y el consenso

En un sistema distribuido, las acciones que se pueden implementar para lograr una coordinación son aquellas relacionadas con el concepto de exclusión mutua, de tal forma que dos o más procesos no puedan cambiar

un recurso al mismo tiempo. Existen algunos mecanismos para lograrlo, por ejemplo, a través de un proceso líder que coordine los cambios en los recursos y los procesos. De la misma manera, para consensuar se deben analizar algoritmos para acordar cambios, inclusive frente a la presencia de procesos fallidos o fallas que se pueden dar en el canal de comunicación.

Figura 58.

Contexto de la coordinación y el consenso



Nota. Elaborado por el autor

Como se puede ver en la Figura 58, las acciones de coordinación y consenso tienen que ver con las entidades de comunicación, con los roles y responsabilidades altamente relacionados con la elección del líder y con los modelos fundamentales, con los modelos de interacción por el intercambio de mensajes coordinados, también con los modelos de fallos por la posibilidad de acordar un valor a pesar de los fallos que puedan existir. Y con el modelo de seguridad, por la necesidad de validar la identificación de los procesos líderes.

En el apartado 5.4 Sincronización de relojes físicos, analizamos los algoritmos de sincronización de reloj donde algunos de estos utilizan a un proceso líder para coordinar y consensuar el tiempo, pero aún no se analizó el proceso de elección de líder. Como ejemplo de coordinación, en los tableros de noticias como Twitter la respuesta a un tweet debe mostrarse después del tweet original. El ejemplo de exclusión mutua se puede ver en

las bases de datos donde se debe bloquear registros mientras se están actualizando.

7.2. Algoritmos de coordinación

Los algoritmos de coordinación son fundamentales en sistemas distribuidos, entre otras cosas, sirven para, dinámicamente, reasignar el rol de líder cuando el líder principal se ha detenido y, por lo tanto, se debe mantener el acceso a los recursos; se los usa también cuando existen actualizaciones concurrentes a las bases de datos o para acordar o consensuar acciones.

Los algoritmos de coordinación deben afrontar algunos retos en sistemas distribuidos donde tener una solución centralizada no es apropiado porque si un servidor decide el acceso a recursos, puede presentar dos problemas, convertirse en un cuello de botella al existir solo un equipo respondiendo a todas las peticiones y ser un único punto de fallo.

Otro reto que considerar son las características de las topologías de red y los problemas de conectividad en la red.

Las fallas deben ser toleradas, se debe considerar que los procesos son autónomos y, dada su naturaleza, pueden fallar en forma aislada y afectar a todo el sistema; inclusive los enlaces pueden fallar.

En los sistemas distribuidos también se debe afrontar la idea de que algunos resultados esperados simplemente no pueden darse. Imagine que un grupo de amigos ha acordado una reunión a una hora determinada, posiblemente la mayoría tendrá diferente tiempo en su reloj, otros posiblemente no lean el mensaje o lo lean muy tarde, y, por lo tanto, el objetivo de estar a una misma hora todos es imposible de lograr.

La coordinación y consenso en un sistema distribuido en una comunicación síncrona se pueden lograr a través de la implementación de un detector de fallos que considere una aproximación del tiempo que le toma a los procesos ejecutarse, enviar los mensajes y donde los relojes locales se pueden sincronizar. Por otro lado, en un sistema distribuido asíncrono, que es el caso más común, donde no existen tiempos indeterminados para la ejecución de los procesos o la recepción de mensajes, la coordinación y el consenso es una tarea compleja.

A continuación, se detallan algunos algoritmos importantes en el proceso de coordinación de sistemas distribuidos:

7.2.1. Tópicos de coordinación

Cuando se intenta coordinar acciones o eventos en sistemas distribuidos, se pueden presentar algunos problemas cuando queremos realizar los procesos de elección de líder, cuando queremos hacer una exclusión mutua o cuando se desea consensuar o acordar un valor.

Un sistema distribuido entra en un proceso de elección de líder por eventos de inicio del sistema, o porque el líder tuvo un fallo y se detuvo, o porque la red tuvo una partición o una reconfiguración.

Una partición de la red se da cuando, por un problema en el canal de comunicación, el grafo que la conforma queda dividido en dos o más subconjuntos totalmente independientes. Por ejemplo, imagine que existe un equipo en Internet donde funciona un proceso que permite sincronizar el tiempo en todo el mundo. Suponga ahora que el cable submarino que une América y Europa sufre un desperfecto. Ahora la red tiene dos partes independientes, los equipos en cada una de las partes van a iniciar un proceso de elección de líder. Una vez que el canal de comunicación se restablezca debido a la reconfiguración de la red, existirá nuevamente un proceso de elección.

Un acceso a un recurso compartido de un sistema distribuido se lo hace a través de un recurso denominado zona crítica en donde los procesos, únicamente a través de mensajes, pueden modificarlo. En un sistema distribuido el acceso a la zona crítica o de exclusión mutua se debe coordinar que proceso o procesos tienen acceso exclusivo al recurso e informar al resto de procesos cuando la zona crítica esté disponible o la exclusión del recurso ha cambiado.

Consenso significa que todos o la mayoría de los procesos, dependiendo del nivel de consenso deseado, van a acordar un valor. Puede ser un desafío de coordinación en sistemas distribuidos dependiendo de los tipos de fallas que se den en los procesos, canales o memoria de entrada y salida. Por ejemplo, si existe una falla-parada, el proceso p falló se supone que p no va a participar en el consenso, por otro lado, si tenemos una falla arbitraria o bizantina en un proceso q en este caso no ha fallado el proceso, pero los valores generados por q afectan al consenso en el sistema distribuido.

7.2.2. Coordinación y fallas

Para poder entender los algoritmos de elección de líder, exclusión mutua y consenso vamos a suponer las siguientes afirmaciones:

- a. Los enlaces son confiables, esto significa que suponemos que la red no fallará y que eventualmente la red entregará los mensajes.
- b. Los procesos pueden fallar.
- c. El sistema distribuido tendrá que implementar un servicio de detección de fallas.
 - Los procesos envían un saludo cada cierto tiempo al detector de fallas.
 - El detector de fallas registra los mensajes de respuesta enviados por los procesos no fallidos.

Un detector de fallos en sistemas distribuidos asíncronos es poco confiable debido a los tiempos de respuesta indeterminados. Un proceso envía una respuesta al detector de fallos, el cual lo coloca en el conjunto de procesos sanos o no fallidos, luego de enviar la respuesta el proceso falla, en este caso el conjunto de procesos sanos es inconsistente.

Para los algoritmos que vamos a analizar, podemos mencionar que los procesos debido a las fallas pueden ser:

- Sospechosos.
- No sospechosos.
- Fallidos.

Un proceso es sospechoso cuando no ha existido una comunicación, pero no porque el proceso ha fallado sino más bien porque la respuesta se demora demasiado en llegar. Nuevamente tomemos el caso de la reunión de amigos a una hora determinada, donde cada amigo es un proceso, uno de ellos se demora en confirmar, eso no significa que no se va a reunir, simplemente involucra que el mensaje de respuesta no ha llegado, puede ser que no ha leído aún sus mensajes en su celular.

Un proceso es no sospechoso, sin embargo, no hay garantía de que este no haya fallado desde el último mensaje enviado por él. En el mismo ejemplo de la reunión, aunque quedaron en reunirse a las siete, uno comenta que llegará a las ocho, sin embargo, puede ser que algo le sucede a las 07h30.

En este caso no es sospechoso en el lapso desde su mensaje de respuesta hasta las 08h00.

Un proceso fallido es aquel proceso que se ha determinado realmente que falló.

7.2.3. Propiedades esperadas de los algoritmos de coordinación

Podemos analizar los algoritmos de elección de líder, exclusión mutua y consenso basado en sus propiedades cualitativas y cuantitativas. Las propiedades cualitativas determinan una forma subjetiva de evaluar a los algoritmos, mientras que las cuantitativas miden objetivamente un criterio de acuerdo a los valores esperados.

Las propiedades cualitativas son las de seguridad (Safety) y de vitalidad (Liveness). Un algoritmo satisface la propiedad de seguridad cuando garantiza que algo no deseado o malo no pasará nunca. En otras palabras, si hay un resultado este satisface la especificación del algoritmo. Por ejemplo, una persona inocente nunca será encarcelada.

La propiedad de vitalidad de un algoritmo garantiza que algo eventualmente pasará. Existirá un resultado. Eventualmente involucra que no está atado a un tiempo, si se deja correr el algoritmo el tiempo suficiente al final existirá un resultado. Por ejemplo, en una votación un representante será eventualmente elegido.

En las propiedades cuantitativas analizaremos el ancho de banda que consumen los algoritmos basado en la cantidad de mensajes necesarios enviados y la cantidad de pasos necesarios que necesita el algoritmo para finalizar.

