Sistemas Distribuidos

Francisco de Asís López Fuentes

Obra ganadora Concurso 2014 para publicación de libros de texto y materiales de apoyo a la Docencia

10 años
2005 – 2015
Casa abierta al tiempo
UNIVERSIDAD AUTÓNOMA METROPOLITANA
Unidad Cuajimalpa

Francisco de Asís López Fuentes

Sistemas distribuidos

Esta investigación fue dictaminada por pares académicos

Clasificación Dewey: 004.36 L67
Clasificación LC: QA76.9.D5 L67
López Fuentes, Francisco de Asís
Sistemas distribuidos / Francisco de Asís López Fuentes. -- México : UAM, Unidad Cuajimalpa, c2015.
200 p. : il., diagramas, col. ; 24 cm. (Una década de la Unidad Cuajimalpa de la Universidad Autónoma Metropolitana)
ISBN de la Colección Una Década: 978-607-28-0452-4
ISBN de este libro: 978-607-28-0476-0
1. Procesamiento electrónico de datos – Procesos distribuidos – Libros de texto 2. Programación de computadoras – Libros de texto 3. Red de computadoras – Libros de texto 4. Universidad Autónoma Metropolitana – Unidad Cuajimalpa – Planes de estudio

Universidad Autónoma Metropolitana Unidad Cuajimalpa
.Avenida Vasco de Quiroga 4871,
Col. Santa Fe Cuajimalpa. Delegación Cuajimalpa de Morelos,
C.P. 05348, México D.F. (Tel.: 5814 6500)
www.cua.uam.mx

ISBN DE LA COLECCIÓN UNA DÉCADA: 978-607-28-0452-4
ISBN DE ESTE LIBRO: 978-607-28-0476-0

Diseño de portada: Ricardo López Gómez.
Formación y edición: Juan Carlos Rosas Ramírez.

CONTENIDO

Prefacio 9

Objetivos 13

Capítulo 1. Introducción a los sistemas distribuidos 15
1.1 Introducción 15
1.2 Ventajas y desventajas de los sistemas distribuidos 16
1.2.1 Ventajas de los sistemas distribuidos con respecto a los sistemas centralizados 16
1.2.2 Ventajas de los sistemas distribuidos con respecto a las computadoras aisladas 16
1.2.3 Desventajas de los sistemas distribuidos 17
1.3 Formas distintas de organizar *n* computadoras 17
1.4 Aspectos del diseño de sistemas distribuidos 17
1.5 Taxonomía de los sistemas distribuidos 19
Ejercicios 20
Actividad integradora 21

Capítulo 2. Redes de computadoras 23
2.1 Introducción 23
2.2 Principales componentes de una red de cómputo 24
2.3 Modo de operación y conmutación 26
2.4 Tipos de redes 27
2.5 Topología de redes 28
2.6 Modelo OSI 29
2.7 Norma IEEE-802 30
2.8 Protocolos y paquetes 32

2.8.1 Protocolos 32
2.8.2 Paquetes 33
2.8.3 Redes de banda ancha 33
2.8.4 Redes inalámbricas 35
Ejercicios 35
Actividad integradora 36

Capítulo 3: Modelos de arquitecturas 37
3.1 Introducción 37
3.2 Modelo cliente - servidor 37
3.3 Proxy 39
3.4 Peer-to-Peer 40
3.5 Applets 42
3.6 Clúster 44
3.7 Grid 45
3.8 Arquitectura de capas 46
3.9 Middleware 48
3.10 CORBA 49
Ejercicios 52
Actividad integradora 53

Capítulo 4. Procesos y comunicación 55
4.1 Introducción 55
4.2 Hilos 55
4.3 Cliente 57
4.4 Servidores 57
4.5 Comunicación entre procesos 58
4.5.1 Modelo cliente - servidor (C-S) 58
4.5.2 Llamada de procedimiento remoto (RPC) 60
4.5.3 Comunicación en grupo 62
4.6 Interfaz de programación de aplicaciones (API) 63
4.6.1 La interfaz de socket 64
4.6.2 Funciones de la API de sockets 64
4.6.3 Ejemplo de cliente servidor usando socket 66
Ejercicios 68
Actividad integradora 69

Capítulo 5. Sincronización 71
5.1 Introducción 71

5.2 Sincronización de relojes 71
5.3 Algoritmos para la sincronización de relojes 72
5.3.1 El Algoritmo de Lamport 72
5.3.2 El algoritmo de Christian 73
5.3.3 El algoritmo de Berkeley 74
5.4 Exclusión mutua 74
5.4.1 Algoritmo de servidor centralizado 75
5.4.2 Algoritmo de Ricart y Agrawala 76
5.4.3 Algoritmo de anillo de token 77
5.5 Algoritmos de elección 78
5.5.1 El algoritmo del Grandulón 78
5.5.2 Algoritmo de anillo 80
5.6 Algoritmos de consenso 82
5.6.1 Problema de los generales bizantinos 82
Ejercicios 85
Actividad integradora 85

Capítulo 6. Transacciones distribuidas 87
6.1 Introducción 87
6.2 Modelo de transacciones 88
6.3 Problemas debido a la concurrencia de transacciones 88
6.4 Recuperación de transacciones 89
6.5 Transacciones anidadas y distribuidas 89
6.5.1 Transacciones anidadas 89
6.5.2 Transacciones distribuidas 90
6.6 Implantación de sistemas distribuidos 90
6.7 Transacciones con replicación 91
6.8 Problema del "deadlock" en los sistemas distribuidos 92
6.9 Servicios web 92
Ejercicios 96
Actividad integradora 97

Capítulo 7. Sistemas operativos distribuidos 99
7.1 Introducción 99
7.2 Núcleo y servidores 99
7.3 Nombramiento y protección de recursos 100
7.3.1 Nombrado de recursos 100
7.3.2 Protección de recursos 101
7.4 Casos de sistemas operativos distribuidos 101

7.4.1 Mach 102
7.4.2 Chorus 106
7.4.3 DCE 110
Ejercicios 116
Actividad integradora 116

Capítulo 8. Sistemas de archivos distribuidos 119
8.1. Introducción 119
8.2 Servicios de archivos 119
8.3. Servicio de archivos planos 120
8.4 Servicio de directorio 120
8.5 Módulo cliente 121
8.6 Sistemas NFS y AFS 121
8.6.1 Sistema NFS (Network File Systems) 121
8.6.2 Sistema AFS (Andrew File Systems) 123
8.7 Memoria compartida distribuida 125
8.7.1 Generalidades 125
8.7.2 Consistencia en la DSM 126
8.7.3 DSM basada en paginación 127
Ejercicios 128
Actividad integradora 129

Capítulo 9. Replicación, consistencia y tolerancia a fallas 131
9.1. Introducción 131
9.2 Replicación 132
9.2.1 Beneficios de usar replicación en los sistemas distribuidos 132
9.2.2 Requisitos para realizar la replicación 133
9.2.3 Modelo general de gestión de réplica 134
9.2.4 Servicios de tolerancia a fallas basados en replicación 135
9.3 Consistencia 137
9.3.1 Tipos de inconsistencias 138
9.3.2 Modelos de consistencia 139
9.4. Tolerancia a fallas 141
9.4.1 Origen de una falla 141
9.4.2 Clasificación de fallas 142
9.4.3 Fallas en los procesos distribuidos 143
9.4.4 Redundancia 144
Ejercicios 145
Actividad integradora 145

Capítulo 10. Seguridad 147
10.1 Introducción 147
10.2 Ataques a la seguridad 148
10.3 Servicios de seguridad 150
10.4 Mecanismos de seguridad 151
10.5 Sistemas criptográficos 152
10.5.1 Requisitos para la criptografía de llave pública 154
10.6 Autenticación 155
10.6.1 Kerberos 155
Ejercicios 159
Actividad integradora 160

Capítulo 11. Multimedia distribuida 161
11.1. Introducción 161
11.2 Estándares de codificación de video 162
11.2.1 Estándar H.264/AVC 163
11.2.2 Codificación de video escalable (SVC) 164
11.3 Infraestructuras para flujos de video 165
11.3.1 Multicast IP 166
11.3.2 Red de distribución de contenidos 167
11.3.3 Multicast de capa de aplicación 168
11.4 Sistemas de flujos de video basados en redes P2P 169
11.4.1 Topología de flujo de video P2P basado en árbol 170
11.4.2 Topología de flujo de video P2P basado en bosque 171
11.4.3 Topología de flujo de video P2P basado en malla 171
Ejercicios 174
Actividad integradora 174

Capítulo 12. Cómputo en la nube 177
12.1 Introducción 177
12.2 Abstracción y virtualización 178
12.3 Modelos de cómputo en la nube 178
12.4 Tipos de servicios del cómputo en la nube 179
12.5 Tipos de cómputo en la nube 180
12.6 Características del cómputo en la nube 181
12.7 Ventajas del cómputo en la nube 182
12.8 Retos del cómputo en la nube 183
12.9 Cómputo en la nube soportado
por redes Peer-to-Peer (P2P) 183

Ejercicios 186
Actividad integradora 187

Glosario 189

Referencias 193

Prefacio

El objetivo de estas notas es apoyar la UEA (Unidad de Enseñanza-Aprendizaje) de Sistemas distribuidos de la Licenciatura en Tecnologías y Sistemas de Información de la Universidad Autónoma Metropolitana Unidad Cuajimalpa. Generalmente, esta UEA se imparte a alumnos que previamente aprendieron y pusieron en práctica diferentes conocimientos relacionados con la programación, los sistemas operativos y las redes de comunicación. Además, algunos alumnos también poseen conocimientos relacionados con la tecnología web y las bases de datos. Con estos conocimientos, los estudiantes pueden desarrollar habilidades para implementar diferentes sistemas de información pero también han detectado la necesidad de implementar sistemas de información que ofrezcan y usen servicios distribuidos en diferentes sitios.

Actualmente las tecnologías de la información y comunicación (TIC) son esenciales para mejorar la gestión tradicional en las organizaciones. En este contexto, el cómputo distribuido juega un rol importante en el diseño de los sistemas de información y comunicación. La gestión de actividades distribuidas en las organizaciones permite diseñar sistemas de información globales correctamente alineadas con las necesidades del negocio, con el objetivo de maximizar el potencial y la continuidad del servicio que prestan los sistemas de información. Los sistemas distribuidos es un área muy amplia, pues abarca desde conceptos fundamentales de redes de comunicación hasta temas tan emergentes como el cómputo en la nube o multimedia distribuida en red, pasando por los temas clásicos de sincronización, comunicación de procesos distribuidos y exclusión mutua.

Es importante que el futuro egresado en Tecnologías y Sistemas de la Información conozca la importancia de los sistemas distribuidos, así como los algoritmos y técnicas para implementar de manera apropiada estos sis-

temas. El alumno debe de saber que la implementación de un sistema distribuido puede resultar más costosa y de mayor complejidad técnica que la de un sistema centralizado, tanto en cuestiones de hardware como de software, pero también debe de conocer su administración y control. Por ejemplo, un sistema distribuido está más propenso a sufrir ataques de seguridad que uno centralizado, debido principalmente a que existen diversos puntos de acceso al sistema. En contraste, un sistema distribuido es más robusto y presenta una mayor tolerancia a fallas que un sistema centralizado.