7.3. Algoritmos para elección de líder

Los algoritmos de elección de líder en un sistema distribuido deben asegurar la propiedad de seguridad de tal forma que cada proceso conocerá y registrará al líder. También deben cumplir con la propiedad de vitalidad, de tal forma que todos los procesos participarán y eventualmente descubrirán al líder.

Para lograr la elección de líder, los algoritmos en un sistema distribuido, los N procesos, deben tener una identificación, sin embargo, esta identificación

puede no ser única, por ejemplo, piense en un grupo de personas donde algunos de ellos pueden compartir el mismo nombre. Estos procesos deben seleccionar un líder de entre ellos. Los procesos pueden simultáneamente llamar a un proceso de elección o el proceso de elección se hará luego de que una falla ha ocurrido.

La elección involucra que cada proceso debe tener una forma de identificación única, debe tener una variable para almacenar la identificación del líder cuando sea elegido o la variable tendrá un valor indeterminado cuando el líder aún no se ha definido.

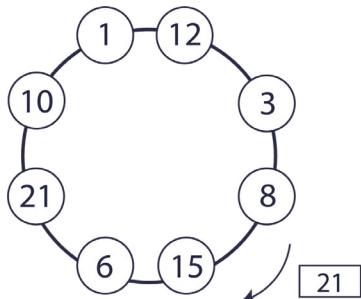
Es momento de analizar algunos tipos de algoritmos de elección:

7.3.1. Algoritmo de elección basado en anillo

En este algoritmo los procesos se organizan en una forma de anillo, indiferentemente de la tecnología utilizada. En la Figura 59, cada proceso está unido al anillo, tiene un identificador único, tiene un canal de comunicación con dos procesos el anterior y el posterior. Los mensajes se pasan en un sentido de reloj de proceso a proceso.

Figura 59.

Algoritmo de elección de líder basado en anillo



Nota. Elaborado por el autor

El objetivo de este algoritmo es elegir como líder al proceso que tiene el identificador más alto. El identificador puede ser un valor simple, como la dirección IP, o puede ser un valor compuesto como, por ejemplo, la dirección IP más un número de proceso e inclusive la carga del equipo o el ancho de banda del canal donde el proceso reside. Además, el identificador puede ser creado dinámicamente.

Los procesos pueden enviar dos tipos de mensajes, $elegir(UID)$ y $elegido(UDI)$, además, los procesos pueden tener dos estados: no-participante y participante. Cuando arranca un proceso de elección, un proceso inicia en un estado no-participante.

La elección de líder se hará en dos fases: determinar el líder y anunciar ganador. La fase de **determinar el líder** se inicia en un proceso, p , que cambia su estado a *participante* y envía el mensaje de $elegir(k)$, donde k es el identificador del propio proceso. El mensaje es enviado a su vecino en un sentido de reloj. Cuando un no-participante q recibe un mensaje $elect(k)$, compara su identificador con el que llegó en el mensaje, selecciona el mayor $k = \max[k, k_q]$ y reenvía el mensaje $elegir(k)$ a su vecino para finalmente cambiar su estado a *participante* y ser considerado en el proceso de elección.

Cuando un proceso, r , participante recibe un mensaje $elegir(k)$, determina si su identificador k_r es mayor o menor k ; si es mayor envía a su vecino su identificador en el mensaje $elegir(k_r)$, y si es menor ignora el mensaje.

La fase de **anunciar al ganador** empieza cuando ya se ha determinado el líder. Cuando un proceso participante, l , recibe un mensaje $elegir(k)$ y el identificador es el mismo que el suyo $k_l == k$, entonces se ha encontrado el líder, el proceso l pasa a ser no-participante y envía el mensaje $elegido(k)$ a su vecino.

Cuando un proceso participante recibe un mensaje de $elegido(k_l)$, almacena y registra el identificador del proceso líder k_l , pasa su estado a no participante y reenvía el mensaje $elegido(k_l)$ a su vecino.

Cuando un proceso no-participante recibe el mensaje $elegido(k_l)$, simplemente registra el identificador del líder y reenvía el mensaje $elegido(k_l)$ a su próximo vecino.

El algoritmo de elección basado en anillo cumple con la propiedad de seguridad, si los enlaces no fallan, todos los procesos van a tener un líder o van a participar de la elección. También cumple con la propiedad de vitalidad y es claro verificar cuándo solo un proceso de elección está ejecutándose, si hay varios procesos, de cualquier manera existirá un solo líder en el sistema.

Con las propiedades cuantitativas, el peor caso se da cuando un proceso inicia la elección y el proceso con el identificador más alto es el próximo

anterior. En este caso necesitaremos al menos $3n-1$ mensajes y de pasos para finalizar el proceso.

Para que este algoritmo funcione correctamente se debe asumir que no hay fallas en los procesos, ni en los canales. Si existen fallas, el algoritmo no funciona. En este caso el detector de fallos debe determinar los procesos fallidos y correctos antes de una elección.

7.3.2. Algoritmo de bully

El algoritmo de bully (Garcia-Molina, 1982) tiene como objetivo que el proceso con el identificador más alto sea elegido coordinador. Este algoritmo tolera las fallas en los procesos. Cada proceso tiene un identificador único y conoce el identificador de los otros procesos, por lo tanto, debe existir una etapa o configuración de descubrimiento inicial donde todos los procesos envíen y conozcan sus identificadores al resto de procesos. Esta etapa de descubrimiento puede tomar un tiempo considerable de acuerdo a la cantidad de procesos que existan en el sistema, es aplicable cuando los identificadores de los procesos no cambian. Este algoritmo, en un sistema síncrono, debería tener la capacidad de detectar fallos y si no existe una respuesta a una solicitud, por lo general usando temporizadores y límites de tiempo a los procesos.

En este tipo de algoritmo, un proceso detecta que el proceso líder falla porque no tuvo una respuesta a un mensaje o porque el conteo de un temporizador finalizó. En este caso, el proceso inicia una etapa de elección enviando los identificadores de los potenciales candidatos a líder, los procesos con el identificador más alto. Si el proceso no recibe respuesta de los candidatos el proceso se declara el ganador de la elección y se proclama líder. Si se genera una respuesta, el proceso finaliza su etapa de elección. Cuando un proceso recibe una notificación, este responde al emisor e inicia su etapa de elección. El algoritmo utiliza tres tipo de mensajes:

- Mensaje de elección: enviado a los nodos candidatos con identificadores más altos.
- Mensaje de respuesta: al proceso emisor y a los nodos con identificadores más bajos.
- Mensaje del coordinador: enviado por el proceso que está actuando o ha sido elegido como líder.

Hemos preparado un recurso digital en donde, en forma interactiva, podrá entender de mejor forma cómo funciona el algoritmo de bully.

Elección de líder – Algoritmo de Bully.

Este algoritmo cumple con la propiedad de vitalidad debido a la suposición de sincronización, que se puede lograr con temporizadores. Con respecto a la propiedad de seguridad, si el grupo de procesos es estable, la cumple claramente. Sin embargo, si el grupo no es estable o si une un proceso y se declara como líder durante una etapa de elección, esta propiedad no está garantizada. En este caso, se necesitará implementación de un reloj lógico vectorial para garantizar la propiedad de seguridad.

Con respecto a las propiedades cuantitativas, se las puede definir para el mejor caso cuando el proceso con el segundo identificador más alto detecta la falla, y el peor caso, cuando el proceso con el identificador más bajo detecta la falla. Con respecto al ancho de banda, en el mejor caso, se necesitará $(N-2)$ mensajes de coordinador enviados desde el proceso con el segundo mejor identificador, mientras que en el peor caso se necesitará una magnitud de orden cuadrático $O(N^2)$. Con respecto a los pasos, en el mejor caso se necesitará simplemente un mensaje, y en el peor caso se necesitará $(N-1)$ pasos.

Recursos:

En el REA Curso Abierto de Sistemas Distribuidos de la Universidad Carlos III de Madrid (López Fuentes, 2015), específicamente en el tema 7 ([Tema 7 - Coordinación y sincronización distribuida – \(uc3m.es\)](#)) existe un enfoque similar de esta unidad donde detallados los dos algoritmos de elección de líder estudiados, el de bullying (matón) y el de anillo, invito a hacer un cuadro comparativo de lo analizado de tal forma que se amplíen sus conocimientos.



Actividades de aprendizaje recomendadas

Es importante que en esta semana usted realice las siguientes actividades para consolidar su aprendizaje:

- Leer y familiarizarse con el material didáctico de la asignatura.

- Lea nuevamente los contenidos de esta guía y cree un mapa conceptual.
 - Consulte el plan docente donde se muestra la secuencia de estudio de los contenidos, así como las actividades que complementan el aprendizaje organizadas por fechas, con su respectiva valoración, rúbricas de evaluación y distribución de las calificaciones por bimestre.
 - El entorno virtual de aprendizaje es el medio de interacción permanente entre docente y estudiante, donde encontrará orientaciones semanales, recursos, herramientas de interacción para con sus compañeros y docente.
- Consultar qué sistemas distribuidos para middleware están disponibles en el mercado y, de acuerdo a las características de estos, determinar en una tabla comparativa los elementos de la arquitectura que los componen, así como también qué modelo de arquitectura utilizan.
- Cree un diagrama de flujo de datos para el algoritmo de elección de líder basado en anillo.
- Cree un diagrama de flujo de datos para el algoritmo bully para elección de líder.
- En el reloj distribuido que se ha desarrollado hasta el momento, incluya un componente para la elección de líder usando los dos algoritmos: basado en anillo y el algoritmo de bully.

Nota: conteste las actividades en un cuaderno de apuntes o en un documento Word.



Semana 15

7.4. Algoritmos para exclusión mutua

La exclusión mutua es otro de los problemas de coordinación que deben afrontar los sistemas distribuidos. Trata de asegurar que dos o más

procesos no accedan a un recurso compartido simultáneamente. A nivel del sistema operativo, por ejemplo, para acceder a la memoria o al procesador se lo hace a través de interrupciones, mientras que en un sistema distribuido, los procesos deben negociar el acceso a esa zona crítica (ZC) únicamente a través del intercambio de mensajes.

Para poder diseñar y analizar los algoritmos de exclusión mutua, haremos las siguientes suposiciones:

- N procesos asíncronos, que por simplicidad suponemos que no fallan.
- El canal de comunicación es confiable, por lo que el envío de mensajes es confiable
- Para ejecutar o usar la zona crítica cada proceso puede usar las siguientes primitivas: *ingresar()*, *salir()*, *accederRecurso()*

Las propiedades que deben garantizar los algoritmos de exclusión mutua y que usaremos para analizarlos son:

- Seguridad: solo un proceso debe estar en la sección crítica al mismo tiempo.
- Vitalidad: las solicitudes para entrar y salir eventualmente están garantizados, la zona crítica no debe quedar bloqueada por algún proceso fallido o mensaje que no llegó.
- Ordenamiento: el ingreso a la zona crítica puede ser a través de una política FIFO (primero entrar, primero en salir) de acuerdo al orden de causalidad de eventos definido por Lamport y que revisamos en el apartado de reloj lógico.

Le invito a revisar a continuación algunos algoritmos para exclusión mutua:

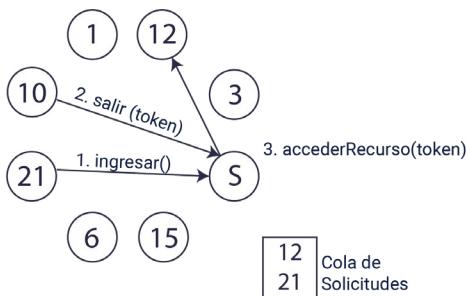
7.4.1. Servidor central

El primer algoritmo está basado en un servidor central que gestiona el acceso al recurso compartido. Para ingresar a una sección crítica, a un proceso, envía una solicitud al servidor *ingresar()*. De acuerdo al estado de la cola de peticiones, el servidor responde con un token al proceso cuando la zona crítica no está siendo usada. Cuando el proceso ejecuta la primitiva *salir()*, devuelve al servidor el token informando que ya no está ocupando la

zona crítica. Si algún proceso está ocupando el token y llega al servidor una nueva petición *ingresar()* de otro proceso, entonces el servidor toma esta nueva petición y la coloca en la cola. Cuando la zona crítica es liberada, se atiende en un orden FIFO al resto de procesos en la cola y se informa al proceso siguiente.

Figura 60.

Exclusión mutua usando servidor central



Nota. Elaborado por el autor

En la Figura 60 se presenta un ejemplo de exclusión mutua, un proceso S previamente ha sido elegido como coordinador, tiene una cola de procesos para gestionar la distribución del token y, por lo tanto, el acceso a la sección crítica. El proceso 10, actualmente, está usando la zona crítica. El proceso 21 desea acceder a la zona crítica enviando un mensaje *ingresar()* al proceso servidor, pero como el token está ocupado su solicitud pasa al final de la cola de solicitudes. Una vez que el proceso 10 libera el token, informa al servidor con la primitiva *salir(token)*, el servidor, entonces, garantiza el acceso al proceso que está primero en la cola de solicitudes en este caso el proceso 12.

Si no ocurren fallas, este algoritmo garantiza las propiedades de seguridad y vitalidad. Sin embargo, el orden no va a ser mantenido debido a los tiempos de envío y recepción de mensajes diferentes que se pueden dar entre procesos. Una desventaja de este algoritmo está dada porque el servidor central se convierte en un cuello de botella y, por lo tanto, un simple punto de falla. Para garantizar el orden se puede implementar un reloj lógico vectorial.

La propiedad de seguridad está garantizada por el uso del token, que no permite que dos procesos estén ocupando la zona crítica simultáneamente. La vitalidad está garantizada mientras los procesos no fallen.

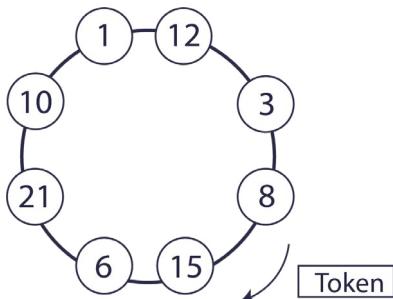
El ancho de banda de un proceso, medido en la cantidad de mensajes, será tres, dos mensajes para solicitar y uno liberar. Un proceso tendrá que esperar por ser atendido el tiempo directamente proporcional a la longitud uso de la cola. Si no hay ningún proceso en la cola usando la zona crítica, una petición será atendida inmediatamente.

7.4.2. Algoritmo basado en anillo

En el algoritmo basado en anillo, Figura 61, los procesos son distribuidos en un anillo lógico. En forma similar al algoritmo basado en servidor central, se usa un token para garantizar el acceso a la zona crítica. El token es pasado en el anillo de un proceso a otro en orden de las manecillas de reloj. Cuando un proceso recibe el token puede ingresar a la zona crítica, si el proceso no requiere usar la zona crítica pasa inmediatamente el token al próximo proceso.

Figura 61.

Exclusión mutua basado en anillo



Nota. Elaborado por el autor

Este algoritmo también logra cumplir con la propiedad de seguridad porque solo un proceso puede acceder a la zona crítica y con la propiedad de vitalidad, porque el token eventualmente será liberado. Sin embargo, no garantiza un orden porque no necesariamente los requerimientos de los procesos para el acceso a la zona crítica serán en el orden del anillo, solamente el acceso a la zona crítica está directamente relacionado con el orden lógico del anillo. Además, utiliza una cantidad significativa de ancho de banda porque siempre el token va a estar viajando por los procesos, indiferentemente de que requiera acceso a la zona crítica.

7.4.3. Algoritmo de exclusión multicast

El algoritmo de exclusión mutua en multicast (Ricart & Agrawala, 1981) utiliza las ventajas de las comunicaciones multicast para mejorar en el uso de ancho de banda y la cantidad de pasos para lograr la exclusión mutua.

Trabaja sobre una cantidad de N de procesos interconectados asíncronos donde cada uno tiene un identificador único en el sistema distribuido y un reloj lógico para gestionar el orden de los eventos.

Los procesos envían un mensaje multicast al resto de procesos solicitando el acceso a la zona crítica, incluyendo una estampa de tiempo del reloj lógico y el identificador del proceso. El acceso garantizado a la zona crítica del proceso, cuando existen solicitudes simultáneas, se resuelve con la inclusión de la estampa de tiempo de tal forma que satisface la propiedad de ordenamiento por el uso del reloj lógico vectorial.

Si la tecnología subyacente soporta multicast, un proceso envía un solo mensaje al resto de procesos, si no soporta multicast se puede simular. En este sentido, por ejemplo, si el sistema distribuido está conformado por cinco procesos y la tecnología no soporta multicast, se debe enviar cuatro diferentes mensajes, uno para cada uno de los otros procesos.

Un proceso puede estar en tres estados diferentes y se debe registrar su estado en una variable, los estados pueden ser:

- RELEASED: el proceso está fuera de la zona crítica.
- WANTED: el proceso trata de ingresar a la zona crítica.
- HELD: cuando el proceso está usando la zona crítica.

Cuando un proceso inicia su estado es RELEASED, si el proceso necesita la zona crítica cambia su estado a WANTED y envía por multicast la solicitud con la estampa de tiempo a todos los procesos. Luego, espera hasta que el resto de los procesos pertenecientes al grupo respondan a su solicitud, pasa su estado a HELD y puede usar la zona crítica.