Los sistemas distribuidos es un tema muy extenso. En este trabajo centramos nuestra atención en los algoritmos distribuidos relacionados a la comunicación y sincronización de procesos, así como a las arquitecturas distribuidas. También se da un repaso al tema de redes de computadoras, sistemas operativos distribuidos y aspectos relacionados a la seguridad informática por su importante relación con los sistemas distribuidos. Finalmente, se abordan temas actuales como multimedia en red y cómputo en la nube. Existen algunos temas como objetos distribuidos y servicios web que no se abordan de una manera extensa. El presente trabajo está organizado de la siguiente manera. En el capítulo 1 se ofrece un panorama general de los sistemas distribuidos, su importancia y evolución a través de los años. Una introducción a los fundamentos de redes se muestra en el capítulo 2. Las principales arquitecturas de los sistemas distribuidos son presentadas y discutidas en el capítulo 3. El capítulo 4 describe los procesos distribuidos y las técnicas de comunicación entre procesos. El capítulo 5 está dedicado a la sincronización de procesos y tareas en los sistemas distribuidos, así como a los algoritmos y protocolos relacionados. Los temas acerca de la replicación, consistencia y tolerancia a fallas son expuestos en el capítulo 6. El capítulo 7 presenta una introducción a los sistemas de archivos distribuidos. En el capítulo 8 se estudian aspectos relacionados a las transacciones en los sistemas distribuidos. Las principales características de los sistemas operativos distribuidos son revisados en el capítulo 9. Los conceptos básicos de seguridad y su importancia en los sistemas distribuidos son discutidos en el capítulo 10. Algunas tendencias actuales de los sistemas distribuidos, como multimedia en red y cómputo en la nube, son revisados en los capítulos 11 y 12, respectivamente. Más de 80 figuras ilustran la explicación de diferentes algoritmos y arquitecturas distribuidas.

Planeación recomendada del presente texto para la Licenciatura en Tecnología y Sistemas de Información.

Tema	Semanas											
	1	2	3	4	5	6	7	8	9	10	11	12
Capítulo 1: Introducción a los sistemas distribuidos												
Capítulo 2: Redes de computadoras												
Capítulo 3: Modelos de arquitecturas												
Capítulo 4: Procesos y comunicación												
Capítulo 5: Sincronización												
Capítulo 6: Transacciones distribuidas												
Capítulo 7: Sistemas operativos distribuidos												
Capítulo 8: Sistemas de archivos distribuidos												
Capítulo 9: Replicación, consistencia y tolerancia a fallas												
Capítulo 10: Seguridad												
Capítulo 11: Multimedia distribuida*												
Capítulo 12: Cómputo en la nube*												

* Se puede trabajar estos capítulos como temas de proyecto final de la UEA.

Nota: La UEA se imparte siete horas a la semana, de las cuales cuatro horas son teóricas y tres prácticas.

Objetivos

Que al finalizar del curso el alumno sea capaz de:

- Aplicar los principales modelos arquitecturales, de programación y de comunicación de la computación distribuida.
- Aplicar los algoritmos distribuidos básicos en la construcción de aplicaciones distribuidas.
- Explicar el funcionamiento de los servicios distribuidos más comunes.

Capítulo 1. Introducción a los sistemas distribuidos

Objetivo: Que el alumno comprenda la importancia de los sistemas distribuidos en la sociedad actual, sus aspectos de diseño y sus principales retos tecnológicos.

1.1 Introducción

En años recientes, el avance en las tecnologías de cómputo y las telecomunicaciones han permitido una gran expansión de los sistemas de información, así como su alta disponibilidad, independientemente de su campo de aplicación. Las telecomunicaciones permiten la conectividad de un gran número de usuarios ubicados en cualquier parte del mundo por medio de la transmisión de voz, datos o video a través de una gran variedad de dispositivos. Diferentes redes de comunicación de área local (LAN), metropolitanas (MAN), así como de área amplia (WAN), pueden ser accedidas a través de Internet. Esto ha permitido que paralelamente surjan instalaciones de cómputo donde pueden ser desplegadas aplicaciones para realizar procesamiento distribuido de tareas. Estas nuevas facilidades ofrecen a los usuarios y organizaciones una gran flexibilidad para estructurar sus propios sistemas de información de una manera eficiente, así como la oportunidad de interactuar con otros sistemas de información de una manera distribuida. Como consecuencia, esto ha generado una gran dependencia de estos sistemas distribuidos para poder transmitir o procesar información.

Tanenbaum [1996] define un sistema distribuido como una colección de computadoras independientes que aparecen ante los usuarios del sistema como una única computadora. El advenimiento de los sistemas distribuidos ha estado soportado en dos importantes innovaciones tecnológicas:

- El microprocesador.
- Las redes de área local.

1.2 VENTAJAS Y DESVENTAJAS DE LOS SISTEMAS DISTRIBUIDOS

1.2.1 Ventajas de los sistemas distribuidos con respecto a los sistemas centralizados

Entre las principales ventajas de los sistemas distribuidos con respecto a las computadoras centralizadas se encuentran:

- *Economía*: Los microprocesadores ofrecen una mejor relación precio/rendimiento que las computadoras centrales.
- *Velocidad*: Un sistema distribuido puede tener mayor poder de cómputo que una computadora centralizada individual.
- *Distribución inherente*: Implica que un sistema distribuido puede emplear aplicaciones instaladas en computadoras remotas.
- *Confiabilidad*: El sistema es consistente, aun si una computadora del sistema deja de funcionar.
- *Crecimiento proporcional*: Cada vez que se requiera mayor poder de cómputo en el sistema, solo se pueden adicionar los incrementos de cómputo requeridos.

1.2.2 Ventajas de los sistemas distribuidos con respecto a las computadoras aisladas

Con respecto a las computadoras aisladas, es decir, aquellas que no se encuentran conectadas a una red, los sistemas distribuidos tienen las siguientes ventajas [Coulouris, Dollimore & Kinderberg, 2001]:

- *Datos compartidos*: Permite que distintos usuarios tengan acceso a una base de datos o archivo común.
- *Dispositivos compartidos*: Permite compartir un recurso costoso entre distintos usuarios, como plotters o impresoras láser.
- *Comunicación*: Brinda la posibilidad de comunicación de usuario a usuario (telnet, correo electrónico, etc.).

- *Confiabilidad*: Facilita la repartición de la carga de trabajo entre las distintas computadoras con base en su funciones y capacidades, brindando una mayor flexibilidad y confiabilidad al sistema.

1.2.3 Desventajas de los sistemas distribuidos

A pesar de los diferentes beneficios que introducen los sistemas distribuidos, todavía existen diferentes retos que deben ser resueltos como los siguientes [Coulouris *et al.*, 2001]:

- *Software*: Gran parte del software para sistemas distribuidos está aún en desarrollo.
- *Redes*: Los problemas de transmisión en las redes de comunicación todavía son frecuentes en la transferencia de grandes volúmenes de datos (por ejemplo, multimedia).
- *Seguridad*: Se necesitan mejores esquemas de protección para mejorar el acceso a información confidencial o secreta.
- *Tolerancia a fallas*: Las fallas operativas y de componentes aún son frecuentes.

1.3 Formas distintas de organizar *N* computadoras

La organización de cierta cantidad de computadoras se puede realizar usando alguno de los casos de los siguientes sistemas operativos:

- Sistema operativo de red.
- Sistema operativo distribuido.
- Sistema operativo de multiprocesamiento.

1.4 Aspectos del diseño de sistemas distribuidos

Transparencia
Es una característica de los sistemas distribuidos para ocultar al usuario la manera en que el sistema funciona o está construido, de tal forma que el usuario tenga la sensación de que todo el sistema está trabajando en una sola máquina local. Entre las principales transparencias deseables en un sistema distribuido están [Coulouris *et al.*, 2001]:

- *De localización*: Los usuarios no pueden saber dónde se encuentran los datos localizados.
- *De migración*: Los recursos se pueden mover a voluntad sin cambiar su nombre.
- *De réplica*: Los usuarios no pueden ver el número de copias existentes.
- *De concurrencia*: Varios usuarios pueden compartir recursos de manera automática.
- *De paralelismo*: La actividad o consulta puede requerir procesamiento paralelo sin que el usuario lo perciba.
- *De fallas*: Cuando una computadora del sistema falla, esta es imperceptible para el usuario.
- *De desempeño*: El funcionamiento y velocidad de las máquinas donde se consulta es imperceptible para el usuario.
- *De escalabilidad*: El usuario ignora cuándo en el sistema se agrega otra computadora.

Flexibilidad
Facilita modificaciones al diseño inicial.

Confiabilidad
Permite que, en caso de que una computadora falle, otra la pueda sustituir en la realización de sus tareas asignadas.

Desempeño
Está en referencia a los tiempos de respuesta de una aplicación.

Escalabilidad
Permite que a la arquitectura actual se le pueda adicionar más poder de cómputo.

Repartición de la carga
Se debe analizar con qué equipos cuenta el sistema y los diferentes recursos de cómputo en cada uno de ellos, como capacidad de disco, velocidad de la red, etc. Los tipos de arquitectura a usar pueden ser:

- Servidores-estación de trabajo.
- Pila de procesadores.
- Multiprocesadores con memoria compartida.
- Multiprocesadores con memoria distribuida.

Mantenimiento de consistencia

Verificar que todos los conceptos involucrados con el sistema operativo, al operar en un esquema distribuido, sigan realizándose de manera correcta. Entre los puntos a observar están los siguientes:

- Modificación.
- Caché.
- Falla.
- Replicación.
- Interfaz de usuario.
- Reloj.

Funcionalidad

Implica que el sistema distribuido a implementar funcione de acuerdo con las metas trazadas y que permita hacer más eficiente el trabajo que antes se hacía usando un sistema centralizado.

Seguridad

Es importante considerar todos los factores de riesgo a que se expone la información en un ambiente distribuido, por ello se deben de implementar los mecanismos de seguridad que permitan proteger esta información.

1.5 Taxonomía de los sistemas distribuidos

Con base en su taxonomía, los sistemas distribuidos pueden clasificarse de la siguiente manera:

1. Sistemas con software débilmente acoplado en hardware débilmente acoplado.
   Ejemplo: Sistema operativo de red, como es el caso de NFS (Network File System - Sistema de archivo de red).
2. Sistemas con software fuertemente acoplado en hardware fuertemente acoplado.
   Ejemplo: Sistemas operativos de multiprocesador (sistemas paralelos).
3. Sistemas con software fuertemente acoplado en hardware débilmente acoplado.
   Ejemplo: Sistemas realmente distribuidos (imagen de sistema único).

Un caso de los sistemas distribuidos con software y hardware débilmente acoplado son los sistemas operativos de red. Algunas prestaciones de estos sistemas son:

- Conexión remota con otras computadoras.
- Copia remota de archivos de una máquina a otra.
- Sistema de archivos global compartidos.

EJERCICIOS

1. Menciona tres ventajas y tres desventajas de los sistemas distribuidos con respecto a los centralizados.

2. Indica la importancia de la transparencia en los sistemas distribuidos.

3. Explica en qué consiste la transparencia de red en los sistemas distribuidos.

4. Indica cuál es la diferencia entre sistemas fuertemente acoplados y sistemas débilmente acoplados.

5. Indica la diferencia entre un sistema operativo de red y un sistema operativo distribuido.

6. Indica la diferencia entre una pila de procesadores y un sistema distribuido.

7. ¿Qué significa "imagen único" sistema en los sistemas distribuidos?

8. Indica cinco tipos de recursos en hardware y software que pueden compartirse de manera útil.

9. ¿Por qué es importante el balanceo de carga en los sistemas distribuidos?

10. ¿Cuándo se dice que un sistema distribuido es escalable?

11. ¿Por qué existe más riesgo a la seguridad en un sistema distribuido que en un sistema centralizado?

Actividad integradora

1. De la aplicación de los contenidos, así como de las habilidades y actitudes desarrolladas.
   Se recomienda exposiciones por parte del profesor, así como discusiones en grupo sobre casos específicos y prácticos de sistemas distribuidos y sus alcances con respecto a niveles de transparencia o integración (fuertemente acoplados o débilmente acoplados).
   El alumno debe de mostrar interés por identificar cada componente de un sistema distribuido y la contribución de cada uno de ellos dentro del sistema. Es importante que el alumno comprenda los diferentes niveles de transparencia que puede integrar un sistema distribuido desde su diseño y el impacto en términos de costo-beneficio para la organización.
   Un par de sesiones podrían ser utilizadas para cubrir este capítulo.
2. Del logro de los objetivos establecidos en el capítulo o apartado del material.
   Se espera alcanzar los objetivos de este capítulo con el apoyo en la solución de los ejercicios que se incluyen al final de este capítulo.