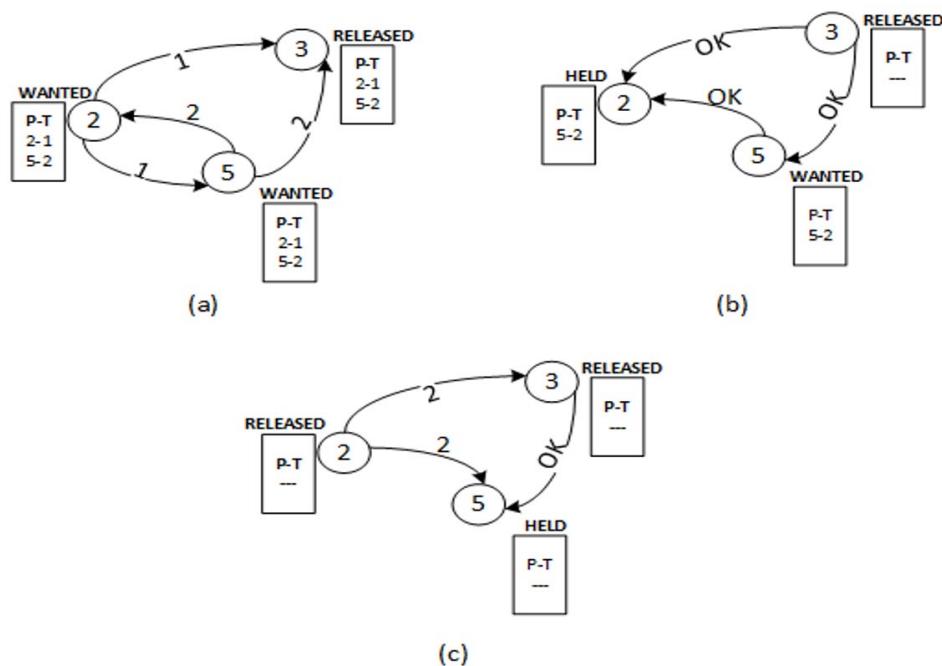
Mientras el proceso esté usando la zona crítica HELD, puede recibir una solicitud de otro proceso h , en este caso coloca esta solicitud en la cola de solicitudes del proceso sin enviar ninguna respuesta. Si el proceso está en RELEASED, envía inmediatamente la solicitud al proceso h .

Si el proceso está buscando ingresar a la zona crítica WANTED y recibe una solicitud de otro proceso h con un estampa de tiempo mayor que aquella que envió, entonces almacena la solicitud del proceso h en su cola de mensajes sin enviar ninguna respuesta, caso contrario si el proceso h tiene una estampa menor de tiempo se envía a h inmediatamente una respuesta.

Cuando el proceso desea salir de la sección crítica, cambia su estado a RELEASED y envía mensajes de respuesta encoladas a los procesos correspondientes.

Figura 62.

Exclusión mutua multicast



Nota. Elaborado por el autor

En la Figura 62 hay tres procesos el 2, 5 y 3 cada uno mostrando la cola de peticiones, cuando inician los procesos todos empiezan con el estado RELEASED, en la Figura 62(a) dos procesos, 2 y 5, desean usar simultáneamente la zona crítica y cambian su estado a WANTED, envían su solicitud con su respectiva estampa de tiempo a todos los procesos en forma de multicast, en cada proceso con la estampa de tiempo se ordenan y se almacenan las peticiones en su respectiva cola. En la Figura 62(b) el proceso 3, como no requiere la zona crítica y responde a los dos procesos que está de acuerdo con que usen la zona crítica, el proceso 5 responde al

proceso 2 el cual tiene la mejor estampa de tiempo cambiando su estado a HELD, se garantiza al proceso 2 el acceso a la zona crítica. En la Figura 62(c) el proceso 2 sale de la zona crítica y envía a todos los procesos la siguiente petición que es la del proceso 5. Una vez que el proceso 5 la recibe, cambia su estado a HELD y se le garantiza el acceso a la zona crítica.

Existen algunas mejoras a este algoritmo de exclusión mutua, por ejemplo, el presentado por (Maekawa, 1985) el cual tiene como objetivo acelerar el tiempo para acceder a la zona crítica, creando un proceso de votación de los procesos. Utiliza un conjunto de votación buscando que el número suficiente de procesos le asignen la zona crítica. Un ejemplo de la cantidad de procesos votando a favor puede ser un proceso más que la mitad.

7.5. Consenso

Cuando todos los procesos deben acordar un valor o un estado, por ejemplo, lo que hemos visto anteriormente como elegir un líder o acordar qué proceso acceda a la zona crítica o cuándo una base de datos distribuida debe poblar un registro y replicarlo en todo el sistema, es un trabajo de coordinación y consenso.

El consenso lo podemos considerar como un modelo con una colección de procesos que pueden fallar, de tal forma que no se levantan más o pueden presentar aquellas fallas bizantinas que analizamos anteriormente. Esta diferencia nos invita a pensar que existen ahora dos subconjuntos en los procesos, los fallidos y los correctos.

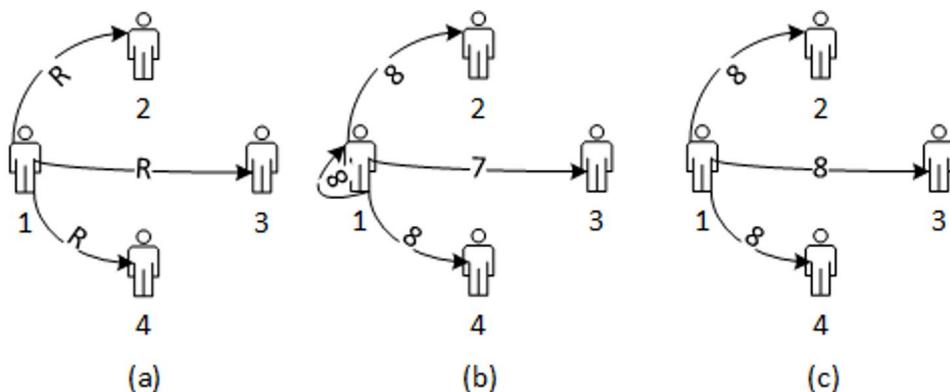
Un proceso de consenso debe tratar de lograr las siguientes propiedades:

- Terminación: en algún momento todos los procesos correctos deciden en algún valor. Entonces el sistema debe implementar una estrategia para confrontar los fallos, por ejemplo, un detector de fallos para determinar los procesos correctos y procesos fallidos.
- Acuerdo: todos los procesos deben acordar en un valor.
- Integridad: si todos los procesos correctos acuerdan un valor, debe ser el mismo valor para todos los procesos.

Los algoritmos de consenso deben considerar un modelo y las fallas que pueden tener. El modelo debe considerar conjunto de N procesos, donde la comunicación se hace exclusivamente a través del paso de mensajes. Vamos a suponer que la comunicación es confiable y el consenso se puede hacer en forma asíncrona y síncrona. Los procesos pueden fallar y quedar en dicho estado o los procesos también pueden mostrar fallas bizantinas, lo que provocaría que los procesos puedan ser considerados como traidores (ataque de hombre en el medio) y que la información que entreguen no sea verdadera (integridad).

En términos generales, los procesos inician con un estado "no decidido" y algún proceso propone un valor inicial desde un conjunto de valores, D , por ejemplo, para una base de datos distribuida un conjunto de valores puede tener los elementos $D = \{\text{abortar}, \text{no abortar}\}$. Entonces los procesos para llegar a un consenso se comunican intercambiando un valor e intentan decidir. Una vez que han decidido, cambian su estado a "decidido", no cambian su decisión salvo que un nuevo consenso se inicie. Los procesos deben alcanzar una decisión indiferente de si ha existido fallas determinantes o fallas arbitrarias o bizantinas.

Figura 63.
Consenso usando secuenciador



Nota. Elaborado por el autor

En la Figura 63 se propone un ejemplo de consenso usando un secuenciador, la persona 1, será la encargada de recopilar y tomar las decisiones. En este caso la persona 1, Figura 63(a) emite un mensaje a un grupo de amigos: "reunámonos". Todos responden en la Figura 63(b) con un valor de la hora en que se pretenden reunir, el secuenciador recopila los

valores y en este caso específico responde con el valor dado por la mayoría, como se puede ver en la Figura 63(c). Pueden existir algunos escenarios probables que deben ser considerados, por ejemplo, qué pasa si alguna de las personas 2 y 4 fallan en el grupo o no pueden ser contactadas, qué valor se debe acordar. De la misma forma qué pasa si esas mismas dos personas, 2 y 4, no contestan hasta las 8, cómo debe proceder el sistema. Qué pasa si los cuatro procesos proponen horas totalmente diferentes.

Para llegar a alguna solución vamos a asumir algunos comportamientos o propiedades, primero vamos a asumir que no hay fallas en los canales de comunicación y que los procesos se envían sus valores a través de multicast, luego se debería esperar hasta que todos los procesos hayan recogido todos los valores, incluido el suyo propio, a continuación, como en el ejemplo anterior, la selección del valor puede ser el más frecuente de un conjunto de valores acordado entre todos, incluyendo el valor "indeterminado". Si se cumplen estos requisitos podemos pensar que los algoritmos podrían cumplir con la propiedad de terminación, si el canal de comunicación es confiable también se cumple la propiedad de integridad y acuerdo. Sin embargo, se debe analizar en el caso de que los procesos fallen, si un proceso falló deja de enviar valores y si es una falla arbitraria los valores que puede enviar pueden ser diferentes inclusive al definido en el conjunto de valores.

7.5.1. Consenso en sistemas síncronos

Llegar a un consenso cuando hay un sistema síncrono se puede lograr debido que se puede determinar los valores máximos que un evento o actividad requiere. Usando multicast básico, significa enmascarar el multicast simplemente enviando el mismo valor con diferentes mensajes a cada uno de los otros procesos en el sistema distribuido, se garantiza la entrega de mensajes a todos los procesos correctos, siempre y cuando el emisor no falle.

En un sistema síncrono se puede admitir que los procesos puedan fallar, si un proceso envía un valor para acordar y después de un tiempo el proceso no responde significa que el proceso ha fallado, aun así, se logra acordar un valor. Sin embargo, si esas fallas son bizantinas, el proceso responde con un valor fuera del conjunto de valores o es un valor sin integridad, el consenso es más complejo de lograr. Si tomamos como ejemplo la Figura 63, una persona usando un grupo de WhatsApp envía una convocatoria a la reunión y les menciona que la respuesta es hasta una hora determinada

(sincronismo), si una persona no responde hasta la hora determinada entonces es análogo a un proceso fallido. Un fallo bizantino sería que una tercera persona tomó el celular de la persona 4 y colocó un mensaje, que, aunque puede estar dentro del conjunto de respuestas es un mensaje malicioso por un ataque de hombre en el medio.

En un sistema síncrono deberíamos analizar también cuántos de todos los procesos pueden fallar, lo denotaremos con f , por ejemplo, en un conjunto de cinco procesos si tres fallan, significa que posiblemente el sistema distribuido falló.

Inicialmente cada proceso propone un valor de un conjunto D, en la Figura 64, por ejemplo, estas cuatro personas se convocan a una reunión y pretenden acordar una hora. En una primera ronda algunos dirán a las 7, otros a las 8 y así respectivamente, definiéndose un conjunto de valores que será registrado por cada persona. En cada ronda este conjunto de valores podrá crecer, sin embargo, en la siguiente ronda solo se enviarán los valores que no se han enviado anteriormente. Cuando estos nuevos valores son recibidos se los almacena en el conjunto de valores. Finalmente, en la ronda $f+1$ cada proceso debe seleccionar un valor del conjunto de valores como una decisión final.

Figura 64.

Algoritmo de consenso de Dolev

Algorithm for process $p_i \in g$; algorithm proceeds in $f + 1$ rounds

On initialization

$Values_i^1 := \{v_i\}$; $Values_i^0 = \{\}$;

In round r ($1 \leq r \leq f + 1$)

$B\text{-multicast}(g, Values_i^r - Values_i^{r-1})$; // Send only values that have not been sent
 $Values_i^{r+1} := Values_i^r$;

while (in round r)

{

On $B\text{-deliver}(V_j)$ from some p_j
 $Values_i^{r+1} := Values_i^{r+1} \cup V_j$;

}

After ($f + 1$) rounds

Assign $d_i = \min(Values_i^{f+1})$;

Nota. Tomado de Coulouris et al. (2012).

En la Figura 64 se muestra un algoritmo propuesto por Dolev & Strong (1983) para consenso de un sistema distribuido síncrono el cual pasaremos a analizar:

- El identificador del proceso, i , es donde se ejecuta este algoritmo, este proceso pertenece a un grupo g para el envío de mensajes multicast.
- Existen en el sistema distribuido N procesos y de estos existen f procesos que pueden fallar.
- Por cada ronda, en el proceso se genera un conjunto de valores V a decidir, por ejemplo, para la ronda 2 el proceso i el conjunto de valores generado se almacena en V_i^2 .
- Cuando se inicia el proceso i tiene unos valores para proponer que son almacenados en el conjunto V_i^1 . También se inicia el conjunto V_i^0 sin ningún valor. De tal forma que en la ronda 1 simplemente se envíen los valores del conjunto V_i^1 .
- El algoritmo de consenso se ejecutará en $f+1$ rondas.
- La primitiva $B\text{-}multicast}(g, V_i^r - V_i^{r-1})$ envía a todos los procesos que pertenecen al grupo g solo los valores que no han sido enviados anteriormente. La variable r es un contador que va de 1 hasta $f+1$.
- Cada ronda debe finalizar, porque existe un temporizador para cada una de las rondas. Si un proceso falló y no entregó a tiempo un valor, entonces la variable f se incrementa 1.
- Si mientras el proceso i está en una ronda r recibe mensajes con un conjunto de valores a acordar de otro proceso con un identificador j , estos valores se agregan al conjunto de valores que va a ser usado en la siguiente ronda, V_i^{r+1} . La primitiva $B\text{-}deliver(V_j)$ es usada para indicar que se ha recibido un conjunto de valores del proceso j .
- Finalmente, luego de $(f+1)$ rondas se asigna un valor de decisión que será el mínimo del último conjunto de valores, V_i^{f+1} . El proceso i cambia a estado *decidido*.

7.5.2. Consenso con fallas bizantinas

Hasta ahora hemos analizado el consenso cuando se puede determinar que un proceso ha fallado, sin embargo, qué se puede hacer con aquellos

errores que son arbitrarios con esas fallas donde no se puede determinar si son errores específicamente, aquellos procesos que pueden enviar cualquier valor en cualquier momento o pueden simplemente no enviar un mensaje.

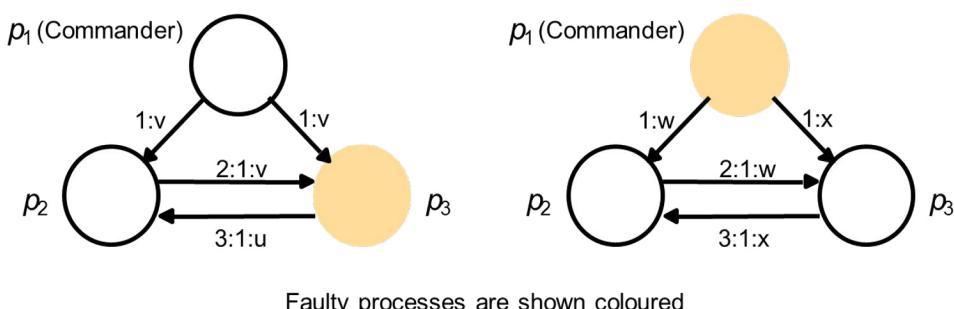
Algunos de estos procesos se analizan a continuación:

7.5.2.1. Generales bizantinos

El problema de los generales bizantinos (Lamport et al., 1982) se describe como un tema de estudio para analizar el consenso en sistemas distribuidos cuando las fallas son aleatorias. Existen tres generales de donde uno da la orden, cada uno con una división del ejército, deben acordar atacar o retirarse, están acampados en las colinas alrededor de un valle donde está el enemigo al cual lo pueden vencer siempre y cuando las tres divisiones ataquen al mismo tiempo. Si solo atacan dos, pierden. El general puede dar las siguientes órdenes: "atacar" o "retirarse". Con esta orden los otros generales ejecutan el plan de acción acordado. Sin embargo, uno de estos, es un traidor que cambiará el mensaje de tal forma que el efecto en el ejército propio sea el más devastador. Para complicar un poco más, estas tres divisiones solo se pueden comunicar a través del intercambio de mensajes usando mensajeros que se pueden demorar más de lo previsto o inclusive pueden ser tomados por el ejército enemigo que está en el valle y cambiar el mensaje que intenta comunicar.

Figura 65.

Consenso con tres generales bizantinos



Nota. Tomado de Coulouris et al. (2012).