Capítulo 2. Redes de computadoras

Objetivo: Que el alumno revise los componentes básicos de las redes de comunicación para entender su importancia vital como infraestructura para la construcción de los sistemas distribuidos.

2.1 Introducción

Las redes de computadoras son un componente importante de los sistemas distribuidos. Una red de computadoras es una colección interconectada de computadoras autónomas que son capaces de intercambiar información [Tanenbaum, 1997]. El objetivo principal de las redes de computadoras es compartir recursos, de tal manera que todos los programas, equipos y datos se encuentren disponibles para quien lo solicite sin importar su ubicación. El uso de las redes de cómputo se ha incrementado durante los últimos años. La comunicación por computadora se ha convertido en una parte esencial de la infraestructura actual. La conectividad se usa en muchos aspectos y, con el crecimiento continuo de la Internet, las demandas de enlaces de mayor capacidad también han aumentado. En una red de cómputo, los datos son transmitidos entre computadoras usando secuencia de bits para representar códigos. La capacidad de transmisión de los datos, referida comúnmente como ancho de banda, es descrita en bits por segundo (bit/s). Las capacidades típicas [Black, 1993] de transmisión de datos se muestran en la tabla 2.1.

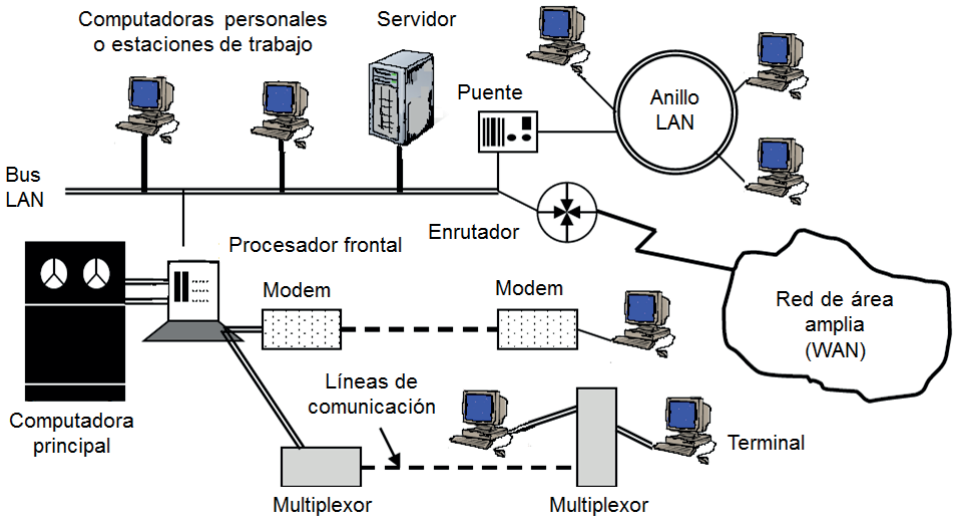
Tabla 2.1. Velocidad de conexión y usos en las redes de computadoras [Black, 1993]

Velocidad típica (bit/s)	Uso típico
0-600	Telégrafo, terminales viejas, telemetría
600-2,400	Terminales operadas humanamente, computadoras personales
2,400-19,200	Aplicaciones que requieren respuesta rápida y/o rendimiento similar como transferencia de archivos
32,000-64,000	Voz digital, aplicaciones de alta velocidad, algo de video
64,000-1,544,000	Muy alta velocidad para múltiples usuarios, tráfico de computadora a computadora, conexión principal de red, video
Mayores a 1.5 MB	Conexión principal de red, video de alta calidad, voz digital múltiple

2.2 PRINCIPALES COMPONENTES DE UNA RED DE CÓMPUTO

Las redes de computadoras están integradas por diversos componentes, algunos de los cuales se muestran en la figura 2.1 [Black, 1993].

Figura 2.1. Componentes de redes de cómputo



Los principales componentes de una red de cómputo son [Black, 1993]:

- *Medios de comunicación:* Cualquier cosa usada para transportar datos en la forma de señales eléctricas (líneas telefónicas, líneas dedicadas o canal LAN).

- *Macrocomputadoras:* Computadoras donde residen las grandes bases de datos o información de una empresa bajo un ambiente centralizado.

- *Terminales de cómputo*: Dispositivos de entrada/salida de una computadora principal, puede consistir de teclado, lector óptico o cámara de vídeo para la entrada y monitor de video o impresora como salida.

- *Enrutadores*: Dispositivo que en la red de comunicación examina las direcciones de la red dentro de un protocolo dado y encamina los paquetes al destino por la ruta más eficiente previamente determinada.

- *Modem*: Dispositivo usado para convertir datos digitales seriales de una terminal transmisor en una adecuada señal analógica para un canal telefónico, así como para reconvertir la señal analógica a digital serial para que sea usada por una terminal receptora.

- *Multiplexores*: Estos dispositivos permiten que más de una terminal comparta la línea de comunicación y pueda resultar en una ventaja sustancial, al reducir el número de líneas usadas.

- *Estaciones de trabajo:* En un entorno de red es una computadora para un único usuario, de alto rendimiento, que sirve para diseño, ingeniería o aplicaciones científicas.

- *Conmutadores:* Dispositivo mecánico o electrónico que sirve para comandar el flujo de señales eléctricas u óptica.

- *Procesador central:* Usado todavía por algunos sistemas de cómputo, su función es manejar el procesamiento de comunicaciones en un entorno de macrocomputadoras, conectando por un lado los canales de comunicación y por el otro la macrocomputadora.

- *Servidores:* Su propósito es proporcionar rendimiento a estaciones de trabajo o computadoras personales o para proporcionar base de datos y servicios de impresión o correo en una red.

2.3 MODO DE OPERACIÓN Y CONMUTACIÓN

El modo de operación de un enlace de comunicación es una característica a considerar desde la perspectiva de un sistema distribuido. Usualmente para transmitir datos por un enlace de comunicación se ocupan tres modos:

1. *Comunicación simplex:* Este modo se presenta cuando los datos viajan en una sola dirección.
2. *Comunicación half-duplex:* Este modo permite que los datos viajen en dos direcciones, una a la vez.
3. *Comunicación full-duplex*: En este modo los datos viajan simultáneamente en ambas direcciones.

Para transferir datos, las redes utilizan comunicación conmutada, lo que permite a los dispositivos compartir líneas físicas de comunicación, los métodos [Tanenbaum, 1997] más usuales de comunicación conmutada son:

- *Conmutación de circuitos:* Crea una ruta única e ininterrumpida entre dos dispositivos que quieren comunicarse así que, mientras estos se comunican, ningún otro puede ocupar esa ruta.

- *Conmutación de mensajes:* En esta conmutación no existe un establecimiento anticipado de la ruta entre el que envía y quien recibe. Cuando el que envía tiene listo un bloque de datos, esta se almacena en la primera central de conmutación, para expedirse después como un salto a la vez. Cada bloque se recibe completo, se revisa y se retransmite, sin límite para el tamaño del bloque.

- *Conmutación de paquetes:* Aquí los datos se dividen en fragmentos llamados paquetes que pueden viajar por múltiples rutas entre distintas computadoras. Como los paquetes pueden viajar en ambas direcciones, requieren una dirección destino. La conmutación de paquetes no reserva ancho de banda y lo adquiere conforme lo necesita, por lo que es muy

útil en el manejo del tráfico interactivo. Aquí los paquetes son guardados en la memoria de las centrales de conmutación.

- *Conmutación híbrida:* Son las variantes que pueden existir en la conmutación de circuitos y paquetes que tratan de aprovechar las ventajas de cada una, como la conmutación de circuitos por conexión rápida y la conmutación por división de tiempo.

2.4 TIPOS DE REDES

Las redes de cómputo, que integran diversos componentes que operan entre sí para compartir recursos, pueden clasificarse de diferentes maneras [Black, 1993]. Los tipos de redes de cómputo más conocidos son:

a) Por su servicio:
- *Redes públicas*, redes de acceso público, por ejemplo, la red telefónica pública comercial.
- *Redes privadas*, utilizadas dentro del ámbito de una empresa para su uso exclusivo.

b) Por su funcionamiento:
- *Redes de conmutación*, se usan en los métodos de conmutación para ofrecer la comunicación entre dispositivos (teléfonos o computadoras personales).
- *Redes de difusión (Broadcast)*, redes en que la comunicación se realiza por difusión desde un dispositivo a más de un dispositivo (señales de satélite, radio y TV).

c) Por su extensión:
- *Redes de área local (LAN),* redes con múltiples usuarios conectados en una parte geográfica pequeña, por lo general no mayor a 1 km; ofrecen canales de comunicación de alta velocidad y por lo general son redes privadas y relativamente libres de error.
- *Redes de área metropolitana (MAN),* redes con múltiples usuarios conectados en una parte geográfica que va de 1 a 10 km, aproximadamente.
- *Redes de área amplia (WAN),* redes con múltiples usuarios conectados en una amplia región geográfica, como entre ciudades, países o continentes, por lo general la comunicación abarca líneas telefónicas, satélites y

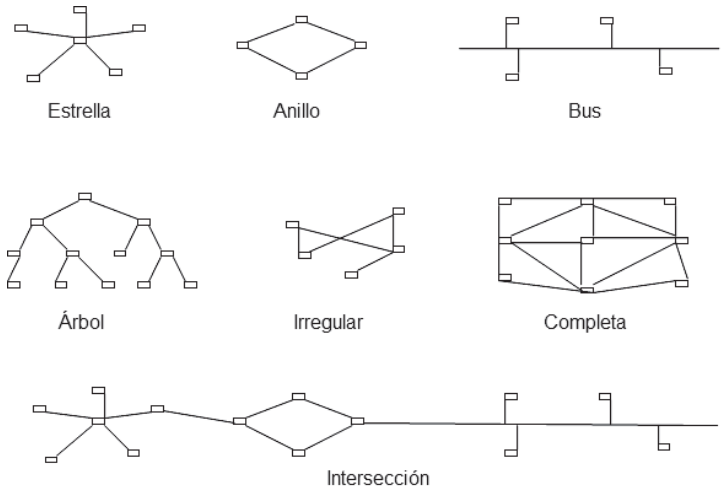
redes de cómputo, sus canales son relativamente de baja capacidad y de mayor margen de error.

2.5 TOPOLOGÍA DE REDES

La topología de redes se refiere al arreglo geométrico que tendrán las conexiones entre las computadoras, además, son una forma para clasificar las redes. Las topologías más usuales que se muestran en la figura 2.2 son:

- *Topología estrella*: Todas las computadoras se conectan a una computadora central. Esta topología no permite la comunicación directa entre dos computadoras que no sean la central.
- *Topología en anillo*: La red forma un anillo continuo en el cual puede viajar la información.
- *Topología en bus:* Emplea un solo medio llamado bus, al cual todas las computadoras se conectan.
- *Topología en árbol:* Existe una computadora principal que sirve como raíz y única salida externa.
- *Topología irregular:* No se respeta un modelo de conexión.
- *Topología de intersección:* En esta topología existe la conexión de dos o más tipos de topologías.
- *Topología completa:* Se dan todos los tipos de topologías.

Figura 2.2. Topologías más usadas para las redes de computadoras

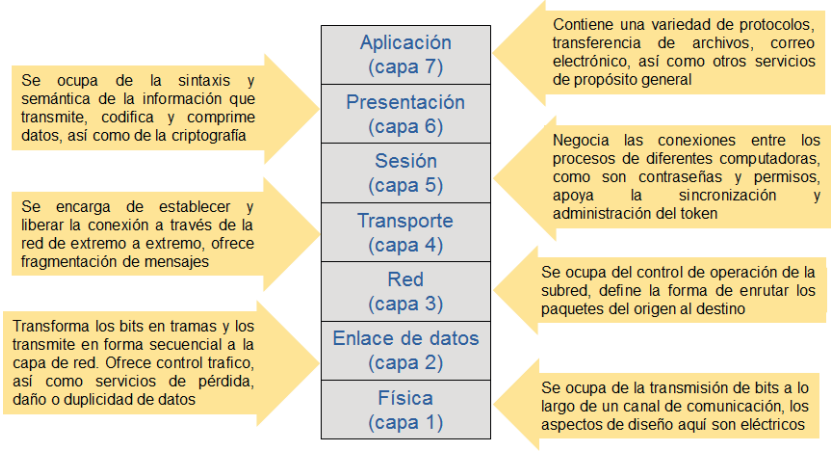


2.6 MODELO OSI

El advenimiento de los sistemas de cómputo propició un campo ideal para que los sistemas fueran abiertos, es decir que diferentes plataformas de diferentes fabricantes se pudieran comunicar. Sin embargo, antes de la década de 1970 esto no era posible porque los sistemas eran cerrados y los fabricantes seguían sus propios modelos de comunicación. Para afrontar este problema, entre 1977 y 1984 los especialistas del ISO (International Standard Organization) crearon el OSI (Open System Interconnection), un modelo de referencia para la interconexión de sistemas abiertos. El modelo ISO/OSI utiliza siete capas para organizar una red en módulos funcionales bien definidos (ver figura 2.3), con la cual los diseñadores puedan construir redes reales, sin embargo, al ser este solo un modelo y no una norma, el diseñador puede modificar el número, nombre y función de la red. Los principios aplicados para el establecimiento de siete capas fueron [Tanenbaum, 1997]:

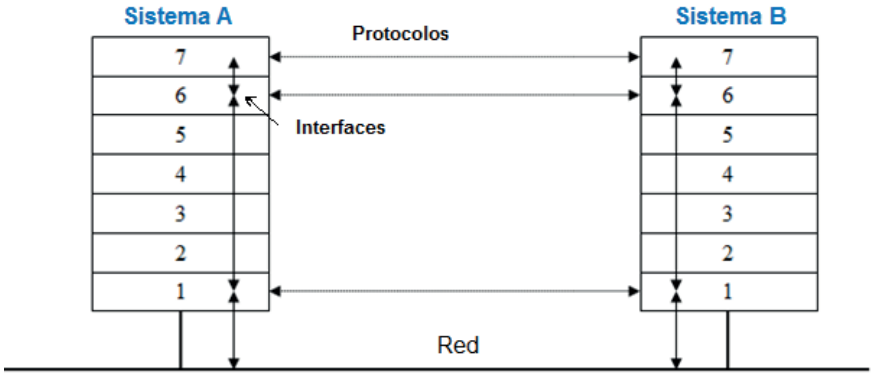
1. Una capa se creará en situaciones en las que se necesita un nivel diferente de abstracción.
2. Cada capa deberá efectuar una función bien definida.
3. La función que realizará cada capa deberá seleccionarse con la intención de definir protocolos normalizados internacionalmente.
4. Los límites de las capas deberán seleccionarse tomando en cuenta la minimización del flujo de información a través de las interfaces.
5. El número de capas deberá de ser lo suficientemente grande para que funciones diferentes no tengan que ponerse juntas en la misma capa y, por otra parte, también deberá de ser lo suficientemente pequeño para que su arquitectura no sea difícil de manejar.

Figura 2.3. Las capas del modelo OSI y su aplicación



Una vez establecido el modelo OSI, las computadoras se pueden comunicar como se ilustra en la figura 2.4. La comunicación se realiza de una capa de un nivel a otra capa del mismo nivel por medio de protocolos (líneas punteadas). Se usan interfaces para transmitir los datos de una capa a otra capa inferior dentro de una misma computadora; así, hasta llegar a la capa física y pasar a la red subiendo a la capa física de la otra computadora.

Figura 2.4. Comunicación entre dos computadoras usando el modelo OSI



2.7 NORMA IEEE-802

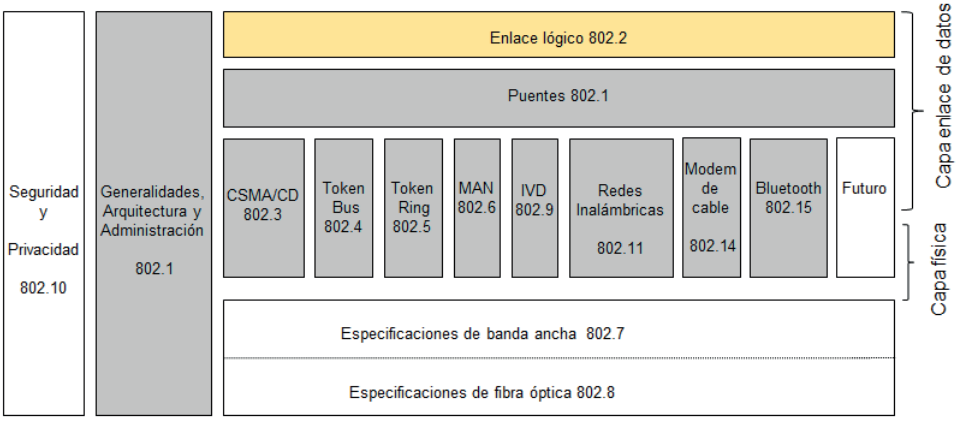
El estándar IEEE-802 del Instituto de Ingenieros Eléctricos y Electrónicos (IEEE) especifica las características que deben de cumplir las redes LAN y define las especificaciones para cada uno de estos, como se indica en la tabla 2.2 (sólo se muestran los estándares iniciales).

Tabla 2.2. Componentes del estándar IEEE-802 para LAN [Black, 1993]

Nombre del estándar	Descripción
802.1	Da una introducción al conjunto de normas y define las primitivas de interfaz
802.2	Describe la parte superior de la capa de enlace, la LLC
802.3	Especificaciones para la tecnología de acceso CSMA/CD (para Ethernet)
802.4	Especificaciones para la tecnología de testigo en bus
802.5	Especificaciones para la tecnología de testigo en anillo
802.6	Especificaciones para redes de área metropolitana
802.7	Especificaciones para redes de banda ancha
802.8	Especificaciones para redes de fibra óptica
802.9	Especificaciones para redes que integran datos y voz
802.10	Especificaciones para seguridad en redes
802.11	Especificaciones para redes inalámbricas

La arquitectura del estándar 802 se muestra en la figura 2.5.

Figura 2.5. El estándar IEEE 802 [adaptado de Black, 1993]



CSMA/CD: Acceso múltiple por detección de portadora con detección de colisión
MAN: Red de área metropolitana
IVD: Integración de voz y datos

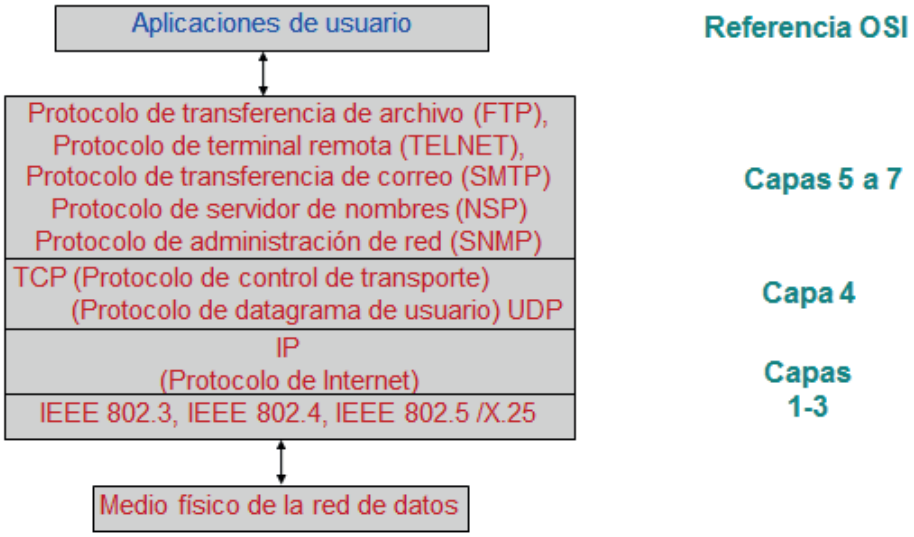
2.8 PROTOCOLOS Y PAQUETES

2.8.1 Protocolos

Los protocolos son un conjunto de reglas que gobiernan la interacción de procesos concurrentes en sistemas distribuidos, estos son utilizados en un gran número de campos como sistemas operativos, redes de computadoras o comunicación de datos.

Uno de los conjuntos de protocolos más usados en Internet y que usamos en los ejemplos de sockets es el TCP/IP (Transmission Control Protocol / Internet Protocol). Este conjunto de protocolos tiene menos capas que el OSI, lo que incrementa su eficiencia. TCP/IP es un protocolo confiable, ya que los paquetes son recibidos en el orden en que son enviados. Muchos sitios también usan el protocolo UDP/IP (User Datagrama Protocol/Internet Protocol). El protocolo UDP/IP es un protocolo no confiable, ya que no garantiza la entrega. La figura 2.6 ilustra las capas del protocolo TCP/IP y UDP/IP comparado con el modelo OSI.

Figura 2.6. Arquitectura de los protocolos TCP/IP y UDP/IP



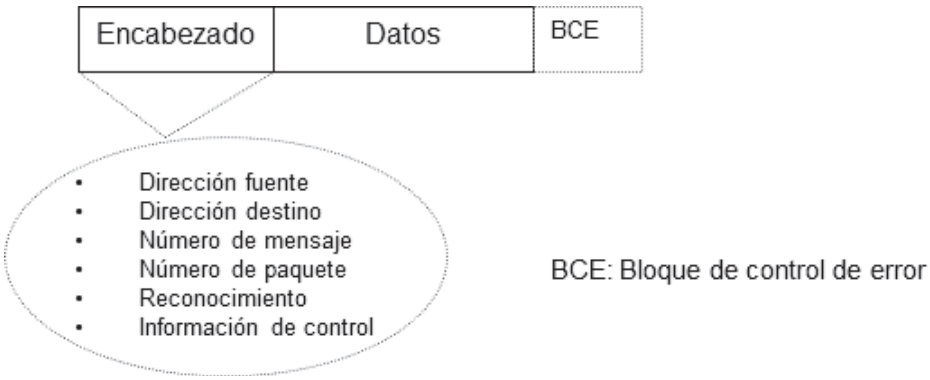
2.8.2 Paquetes

Un paquete es la forma usada para enviar información en un ambiente distribuido o de red. Cada mensaje es dividido y colocado en paquetes. Un paquete contiene toda la información necesaria para construir el mensaje original. Es decir que los paquetes pueden llegar en desorden, pero el nodo destino deberá de ser capaz de poner los paquetes en una forma ordena.

El segmento de datos del paquete contiene (ver figura 2.7):

- Los encabezados de cada capa del protocolo a partir de los datos de la capa de enlace.
- Los datos de la aplicación actual.

Figura 2.7. Formato típico de un paquete



Cuando los paquetes llegan a la capa de enlace de datos (capa 2 del modelo OSI), son entramados en un marco (trama) donde se adiciona un bloque de control de error. Aunque a veces se usan los términos de paquetes y marcos como equivalentes, esto no es correcto, ya que los términos se refieren a unidades en diferentes capas del modelo OSI (la de transporte y la de enlace de datos, respectivamente). Los estándares IEEE 802.3, IEEE-802.4 e IEEE 802.5, tienen su propio formato de marco para sus paquetes.