En la Figura 65 está una analogía para el problema de los generales bizantinos, vamos a asumir que es un sistema síncrono, de los tres procesos p_1, p_2 y p_3 , uno solo es el traidor. El proceso p_1 es el general que da

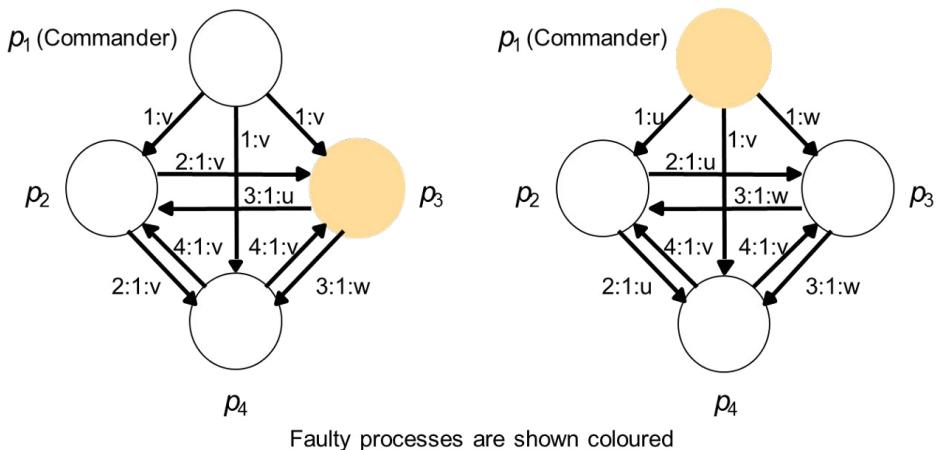
la orden y los procesos p_2 y p_3 son los generales adicionales. Los mensajes tienen la forma “proceso emisor: proceso dueño del mensaje: mensaje”, por ejemplo “3:1:u” se puede traducir como el mensaje es enviado por el p_3 , lleva el mensaje del proceso p_1 y el mensaje es u . (p_3 dice que el proceso p_1 , dijo u).

En la parte izquierda de la Figura 65, el proceso p_3 es el traidor, el proceso p_1 envía un valor v “atacar” a los procesos p_2 y p_3 , luego en la siguiente ronda el proceso traidor p_3 envía un mensaje u al proceso 2 “retirarse”. En este caso el proceso p_2 tiene dos valores y como no puede determinar cuál proceso es el traidor es imposible que llegue a una decisión.

En la parte derecha de la Figura 65, el general traidor, el proceso p_1 , envía el valor de w al proceso p_3 , y al proceso p_2 envía el valor de x . Luego, en la siguiente ronda el proceso p_2 envía el mensaje x al proceso p_3 . Y el proceso p_3 recibe un mensaje w del proceso p_2 . En este caso los dos procesos tienen dos valores diferentes y no pueden llegar a un consenso.

Cuando existen tres generales es imposible llegar a un consenso (Fischer et al., 1985).

Figura 66.
Cuatro generales bizantinos



Nota. Tomado de Coulouris et al. (2012).

La Figura 66 muestra una solución al problema anterior agregando un general o un proceso adicional, ahora en total hay cuatro procesos p_1 , p_2 , p_3 y p_4 . En la parte izquierda el proceso p_3 es el traidor. En la primera ronda

el general p_1 emite la orden "atacar" a través del mensaje v a todos los generales restantes, los procesos p_2 , p_3 y p_4 . En la segunda ronda el proceso traidor p_3 cambia el mensaje y envía el mensaje u y w al proceso p_2 y al proceso p_4 , respectivamente. El proceso p_2 a su vez envía el mensaje V a los procesos p_3 y p_4 . El proceso p_4 envía el mensaje v a los procesos p_2 y p_3 . Ahora el conjunto de valores en p_2 es $\{v, u, v\}$ y en el proceso p_4 es $\{v, v, w\}$. Los procesos correctos p_2 y p_4 aplicando una simple función de mayoría llegan a un consenso y deciden de acuerdo con la orden del general v .

En la parte derecha de la Figura 66, el proceso p_1 es el traidor enviando un mensaje diferente u , w , v a los procesos p_2 , p_3 y p_4 , respectivamente. En la segunda ronda los procesos correctos p_2 , p_3 y p_4 intercambian los valores recibidos. En este caso el conjunto de valores para p_2 es $\{u, w, v\}$, en p_3 es $\{w, u, v\}$ y en p_4 es $\{v, u, w\}$. Al no existir un resultado aplicando la función de mayoría, acuerdan en el valor indeterminado. Es decir, existe consenso.

7.5.3. Consenso en sistemas asíncronos

Todas las soluciones al consenso que se han analizado hasta el momento funcionan en sistemas síncronos. Los algoritmos consideran un temporizador cuando se envían los mensajes en cada una de las rondas que se ejecutan, considerando procesos fallidos a aquellos que no han enviado una respuesta dentro del plazo establecido.

Sin embargo, existen pruebas de que no hay garantía de que se llegue a un consenso en sistemas asíncronos (Fischer et al., 1985), donde los procesos pueden responder los mensajes en tiempos no determinados y no se puede distinguir procesos fallidos de procesos lentos. Lo que se puede lograr en sistemas asíncronos, es lograr un consenso con alguna probabilidad de éxito en el consenso.

Finalmente, vamos a mencionar tres técnicas que se pueden utilizar para lograr consenso en un sistema asíncrono:

- Tratar un sistema asíncrono como síncrono usando mecanismos, como por ejemplo, los usados por el protocolo TCP para el transporte confiable de información, el uso de temporizadores con valores ajustables de acuerdo al comportamiento de la red, identificar los mensajes y el origen de los mismos, usar mensajes de retroalimentación para los mensajes que han sido recibidos.

- Enmascarar las fallas, por ejemplo, en una base de datos distribuida se pueden hacer réplicas de los registros o de toda la base de datos. Además, se puede guardar en almacenamiento primario, como un disco duro, cada cierto tiempo el estado y el valor de los procesos, de tal forma que puedan comenzar desde un punto inmediatamente próximo inferior a la falla. Se puede distribuir el proceso entre diferentes servidores de tal forma que si un proceso falla, una copia de este toma su lugar.
- Usar detectores de fallos para lograr consenso. Un detector de fallos es un componente adicional al sistema distribuido que trata de determinar si los procesos han fallado. De esta manera se garantiza que el consenso se logre porque se determina la cantidad de procesos fallidos.



Actividades de aprendizaje recomendadas

Es importante que en esta semana usted realice las siguientes actividades para consolidar su aprendizaje:

- Leer y familiarizarse con el material didáctico de la asignatura.
 - Lea nuevamente los contenidos de esta guía y cree un mapa conceptual.
 - Consulte el plan docente donde se muestra la secuencia de estudio de los contenidos, así como las actividades que complementan el aprendizaje organizadas por fechas, con su respectiva valoración, rúbricas de evaluación y distribución de las calificaciones por bimestre.
 - El entorno virtual de aprendizaje es el medio de interacción permanente entre docente y estudiante, donde encontrará orientaciones semanales, recursos, herramientas de interacción para con sus compañeros y docente.
- Genere un diagrama de flujo para el algoritmo de servidor central, anillo y multicast para la gestión de zona crítica.

Nota: conteste las actividades en un cuaderno de apuntes o en un documento Word.

A continuación, se invita a que revise cómo van sus conocimientos haciendo la respectiva autoevaluación.



Autoevaluación 7

1. ¿Cuáles de las siguientes alternativas, tienen relación con actividades de coordinación y consenso?

- a. Roles y responsabilidades.
- b. Modelo de fallos.
- c. Servicios web.
- d. Applets.

2. Seleccione las alternativas que considere correctas.

¿Por qué no es apropiado una solución centralizada de coordinación en un sistema distribuido?

- a. Porque puede ser un cuello de botella y degradar el rendimiento del sistema.
- b. Se convierte en un único punto de fallo.
- c. Porque no se puede acceder al procesamiento general compartido.
- d. Porque los recursos de memoria en cada computador no se usan al máximo disminuyendo el rendimiento.

3. La coordinación y consenso en un sistema distribuido asíncrono es una tarea compleja.

- a. Verdadero
- b. Falso

4. ¿Con qué tipos de fallas es más difícil llegar a un consenso?

- a. Bizantinas.
- b. Fallo parada.
- c. Omisión en la red.
- d. Omisión en el emisor.

- 5. ¿Cuándo se implementa un detector de fallas, un proceso es sospecho sí?**
- a. No hay respuesta del proceso, porque el proceso ha fallado.
 - b. No hay respuesta del proceso, debido a que la respuesta ha demorado más en llegar.
 - c. La respuesta del proceso llegó alterada.
 - d. La respuesta del proceso llegó desde otro proceso.
- 6. La propiedad de vitalidad en un algoritmo de coordinación garantiza:**
- a. Algo que eventualmente pasará.
 - b. Los procesos no fallarán nunca.
 - c. El detector de fallos sigue funcionando indiferentemente de las fallas del sistema.
 - d. Qué algo malo o deseado no pasará nunca.
- 7. ¿En un proceso de elección de líder qué valor tiene la variable de líder cuando aún no se ha elegido?**
- a. La identificación del propio proceso.
 - b. La identificación del proceso vecino con el identificador más alto.
 - c. Un valor indeterminado.
 - d. La identificación del proceso vecino con el identificador más bajo.
- 8. En el algoritmo de elección de líder basado en anillo ¿cuándo un proceso con estado no participante recibe un mensaje elegido(k_l)?**
- a. Almacena y registra el identificador del líder.
 - b. Cambia su estado a no participante.
 - c. Cambia su estado a participante.
 - d. Cambia su estado a líder.
- 9. Además de las propiedades de seguridad y vitalidad ¿qué otra propiedad debe garantizar un algoritmo de exclusión mutua?**
- a. Ordenamiento.
 - b. Reflexión.
 - c. Pertenencia.
 - d. Inclusión.