2.8.3 Redes de banda ancha

Muchas de las tecnologías emergentes se designan como redes multimedia de banda ancha. El propósito de una red de multimedia de banda ancha es proporcionar un servicio de transporte para cualquier tipo de aplicación. Una

red de banda ancha apoya el servicio telefónico, de video, de datos, compras a distancia, aplicaciones CAD/CAM, etc. De acuerdo con Black [1999], el término banda ancha ha sido usado desde hace varios años y se han propuesto algunas definiciones para la tecnología de banda ancha, como las siguientes:

- "Cualquier sistema de alta capacidad que ofrece un sistema de transporte de multimedia.
- Una red que utiliza tecnología de alta frecuencia como mecanismo de transporte para el tráfico de usuarios.
- Cualquier red que opere por encima del intervalo de frecuencia de la voz (0-4 kHz).
- Cualquier red que opere por arriba de la tasa primaria de ISDN (1.544 Mbit/s en Norteamérica y 2.048 Mbit/s en Europa)".

Dentro de las redes de banda ancha, la tecnología de red más popular es la tecnología ATM (Asynchronous Transfer Mode). La tecnología ATM está basada en los esfuerzos del grupo de estudio XVIII del ITU-T, al desarrollar el BISDN para la transferencia a alta velocidad de voz, video y datos a través de redes públicas.

El propósito de ATM es proporcionar una red con multiplexión y conmutación, alta velocidad y bajo retardo para apoyar cualquier tipo de tráfico de usuario, como aplicaciones de voz, datos o video [Ford, Lew, Spanier & Stevenson, 1998]. ATM segmenta y multiplexa el tráfico de usuario en unidades pequeñas de longitud fija llamadas celdas. La celda tiene 53 bytes, de los cuales 5 están reservados para el encabezado de la celda, los 48 bytes restantes se ocupan para datos. Cada celda se identifica con identificadores de circuitos virtuales contenidos en el encabezado. Una red ATM utiliza esos identificadores para encaminar el tráfico a través de computadoras de alta velocidad desde el equipo de las instalaciones del cliente (CPE) transmisor hasta el CPE receptor. ATM ofrece operaciones de detección de error limitada. Con la excepción del tráfico de señalización, ATM no ofrece servicios de retransmisión y son pocas las operaciones que se realizan con el pequeño encabezado. La intención de tener celdas con pocos servicios prestados es implementar una red que sea lo bastante rápida como para apoyar tasas de transferencia.

2.8.4 Redes inalámbricas

Actualmente la movilidad y el cómputo móvil emergen con gran fuerza. Millones de personas intercambian información cada día usando receptores de mensajes, tabletas electrónicas, teléfonos móviles y otros productos de comunicación inalámbrica. Con el auge de la telefonía inalámbrica y los servicios de mensajería, es una tendencia el que la comunicación inalámbrica se aplique al campo de la computación personal y de los negocios. La gente desearía tener facilidades de acceso y compartir información en una escala global y cercana en cualquier sitio en que se encuentre. El propósito de un sistema de comunicaciones móviles se puede inferir del nombre de la tecnología, que es la de prestar servicios de telecomunicaciones entre estaciones móviles y estaciones terrenas fijas, o entre dos estaciones móviles [Black, 1999]. Se pueden distinguir dos formas de comunicación móviles: celular e inalámbrica.

EJERCICIOS

1. ¿Cuál es la diferencia entre una red LAN y MAN?
2. ¿Cuál es la función del protocolo IEEE 802.11?
3. ¿Qué ventajas tiene usar celdas en comparación con usar paquetes en la comunicación de datos?
4. ¿En qué consiste el sistema GSM y cuáles son sus principales componentes?
5. ¿Cuál es la similitud entre una topología en árbol y una de estrella?
6. Investiga las características de los medios físicos de comunicación para redes de cómputo.
7. ¿Qué esquemas de comunicación se utilizan para las redes satelitales?
8. En el modelo OSI, ¿cuál es la diferencia entre un protocolo y una interfaz?
9. ¿Qué tipos de redes existen en la Internet?
10. ¿Cuál es la principal desventaja de una topología en anillo?

11. ¿Qué beneficios aporta usar una topología de árbol en una red de difusión de contenidos?

12. ¿Qué beneficios aporta usar una topología completa o de malla en una red de datos?

13. Considera que dos computadoras transmiten paquetes de 1,500 bytes por un canal compartido que opera a 128,000 bits por segundo. Si las computadoras tardan 90 microsegundos entre la terminación de transmisión de una computadora y el inicio de otra, ¿qué tiempo se requiere para que una computadora envíe un archivo de 5 MB a la otra computadora?

ACTIVIDAD INTEGRADORA

1. De la aplicación de los contenidos, así como de las habilidades y actitudes desarrolladas.
   Se recomienda exposiciones por parte del profesor, así como discusiones en grupo sobre las diferentes topologías de las redes de comunicación y sus componentes. Hacer hincapié en el modelo de referencia OSI y su importancia para los sistemas abiertos. El capítulo puede servir también como un repaso para los alumnos que ya cursaron la UEA Arquitectura de redes (Modelo OSI de ISO). Como repaso, se recomienda resumirlo en una o dos de sesiones de clases con actividades extraclase para los alumnos.
   El alumno debe de mostrar interés por entender el principio de operación de las redes de comunicación de datos, ya que son la infraestructura que soporta el cómputo distribuido.
2. Del logro de los objetivos establecidos en el capítulo o apartado del material.
   Se espera alcanzar los objetivos de este capítulo con el apoyo en la solución de los ejercicios que se incluyen al final de este capítulo.

Capítulo 3. Modelos de arquitecturas

Objetivo: Que el alumno comprenda los diferentes paradigmas de cómputo, desde el cliente-servidor al grid, y cómo se integran en los modelos arquitectónicos de los sistemas distribuidos.

3.1 Introducción

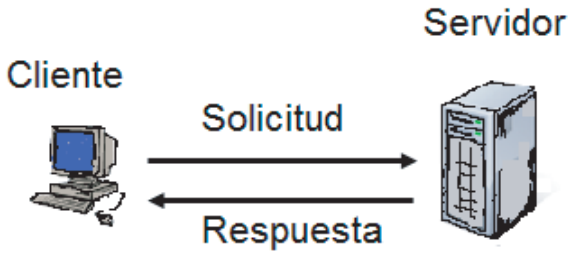
La arquitectura de un sistema es su estructura en términos de los componentes especificados por separado y sus interrelaciones. El objetivo de una arquitectura general es asegurar que la estructura reunirá presentes y probables futuras demandas sobre el mismo. Las principales preocupaciones son que el sistema sea fiable, manejable, adaptable y rentable [Coulouris *et al.*, 2012]. La arquitectura de un sistema distribuido guarda algunos aspectos similares con el diseño arquitectónico de un edificio, los cuales determinan no solo su apariencia, sino también su estructura general y el estilo arquitectónico (gótico, neoclásico y moderno) y proporciona un marco coherente de referencia para el diseño. Todos los tipos de sistemas distribuidos tienen características básicas comunes. Un modelo de arquitectura es una descripción abstracta simplificada pero consistente de cada aspecto relevante del diseño de un sistema distribuido. Este capítulo describe los principales modelos arquitectónicos de los sistemas distribuidos, particularmente en paradigmas cliente-servidor, peer-to-peer, grid, proxy, cluster y las principales diferencias entre estos.

3.2 Modelo cliente - servidor

El modelo cliente-servidor es la arquitectura más citada cuando se discuten los sistemas distribuidos. Es el modelo más importante y sigue siendo el más

ampliamente utilizado. La figura 3.1 ilustra la estructura simple de esta arquitectura, en la cual los procesos toman el rol de ser clientes o servidores. En particular, los procesos de cliente interactúan con los procesos de servidor individuales en equipos anfitriones (host) potencialmente separados, con el fin de acceder a los recursos compartidos que administran.

Figura 3.1. Ejemplo de una estructura simple cliente-servidor



El modelo cliente-servidor puede tomar diferentes configuraciones. Por ejemplo, puede existir más de un cliente conectado a un servidor, como se indica en la figura 3.2. También se puede tener un grupo de servidores interconectados dedicados a dar servicio a un grupo de clientes. Este escenario se indica en la figura 3.3.

Figura 3.2. Ejemplo de estructura cliente-servidor para dos clientes

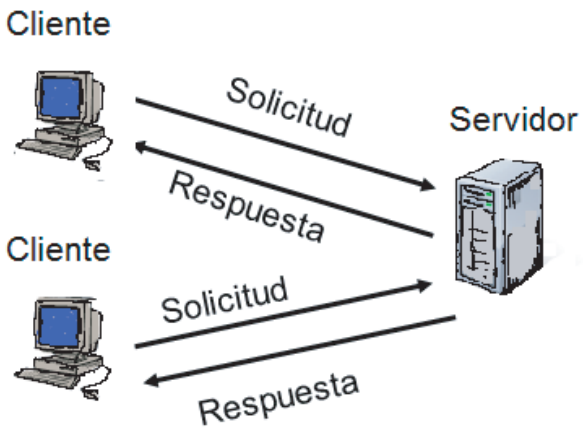
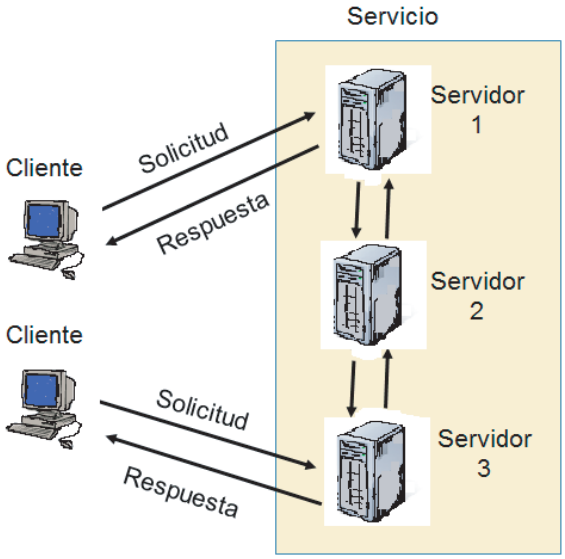


Figura 3.3. Grupo de servidores interconectados
basado en el modelo cliente-servidor



3.3 PROXY

Es un servidor que se emplea como intermediario entre las peticiones de recursos que realiza un cliente a otro servidor. Por ejemplo, si una computadora *A* solicita un recurso a una computadora *C*, lo hará mediante una petición a la computadora *B* que, a su vez, trasladará la petición a la computadora C. De esta manera, la computadora *C* no sabrá que la petición procedió originalmente de la computadora *A*. Esta situación estratégica de punto intermedio suele ser aprovechada para soportar una serie de funcionalidades, como:

- Proporcionar caché.
- Control de acceso.
- Registro del tráfico.
- Prohibir cierto tipo de tráfico.
- Mejorar el rendimiento.
- Mantener el anonimato.

El proxy más conocido es el servidor proxy web, su función principal es interceptar la navegación de los clientes por páginas web por motivos de seguridad, rendimiento, anonimato, entre otros. Las figuras 3.4 y 3.5 muestran dos ejemplos del uso de proxy.

Figura 3.4. Arreglo de proxy cliente y proxy servidor
para acceder al servidor desde dos clientes

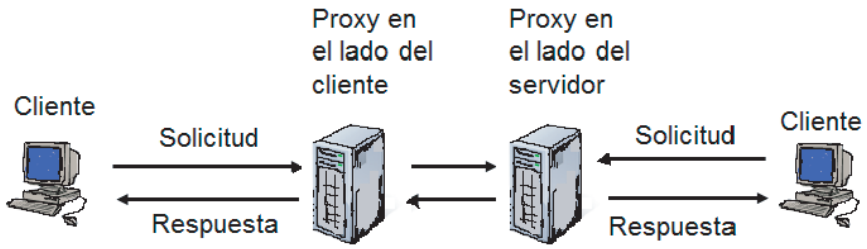
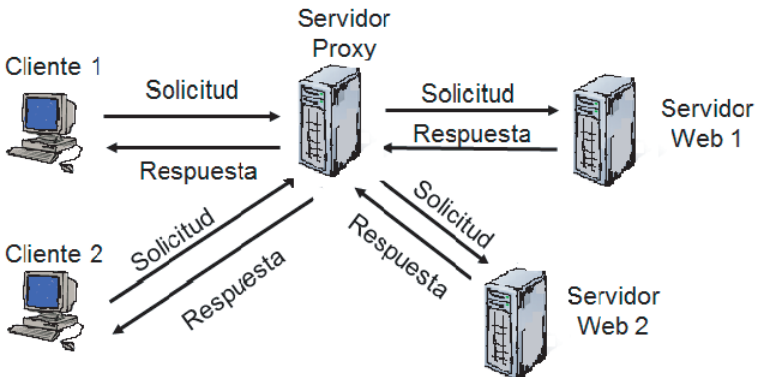


Figura 3.5. Acceso a servidores web vía un proxy



3.4 PEER-TO-PEER

El paradigma peer-to-peer (P2P) ha sido un tema muy atractivo para muchos investigadores de diferentes áreas, tales como redes, sistemas distribuidos, teoría de la complejidad, bases de datos y otros.

En el modelo cliente-servidor tradicional, dos tipos de nodos son empleados: clientes y servidores. En este contexto, los clientes solo solicitan servicios y el servidor solo proporciona a los clientes el servicio apropiado. Un servidor puede aceptar varias solicitudes, procesarlas y devolver los contenidos solicitados a los clientes. En la Internet actual, los clientes incluyen navegadores web, clientes de chat en línea y clientes de correo electrónico, mientras que los servidores normalmente son servidores web, servidores FTP y servidores de correo.

En contraste, en los sistemas P2P no se requiere una infraestructura dedicada. Los servidores dedicados y clientes no existen, ya que cada peer puede tomar el papel tanto de servidor como de cliente al mismo tiempo. Una ventaja

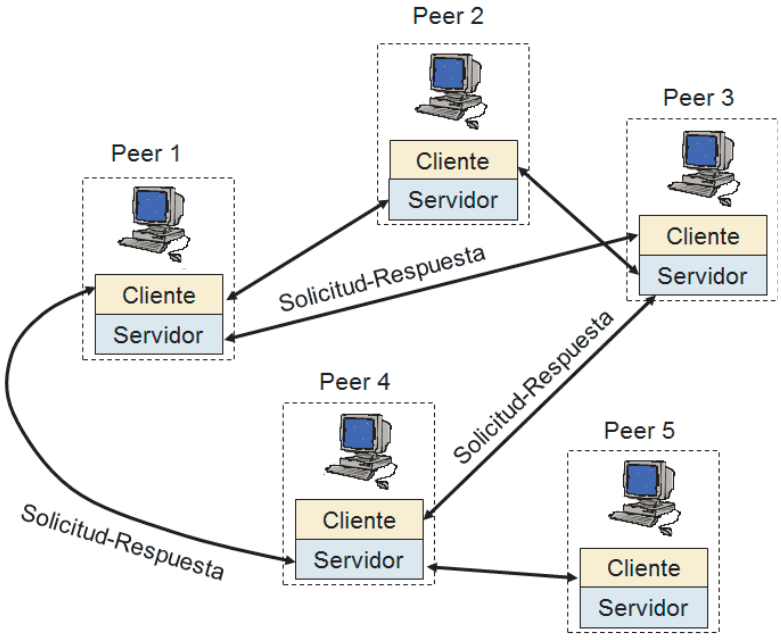
importante de los sistemas peer-to-peer es que todos los recursos disponibles son proporcionados por los peers. Durante la distribución de un contenido, los peers aportan sus recursos para transmitir el contenido a los demás peers. Por lo tanto, cuando un nuevo peer se agrega al sistema al sistema P2P, la demanda se incrementa pero la capacidad general del sistema también. Esto no es posible en un modelo cliente-servidor con un número fijo de servidores.

El paradigma P2P se muestra en la figura 3.6, donde se puede ver que en un peer están ejecutándose al mismo tiempo tanto un proceso servidor como uno cliente, también ambos ofrecen y solicitan respectivamente servicios a otros procesos similares en otros peers.

Beneficios de un sistema peer-to-peer:

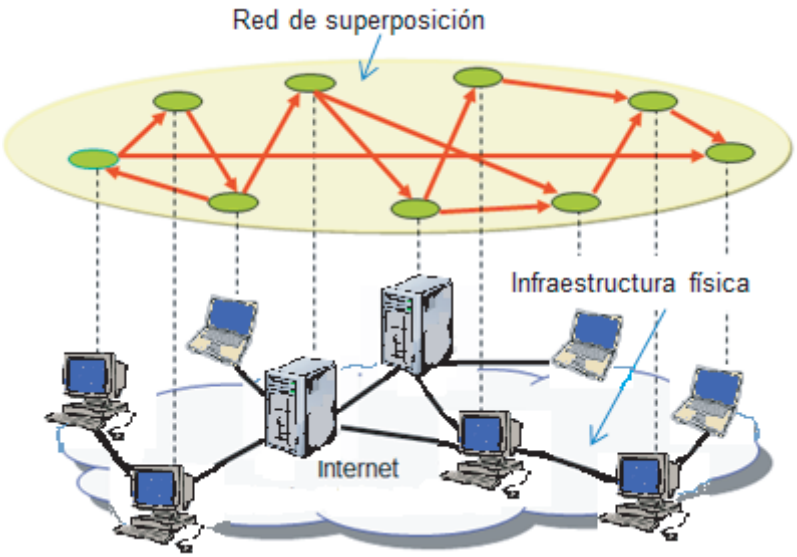
- Nodos comparten recursos.
- Se pueden desplegar algoritmos distribuidos.
- Escalamiento más fácil del sistema.
- Ahorro de costos.
- Flexibilidad.
- Ningún punto único de falla.
- Mayor robustez del sistema.

Figura 3.6. Paradigma peer-to-peer



Una infraestructura de comunicación P2P está formada por un grupo de nodos ubicados en una red física. Estos nodos construyen una abstracción de red en la parte superior de la red física conocida como red superpuesta, que es independiente de la red física subyacente. La figura 3.7 muestra este escenario. La red superpuesta se establece para cada sistema P2P a través de conexiones TCP o HTTP. Debido a la capa de abstracción de la pila de protocolo TCP, las conexiones físicas no son reflejadas por la red de superposición. La red superpuesta construye túneles lógicas entre pares de nodos [Ripeanu, Foster, Iamnitchi & Rogers, 2007], con el fin de implementar su propio mecanismo de enrutamiento para transportar sus mensajes.

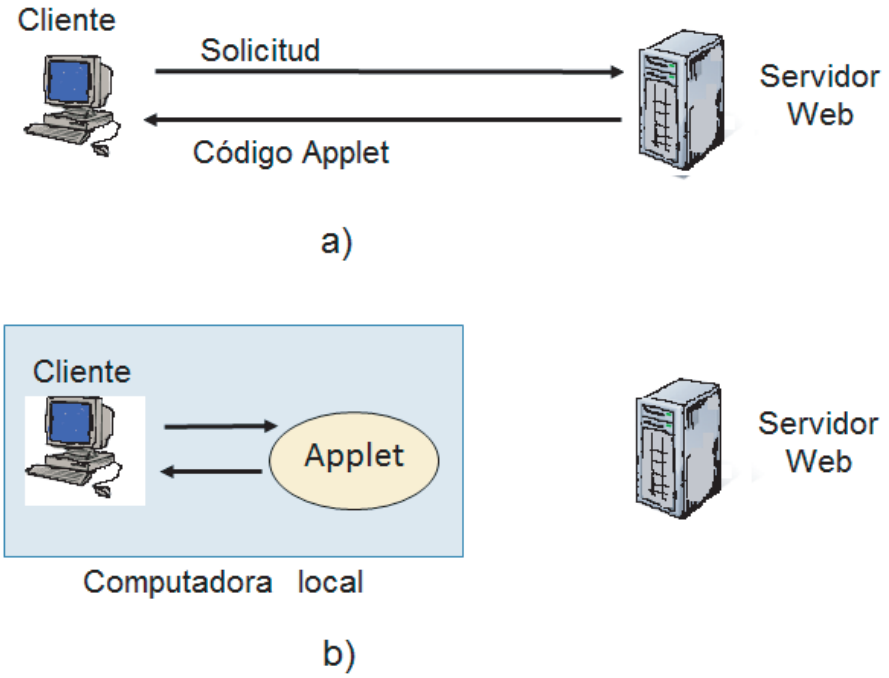
Figura 3.7. Ejemplo de una red superpuesta peer-to-peer [López-Fuentes, 2009]



3.5 APPLETS

Un *applet* es un código que se ejecuta en el contexto de otro programa, por ejemplo, en un navegador web. El código se descarga en el navegador y se ejecuta allí, como se muestra en la figura 3.8.

Figura 3.8. a) A solicitud del cliente el servidor web, responde con el código del applet. b) El cliente interactúa con el applet (adaptado de [Coulouris et al., 2012])



Un applet normalmente lleva a cabo una función muy específica, que carece de uso independiente, y son ampliamente utilizados en aplicaciones de telefonía móvil. Un applet puede dar una buena respuesta interactiva, ya que no sufre de los retrasos o variabilidad de ancho de banda asociado con la comunicación de la red. Sin embargo, un applet típicamente carece de sesión y tiene privilegios restringidos de seguridad. A menudo, un applet consiste en un código poco confiable, por eso se les impide tener acceso al sistema de archivos local. Los applet que se cargan a través de la red con frecuencia son considerados como códigos de poca confianza [Flanagan, 1998], a excepción de que lleven la firma digital de una entidad especificada como confiable. Ejemplos de los applets más comunes son:

- Java applets.
- Animaciones Flash.
- Windows media player.
- Modelos 3D.

3.6 CLÚSTER

En informática, el término clúster ("grupo" o "racimo") hace referencia a conjuntos o conglomerados de computadoras construidos mediante el uso de hardware común y que se comportan como si fueran una única computadora. Un ejemplo de clúster se muestra en la figura 3.9. El uso de los clústeres varía desde las aplicaciones de supercómputo, servidores web y comercio electrónico hasta el software de misiones críticas y bases de datos de alto rendimiento. El cómputo con clústeres es el resultado de la convergencia de varias tendencias tecnológicas actuales, entre las que se pueden destacar:

- Microprocesadores de alto rendimiento.
- Redes de alta velocidad.
- Software para cómputo distribuido de alto rendimiento.
- Crecientes necesidades de potencia computacional.

Los servicios esperados de un clúster principalmente son:

- Alto rendimiento.
- Alta disponibilidad.
- Escalabilidad.
- Balanceo de carga.

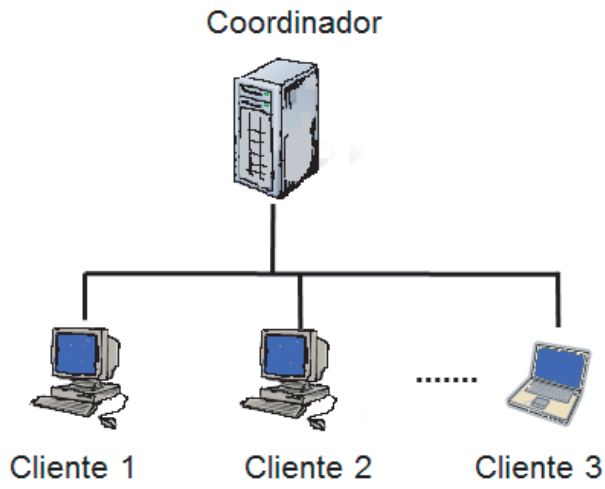
Típicamente respecto a la rapidez y disponibilidad, se espera que un clúster sea más económico que el uso de computadoras individuales.