10. ¿En los algoritmos de acceso a la zona crítica cuál usa menos ancho de banda?
- a. Servidor central.
 - b. Algoritmo de bully.
 - c. Basado en anillo.
 - d. Exclusión multicast.

[Ir al solucionario](#)



Actividades finales del bimestre



Semana 16

Esta última semana del primer bimestre es importante que realice las siguientes actividades para consolidar su aprendizaje:

- Ingrese al entorno virtual de aprendizaje para conocer las orientaciones de estudio y actividades a desarrollar esta semana; así como para realizar consultas a su docente tutor e interactuar con sus compañeros.
- Revise los contenidos abordados en el segundo bimestre de la asignatura y sus anotaciones, resúmenes, síntesis respecto a los mismos, como preparación a la evaluación del segundo bimestre.
- Participe de la actividad suplementaria.
- Revise las autoevaluaciones y cuestionarios de las unidades 5, 6 y 7 para recordar y reforzar los contenidos que allí se abordaron.



4. Solucionario

Autoevaluación 1		
Pregunta	Respuesta	Retroalimentación
1	c	Un sistema distribuido se sirve de una red de computadores y de la capacidad de los equipos dentro de ella.
2	c	Los objetos son uno de los componentes especialmente en sistemas distribuidos orientados a objetos.
3	b	Puede ser que los componentes no estén conectados al mismo tiempo al sistema distribuido
4	d	En un sistema distribuido con infinidad de transacciones, estas deben ser sincronizadas con una forma de estandarización de representación, compensando también los tiempos de procesamiento y transferencia de información y acoplando con los relojes locales en cada dispositivo.
5	a	PayPal es un sistema distribuido que es usado para transacciones comerciales.
6	b	Cuando se transmite un video y una parte se pierde, el emisor puede obviar esa información y continuar con el resto.
7	a y b	Akamai utiliza estrategias de DNS y geolocalización de servidores replicados para mejorar la calidad de servicio.
8	a	Las interfaces de entrada definirán los tipos de datos y su representación requerida.
9	a	Permite al sistema y las aplicaciones expandirse sin cambiar la estructura del sistema o los algoritmos de aplicación.
10	b	Los sistemas distribuidos deben tener la capacidad de cumplir con los requisitos principales de seguridad: confidencialidad, disponibilidad, autentificación, integridad y alcance.

Ir a la
autoevaluación

Autoevaluación 2

Pregunta	Respuesta	Retroalimentación
1	a y d	Cuando se inició Internet el tema de seguridad no era un requisito y el tráfico multimedia digital aún no existía.
2	a	El correo electrónico, DNS y WWW son ejemplos de sistemas distribuidos cada uno con diferentes objetivos de servicio.
3	a	Los primeros computadores por su tamaño eran estáticos y trabajaban de manera autónoma, por lo tanto, no existía la probabilidad de que sean vulnerados o hackeados.
4	a	La Universidad Técnica Particular de Loja en Ecuador, tuvo uno de los primeros computadores en el año de 1979 donde inaugura su primer centro de cómputo .
5	b	DNS, ya es un protocolo distribuido que permite usar nombres nemotécnicos en vez de dirección Ip, logrando abstracción de localización ya que no es necesario conocer la dirección del recurso solo tengo que recordar el nombre asociado.
6	b	Cada organización gestiona por ejemplo la cantidad de ancho de banda que puede tener hacia al Internet a través de un ISP.
7	a	Los estándares para Internet están definidos y son de acceso público a través de los RFC (Request for comments) y son creados a través de la IETF (Internet Engineering Task Force).
8	b	Internet es no orientado a la conexión.
9	a	Los enrutadores son el componente principal del núcleo de la red.
10	c y d	En la frontera de la red están los equipos finales, donde corren las aplicaciones en la que se implementa la fiabilidad de la red.

Ir a la
autoevaluación

Autoevaluación 3		
Pregunta	Respuesta	Retroalimentación
1	a	Los protocolos de comunicaciones se refieren a un conjunto de reglas y formatos que son usados en la comunicación entre procesos.
2	a	TCP es un protocolo de la capa de transporte de TCP/IP.
3	b	Desarrollar aplicaciones para la capa física, enlace de datos y de red no es muy común, debido a que estas funcionalidades están definidas en el hardware de comunicación y en el sistema operativo.
4	d	Se hace con un identificador denominado número de puerto.
5	a	Directamente a través de la interfaz de la capa de transporte definida en los protocolos TCP o UDP.
6	b	Pueden implementar sus propios algoritmos para la capa de transporte lo que se denomina middleware.
7	d	Cuando se usa comunicación indirecta, la responsabilidad de la comunicación entre el equipo local y remoto está asignada a un equipo o proceso en el medio.
8	a	Los middlewares se implementan de la capa de transporte hacia arriba.
9	a y d	En TCP/IP un número de puerto para el socket tiene un tamaño de 16 bits. Esto significa que un mismo equipo puede tener abierto 65536 procesos comunicándose a través de la red.
10	a	En TCP/IP existe una distinción histórica de los números de puertos. Los números de puerto bien conocidos, van de 1 a 1023.

[Ir a la autoevaluación](#)

Autoevaluación 4

Pregunta	Respuesta	Retroalimentación
1	a	Un sistema operativo puede almacenar un dato en forma diferente que otro.
2	a	Un sistema operativo puede almacenar un dato en forma diferente que otro. En los datos numéricos pueden almacenar con el bit más representativo a la izquierda o derecha.
3	c	Es la representación en binario de 12 y el bit más representativo a la derecha.
4	c	Por la heterogeneidad de los sistemas operativos y el hardware.
5	c	Con toda la información el receptor puede determinar cómo almacenar el dato correctamente.
6	a	Relacionada con la representación de datos estructurados (clases) y tipos primitivos (entero, carácter, etc.) que pueden ser pasados como argumentos y resultados en una invocación remota.
7	c	Para almacenar y recuperar los objetos de la memoria secundaria en una forma serializada, se debe representar el estado de los objetos de tal forma que puedan ser reconstruidos.
8	a	La comunicación unicast es usada por TCP y UDP por lo general en un esquema cliente – servidor con una relación de uno a uno.
9	d	Multicast permite que un emisor envíe la misma información a varios destinos que pertenecen a un mismo grupo, en una relación uno a muchos.
10	a	Por lo general los fabricantes no habilitan esta función por defecto.

Ir a la
autoevaluación

Autoevaluación 5

Pregunta	Respuesta	Retroalimentación
1	b	Un sistema centralizado tiene homogeneidad en los componentes.
2	a	La gestión está en diferentes puntos de la red y los fallos se pueden dar en cualquier componente.
3	a	Es más fácil porque hay un único punto de falla.
4	b	Los componentes se deben sincronizar a través del intercambio de mensajes.
5	a	Una relación causal garantiza el orden de los eventos de acuerdo con su aparecimiento en un sistema distribuido.
6	b	Los componentes separados se comunican con procesos que envían y reciben para realizar dos tareas fundamentales, transferencia de información y sincronización.
7	a	Para tener un estado y un tiempo global deben intercambiar mensajes y sincronizar su estado y tiempo, considerando los tiempos de latencia.
8	b y c	b y c son correctas.
9	a	Se puede usar un dispositivo que pertenece al grupo del sistema distribuido como referencia e igualar el resto de los dispositivos de acuerdo a su reloj.
10	a	Para coordinar el tiempo en todo el mundo se utiliza el estándar de tiempo UTC.

Ir a la
autoevaluación

Autoevaluación 6

Pregunta	Respuesta	Retroalimentación
1	b	Modelo de arquitectura
2	a	Modelo físico
3	b	La distribución vertical tiene relación con el modelo de arquitectura.
4	b	Las fallas aleatorias son muy complejas de gestionar.
5	b y c	b y c son correctas
6	a y b	a y b son correctas
7	c	Tiene relación con los patrones de la arquitectura.
8	b	Los sockets TCP son usados por sistemas distribuidos para la comunicación entre procesos.
9	b, c y d	b, c y d son correctos
10	a	Usan protocolos como XML y HTTP que son mundialmente aceptados.