Un clúster puede ser:

- Homogéneo.
- Semihomogéneo.
- Heterogéneo.

Un clúster es homogéneo cuando todas las computadoras tienen la misma configuración en hardware y sistema operativo. Es semihomogéneo cuando las computadoras tienen diferente rendimiento pero guardan una similitud con respecto a su arquitectura y sistema operativo. Finalmente, un clúster es heterogéneo cuando las computadoras tienen diferente hardware y sistema operativo.

Figura 3.9. Ejemplo de clúster



3.7 GRID

El cómputo grid es un paradigma del cómputo distribuido, frecuentemente usado para indicar una infraestructura de gestión de recursos distribuidos que se centra en el acceso coordinado a los recursos informáticos remotos [Foster & Kesselman, 1999]. Estos recursos de cómputo son colectados desde múltiples localizaciones para alcanzar una meta común. A diferencia del cómputo de cluster (en grupo o racimo), el cómputo grid tiende a ser más heterogéneo y disperso geográficamente. Generalmente las grids son usadas para una variedad de propósitos pero puede haber grids especializadas para fines específicos. Los recursos que son integrados por una infraestructura grid son típicamente plataformas de cómputo dedicadas a supercomputadoras de alta gama o clústers de propósito general.

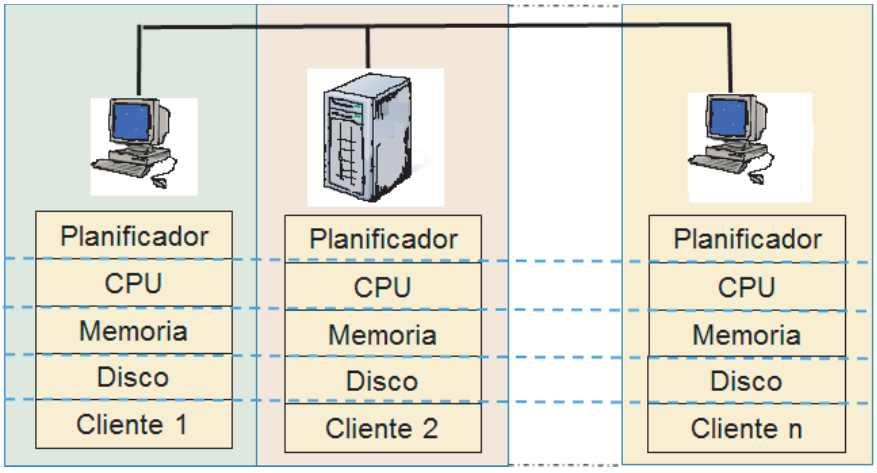
Algunos autores [Foster & Kesselman, 2013] ubican como antecedente del cómputo grid al sistema de tiempo compartido Multics. Sin embargo, este concepto ha sufrido múltiples transformaciones a lo largo de los años y diversas definiciones sobre el cómputo grid pueden ser encontradas en la literatura. Jacob, Brown, Fukui y Trivedi [2005] definen la computación grid como cualquiera de una variedad de niveles de virtualización a lo largo de un continuo, donde a lo largo de ese continuo se podría decir que una solución particular es una implementación de cómputo grid frente a una relativamente simple aplicación usando recursos virtuales, pero incluso en los niveles más simples de virtualización, siempre se requieren habilitar tecnologías de redes.

Beneficios del cómputo grid [Jacob *et al.*, 2005]:

- Explotación de recursos infrautilizados.
- Capacidad de CPU paralelos.
- Recursos virtuales y organizaciones virtuales para la colaboración.
- Acceso a recursos adicionales.
- Balanceo de recursos.
- Fiabilidad.
- Mejor gestión de infraestructuras de TI más grandes y distribuidos.

La conceptualización del cómputo grid se muestra en la figura 3.10.

Figura 3.10. Ejemplo de cómputo grid



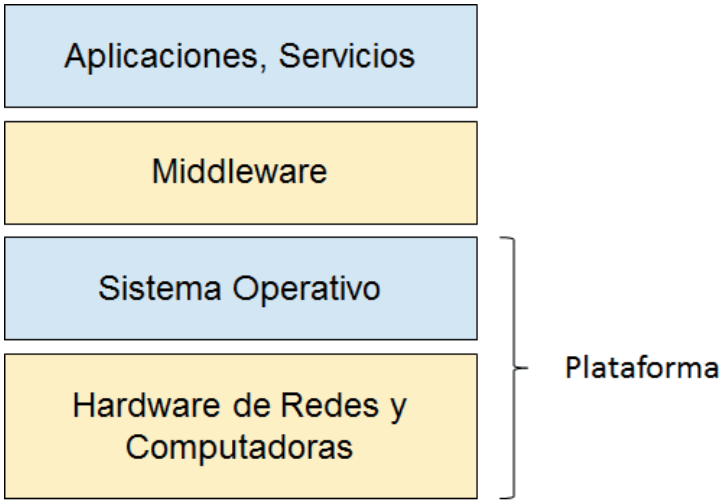
3.8 ARQUITECTURA DE CAPAS

Una arquitectura de capa resulta familiar en los sistemas distribuidos y está relacionado con la abstracción. Con este enfoque, un sistema complejo puede ser dividido en cierto número de capas, donde las capas superiores hacen uso de los servicios ofrecidos por las capas inferiores. De esta manera, una determinada capa ofrece una abstracción de software, sin que las capas superiores o inferiores a esta deban de estar al tanto de los detalles de implementación.

En el caso de los sistemas distribuidos, los servicios se organizan de una manera vertical como capas de servicio. Un servicio distribuido puede ser proporcionado por uno o más procesos del servidor, que interactúan entre sí y con los procesos de cliente para mantener una visión de todo el sistema,

coherente de los recursos del servicio. Por ejemplo, el ajuste de hora entre clientes puede realizarse por un servicio de hora de red a través de Internet, usando el protocolo de Tiempo de Red (NTP). Es útil organizar este tipo de servicio en capas debido a la complejidad de los sistemas distribuidos. Una visión común de una arquitectura de capa se muestra en la figura 3.11.

Figura 3.11. Capas de servicio en un sistema distribuido [Coulouris *et al.*, 2012]



Como se indica en la figura 3.11, un sistema distribuido está constituido principalmente por los siguientes estratos:

- Plataforma.
- Middleware.
- Aplicaciones y servicios.

La *plataforma* para sistemas y aplicaciones distribuidas se compone de las capas de hardware y software de nivel más bajo. Estas capas de bajo nivel proporcionan servicios a las capas superiores, las cuales son implementadas de manera independiente en cada equipo, conduciendo a la interfaz de programación del sistema hasta un nivel que facilita la comunicación y la coordinación entre los procesos. Ejemplos de plataformas son:

- Intel x86/Windows.
- Intel x86/Mac OS X.
- Intel x86/Linux.
- Intel x86/Solaris.

Middleware es un software que tiene como función principal enmascarar la heterogeneidad del sistema distribuido para proporcionar un modelo de programación conveniente a los programadores de aplicaciones. Ejemplos de middleware son:

- CORBA (Common Object Request Broker).
- Java RMI (Java Remote Method Invocation).

Finalmente, la capa de aplicaciones y servicios son las prestaciones que ofrece el sistema distribuido a los usuarios. Se entiende como las aplicaciones distribuidas.

3.9 MIDDLEWARE

En la primera generación de los sistemas distribuidos todos los servicios proporcionados por los servidores debían de programarse a la medida. Así, servicios de acceso a bases de datos, de impresión y transferencias de archivos tenían que ser desarrollados por las propias aplicaciones. Queda en evidencia la necesidad de crear servicios de uso más común por las aplicaciones, de tal manera que pueda incluirse en todas las aplicaciones como software prefabricado. En este escenario surge la idea del middleware, representado por estándares tales como ODBC, OLE, DCOM y CORBA.

Middleware es un conjunto de servicios que permite distribuir datos y procesos a través de un sistema multitarea, una red local, una red remota o Internet [Martínez Gomaríz, 2014]. Los servicios del Middleware pueden ser clasificados en dos grandes grupos:

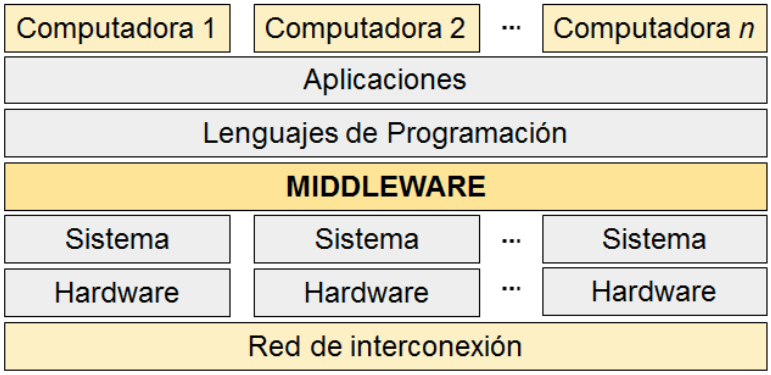
- Servicios de desarrollo.
- Servicios de administración.

El objetivo principal del middleware es conseguir la transparencia en los sistemas distribuidos, por medio de:

- Ofrecer la capacidad, así como solicitar y recibir de manera transparente al sistema.
- Liberar a los diseñadores y administradores del sistema de problemas derivados de la complejidad del sistema operativo.

En la práctica, middleware es representado por procesos u objetos en un conjunto de equipos que interactúan entre sí para implementar la comunicación y el intercambio de recursos de soporte para las aplicaciones distribuidas [Coulouris *et al.*, 2012]. El middleware está relacionado con el suministro de materiales de construcción útiles para la construcción de componentes de software que pueden trabajar con otros en un sistema distribuido. Las abstracciones del middleware apoyan a diversas actividades de comunicación, como la invocación de método remoto, la comunicación entre un grupo de procesos, notificación de eventos, el particionamiento, la colocación y recuperación de objetos de datos compartidos entre los equipos cooperantes, la replicación de objetos de datos compartidos y la transmisión de datos multimedia en tiempo real. Un escenario del middleware es mostrado en la figura 3.12.

Figura 3.12. Escenario del middleware en un sistema distribuido



A pesar de que el middleware tiene como objetivo suministrar transparencia de distribución, en general las soluciones específicas deben de ser adaptables a los requerimientos de las aplicaciones. Tanenbaum y Van Steen [2008] consideran que una solución para este problema es desarrollar diversas versiones de un sistema middleware, donde cada versión sea confeccionada para una clase específica de aplicaciones.

3.10 CORBA

El paradigma orientado a objetos juega un importante rol en el desarrollo de software y cuenta con gran popularidad desde su introducción. La orien-

tación a objetos se comenzó a utilizar para el desarrollo de sistemas distribuidos en la década de 1980. El Grupo de Gestión de Objetos (OMG-Object Management Group) se creó en 1989 como una asociación de las empresas líderes de la tecnología software para definir especificaciones que puedan ser implementadas por ellas y que faciliten la interoperabilidad de sus productos. Los middlewares basados en componentes se han convertido en una evolución natural de los objetos distribuidos, debido a que apoyan la gestión de las dependencias entre componentes, ocultando los detalles de bajo nivel asociados con el middleware, gestión de las complejidades de establecer aplicaciones distribuidas con propiedades no funcionales apropiadas –como la seguridad– y el apoyo apropiado de estrategias de implementación. Los middlewares basados en objetos distribuidos están diseñados para proporcionar un modelo de programación basado en principios orientados a objetos y, por tanto, para llevar los beneficios del enfoque a objetos para la programación distribuida. Los principales ejemplos de middleware basados en objetos distribuidos incluyen Java RMI y CORBA.