Ir a la
autoevaluación

Autoevaluación 7		
Pregunta	Respuesta	Retroalimentación
1	a y b	a y b, por la elección de líder y el consenso aun existiendo fallas.
2	a y b	a y b son correctas, se puede convertir en un cuello de botella al existir solo un ente, respondiendo a todas las peticiones y ser un único punto de fallo.
3	a	Inclusive en algunos casos es imposible lograr este objetivo
4	a	Puede ser un desafío de coordinación en sistemas distribuidos si la falla es bizantina.
5	b	Correcto
6	a	Existirá un resultado
7	c	Aún no existe un líder por lo que el valor es indeterminado.
8	a	Correcto
9	a	El ingreso a la zona crítica puede ser a través de una política FIFO.
10	a	Solo se requieren tres mensajes.

Ir a la
autoevaluación



5. Glosario

DNS – Servicio de nombres de dominio.

FTP – Protocolo de transferencia de archivos.

IETF – Internet Engineering Task Force.

ISP – Internet Service Provider.

P2P – Peer to peer.

PPP – Point to point protocol.

PDU – Unidad de datos del protocolo.

RTT – Round time trip, tiempo de ida y vuelta.

TCP – Transmission Control Protocol.

UTC – Tiempo universal coordinado.

URI – Identificador de recursos uniforme.

WWW – World Wide Web.



6. Referencias bibliográficas

- Atlantic International University. (2021). *Curso de Sistemas Distribuidos*.
Atlantic International University. [https://cursos.aiu.edu/Sistemas Distribuidos de Redes.html](https://cursos.aiu.edu/SistemasDistribuidos de Redes.html)
- Beaulah Soundarabai, P., Sahai, R., Venugopal, K. R., & Patnaik, L. M. (2014). *Improved Bully Election Algorithm for Distributed Systems*.
- Coulouris, G., Dollimore, J., & Kindberg, T. (2012). Distributed Systems: Concepts and Design, Fifth Edititon. In *Computer*.
- Cristian, F., & Fetzer, C. (1999). The Timed Asynchronous Distributed System Model. In *IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS*. <https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.41.4919&rep=rep1&type=pdf>.
- Developer. (2021). *How to Multicast Using Java Sockets* - Developer.com.
<https://www.developer.com/java/data/how-to-multicast-using-java-sockets.html>
- Dolev, D., & Strong, H. R. (1983). Authenticated Algorithms for Byzantine Agreement. *SIAM Journal on Computing*. <https://doi.org/10.1137/0212045>
- Enterprise Project Performance Platform Architecture | EcoSys. (2021).
<https://www.ecosys.net/technology/architecture/>
- Essen, L., & Parry, J. V. L. (1955). An atomic standard of frequency and time interval: A caesium resonator. *Nature*, 176(4476), 280–282. <https://doi.org/10.1038/176280a0>
- Fidge, C. (1991). Logical Time in Distributed Computing Systems. *Computer*.
<https://doi.org/10.1109/2.84874>
- Fischer, M. J., Lynch, N. A., & Paterson, M. S. (1985). Impossibility of Distributed Consensus with One Faulty Process. *Journal of the ACM (JACM)*. <https://doi.org/10.1145/3149.214121>

- Foundation, W. (2016). *Wireshark* Â· Go deep. Wireshark Foundation. <https://www.wireshark.org/>
- Garcia-Molina, H. (1982). Elections in a Distributed Computing System. *IEEE Transactions on Computers*, C-31(1), 48–59. <https://doi.org/10.1109/TC.1982.1675885>
- Google. (2019). *Protocol Buffers* | Google Developers. <https://developers.google.com/protocol-buffers/>
- Goralski, W. (2009). Transmission Control Protocol. In *The Illustrated Network* (pp. 279–300). <https://doi.org/10.1016/b978-0-12-374541-5.50017-1>
- Govoni, N. A. (2012). Internet Service Provider (ISP). In *Dictionary of Marketing Communications*. <https://doi.org/10.4135/9781452229669.n1732>
- Gusella, R., & Zatti, S. (1987). *The Accuracy of the Clock Synchronization Achieved by TEMPO in Berkeley UNIX 4.3BSD*.
- Introduction to TCP/IP (Part 4) - Sockets and Ports - Developer Help.* (n.d.). Retrieved March 5, 2021, from <https://microchipdeveloper.com/com4104:start>
- Ipv6summit. (2021). *Nació desde la industria militar y se expandió al mundo*. <http://ipv6summit.mx/evolucion-de-internet/>
- JSON. (2021). <https://www.json.org/json-en.html>
- Lamport, leslie. (2021). *File:Lamport-Clock-en.svg* - Wikimedia Commons. <https://commons.wikimedia.org/wiki/File:Lamport-Clock-en.svg>
- Lamport, L. (1978). Time, Clocks, and the Ordering of Events in a Distributed System. *Communications of the ACM*. <https://doi.org/10.1145/359545.359563>
- Lamport, L., Shostak, R., & Pease, M. (1982). The Byzantine Generals Problem. *ACM Transactions on Programming Languages and Systems (TOPLAS)*. <https://doi.org/10.1145/357172.357176>

López Fuentes, F. de A. (2015). Sistemas Distribuidos. In *Universidad Autónoma Metropolitana* (Issue 13). <http://ocw.uc3m.es/ingenieria-informatica/sistemas-distribuidos-2013>

Lua, E. K., Crowcroft, J., Pias, M., Sharma, R., & Lim, S. (2005). A survey and comparison of peer-to-peer overlay network schemes. In *IEEE Communications Surveys and Tutorials* (Vol. 7, Issue 2, pp. 72–93). <https://doi.org/10.1109/COMST.2005.1610546>

Maekawa, M. (1985). A [formula omitted] Algorithm for Mutual Exclusion in Decentralized Systems. *ACM Transactions on Computer Systems (TOCS)*. <https://doi.org/10.1145/214438.214445>

Manu. (2021). *Home / SETI Institute*. <https://seti.org/>

Mattern, F. (1989). Virtual Time and Global States of Distributed Systems. *Event London.*

Nygren, E., Sitaraman, R. K., & Sun, J. (2010). The Akamai network: A platform for high-performance Internet applications. *Operating Systems Review (ACM)*. <https://doi.org/10.1145/1842733.1842736>

OMG Object Management Group. (2021). *Welcome To CORBA Web Site!* <https://corba.org/index.htm>

Oracle. (2004). *Java Object Serialization Specification v 1.5.0*. <https://docs.oracle.com/javase/7/docs/platform/serialization/spec/serialTOC.html>

Postel, J. (1980). User Datagram Protocol. In *Rfc 768* (Issue 768, pp. 1–4). IETF. <http://www.ietf.org/rfc/rfc768.txt>

Priano, F., Sandoval, J., & Quijada, J. (2007). *RedIRIS - Incorporación de Nuevas Tecnologías INTRANET*.

RFC 5905 - Network Time Protocol Version 4: Protocol and Algorithms Specification. (2021). <https://tools.ietf.org/html/rfc5905>

Ricart, G., & Agrawala, A. K. (1981). An optimal algorithm for mutual exclusion in computer networks. *Communications of the ACM*, 24(1), 9–17. <https://doi.org/10.1145/358527.358537>

- Robert, M. (2006). *6.824 Distributed Computer Systems* Engineering MIT OpenCourseWare. MIT OpenCourseWare. <https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-824-distributed-computer-systems-engineering-spring-2006/#>
- Spicer, D. (2015, March). *The World's Smallest Computer - CHM*. <https://computerhistory.org/blog/the-worlds-smallest-computer/>
- The Web Expansion – die unaufhaltsame Ausbreitung des Internets - TRENDONE Blog.* (2021). <https://blog.trendone.com/2017/11/06/the-web-expansion-die-unaufrichtige-ausbreitung-des-internets/>
- UTPL. (1979a). *Historia / UTPL*. <https://utpl.edu.ec/es/historia>
- UTPL. (1979b). *Parte de la antigua Unidad de Proyectos y Sistemas Informáticos / Flickr*. <https://www.flickr.com/photos/utpl/5392490035/in/photostream/>
- W3C. (2008). *Extensible Markup Language (XML)* (Issue 1.0).
- Z4 (computer) - Wikipedia. (2021). [https://en.wikipedia.org/wiki/Z4_\(computer\)](https://en.wikipedia.org/wiki/Z4_(computer))



7. Anexos

Ejemplo de tolerancia de aplicaciones en sistemas distribuidos.

Actividad	Ejemplo	Tolerancia a fallos	Tolerancia a pérdida	Tolerancia al retardo
Transporte	Google Maps	9	9	1

Educación

Salud

Comercio

Redes sociales Facebook

Actividades en línea

Entretenimiento Spotify

Actividad	Ejemplo	Tolerancia a fallos	Tolerancia a pérdida	Tolerancia al retardo
Comunicación personal	Skype			