CORBA (Common Object Request Broker Architecture) es una herramienta middleware que facilita el desarrollo de aplicaciones distribuidas en entornos heterogéneos tanto en hardware como en software [Coulouris et al., 2012], ejemplos de estos son:

- Distintos sistemas operativos (Unix, Windows, MacOS, OS/2).
- Distintos protocolos de comunicación (TCP/IP, IPX).
- Distintos lenguajes de programación (Java, C, C++).
- Distinto hardware.

El objetivo principal de CORBA es especificar un middleware para construir aplicaciones del tipo cliente-servidor multi-nivel, distribuidas y centralizadas, y que sean flexibles y extensibles. Para alcanzar estos objetivos, CORBA utiliza diferentes medios, como los que a continuación se describen.

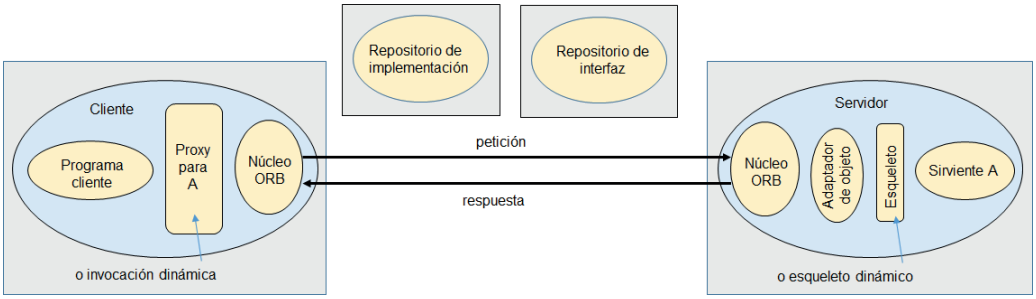
Del paradigma orientado a objeto explota los conceptos de encapsulación, herencia y polimorfismo. En la comunicación a través de invocar métodos remotos es más fácil de programar que los sockets, RPC, etc. Usa el concepto de esqueleto como el encargado de la comunicación con el cliente. Define la separación interfaz-implementación a través de CORBA IDL (Interface Definition Language). Para la referencia a objeto, usa el identificador único de un objeto. CORBA es implementado encima del protocolo TCP/IP y usa modos de comunicación asíncrona y síncrona. Con el uso de envolturas (wrappers), CORBA permite integrar los sistemas heredados, y normalmente

usa solo una instancia de cada clase. CORBA también hace uso de una capa de software conocida como ORB (Object Request Broker), que permite a los objetos realizar llamadas a métodos situados en máquinas remotas a través de una red. Además, ORB maneja la transferencia de estructuras de datos de manera que sean compatibles entre los dos objetos. ORB es un componente fundamental de la arquitectura CORBA y su misión es facilitar la comunicación entre objetos. ORB también debe de facilitar diferentes transparencias, tales como de localización, implementación, comunicación y ejecución.

En CORBA [Coulouris *et al.*, 2001], el servidor crea objetos remotos, hace referencias accesibles a objetos remotos y espera a que los clientes invoquen a estos objetos remotos o a sus métodos. Por su parte, el cliente obtiene una referencia de uno o más objetos remotos en el servidor e invoca a sus métodos.

La arquitectura CORBA (ver figura 3.13) está diseñada para dar soporte al rol de un intermediario de peticiones de objetos que permita que los clientes invoquen métodos en objetos remotos, donde tanto los clientes como los servidores pueden implementarse en diversos lenguajes de programación.

Figura 3.13. Principales componentes
de la arquitectura CORBA [Coulouris *et al.*, 2001]



La función de los principales componentes se describen a continuación [Coulouris *et al.*, 2001]. El *núcleo de ORB* es un módulo de comunicación el cual permite una interfaz que incluye las operaciones de arranque y paro, las operaciones para la conversión entre referencias a objetos remotos y cadenas de texto, así como las operaciones para obtener listas de argumentos para llamadas que emplean invocación dinámica. El *adaptador de objeto* sirve como puente entre los objetos con interfaces IDL y las interfaces IDL, además de las interfaces del lenguaje de programación de las correspondientes clases sirvientes. Las clases *esqueleto* se generan en el lenguaje del

servidor a través de un compilador de IDL. El *esqueleto* sirve para despachar las invocaciones a los métodos remotos, asimismo desempaqueta los argumentos desde los mensajes de petición y empaqueta las excepciones y los resultados en mensajes de respuesta. Los *resguardo/proxy* son los encargados de empaquetar los argumentos de los mensajes de invocación y desempaqueta las excepciones y los resultados de las respuestas. Cada *repositorio de implementación* activa los servidores registrados bajo demanda y localiza los servidores que están en ejecución en cada momento usando el nombre del adaptador de objeto. El *repositorio de interfaz* proporciona información sobre las interfaces IDL registradas a los clientes y a los servidores que lo requieran. La *interfaz de invocación dinámica* permite que los clientes puedan realizar invocaciones dinámicas sobre objetos remotos CORBA cuando no es práctico el uso de proxies. La *interfaz de esqueleto dinámica* permite que un objeto CORBA acepte invocaciones sobre una interfaz para la que no hay un esqueleto. El código delegado permite convertir a objetos CORBA aquellos fragmentos de código existente que fue diseñado sin prever los objetos distribuidos.

EJERCICIOS

1. Ilustra la arquitectura cliente-servidor para una comunicación de muchos a muchos.

2. Cita un ejemplo del uso de un proxy en una arquitectura distribuida.

3. Cita al menos tres desventajas del modelo cliente-servidor.

4. Cita al menos tres características de la arquitectura peer-to-peer.

5. Explica cuál es la función del middleware en los sistemas distribuidos.

6. ¿Cómo está involucrado el uso del caché en los sistemas distribuidos?

7. ¿Cuál es la diferencia entre una red superpuesta y una red física?

8. ¿Qué beneficio aporta trabajar con CORBA en los sistemas distribuidos?

9. Investiga las limitantes que tiene CORBA.

10. ¿Cuál es la diferencia entre una arquitectura grid y un clúster?

11. ¿Cuál es la principal diferencia entre el paradigma cliente-servidor y el paradigma peer-to-peer?

Actividad integradora

1. De la aplicación de los contenidos, así como de las habilidades y actitudes desarrolladas.
   Se recomienda exposiciones por parte del profesor, además de discusiones en grupo sobre las principales arquitecturas en los sistemas distribuidos. El profesor se puede apoyar de ejemplos que existan en la literatura para explicar mejor las ventajas y desventajas de cada modelo. También se recomienda encargar al alumno que estudie artículos científicos los cuales le permitan profundizar su conocimiento con respecto a las nuevas tendencias en las arquitecturas del cómputo distribuido. Un promedio de tres a cuatro sesiones podrían ser requeridas para cubrir los temas de este capítulo.
   El alumno debe de mostrar interés por identificar la utilidad de cada arquitectura para aplicaciones específicas, así como la posibilidad de combinar estas arquitecturas en escenarios de cómputo distribuido modernos.
2. Del logro de los objetivos establecidos en el capítulo o apartado del material.
   Se espera alcanzar los objetivos de este capítulo con el apoyo en la solución de los ejercicios que se incluyen al final de este capítulo, así como reforzar su conocimiento con lecturas de artículos científicos especializados sobre las tendencias del cómputo distribuido.

Capítulo 4. Procesos y comunicación

Objetivo: Que el alumno comprenda los diferentes conceptos relacionados con los procesos y la comunicación distribuidos como la base para el desarrollo de aplicaciones distribuidas.

4.1 Introducción

La comunicación entre procesos es el núcleo de todos los sistemas distribuidos [Tanenbaum & Van Steen, 2008], por tal razón es importante entender la manera en que los procesos localizados en diferentes computadoras pueden intercambiar información. En los sistemas distribuidos tradicionalmente la comunicación está basada en el paso de mensaje. Esta técnica aporta sincronización entre procesos y permite la exclusión mutua, su principal característica es que no requiere memoria compartida, por lo que resulta ser muy importante en la programación de sistemas distribuidos. En este capítulo revisamos diferentes conceptos relacionados con los procesos y la comunicación en los sistemas distribuidos, como hilos, clientes, servidores, la llamada a un procedimiento remoto (RPC), el paradigma cliente-servidor, la comunicación en grupo y la interfaz de sockets.

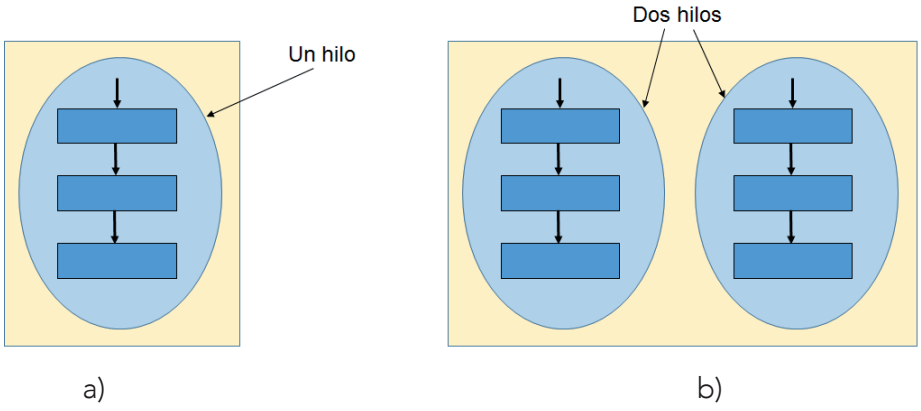
4.2 Hilos

Los hilos se diferencian de los procesos en que los primeros comparten los mismos recursos del programa que las contiene, en tanto los procesos tienen de manera separada su código, así como sus datos. Se pueden identificar hilos de dos tipos de flujo:

- *Flujo único*: En este caso, un programa utiliza únicamente un hilo para controlar su ejecución.
- *Flujo múltiple*: Son aquellos programas que utilizan varios *contextos de ejecución* para realizar su trabajo.

En un sistema multihilos, cada tarea se inicia y termina tan pronto como sea posible, esto facilita la entrada de datos en sistemas en tiempo real, especialmente si estos datos provienen de diferentes fuentes. En un programa multihilo se tiene el hilo principal del programa en ejecución, quien a su vez tiene otros hilos o tareas paralelas en ejecución. Un hilo se define como una secuencia única de control de flujo dentro de un programa, en un programa puede haber más de una secuencia de control o hilos. Un hilo es una parte del programa que se ejecuta independientemente del resto. El hilo es la unidad de código más pequeña que se pueda ejecutar en un entorno multitareas. El uso de hilos permite al programador escribir programas más eficientes, debido a que los hilos permiten optimizar recursos tan importantes como el mejor desempeño del CPU al minimizar sus tiempos de inactividad. El uso de hilos es muy valioso en entornos interactivos en red, ya que permiten sincronizar la diferencia entre la velocidad de transmisión de la red con las de procesamiento del CPU. La velocidad en el manejo del sistema de archivos para lectura y grabación es más lenta comparada con la velocidad de procesamiento de estos datos por el CPU, en este caso el uso de hilos ayuda mucho. Una de las razones de importancia para el estudio de la programación multihilos es que permite acceder a los recursos de tiempo libre de la CPU mientras se realizan otras tareas. La figura ilustra esquemáticamente un programa de un hilo y un programa de dos hilos.

Figura 4.1. Comparación entre hilos:
a) Programa con un solo hilo, b) Programa multihilos



4.3 Cliente

En un sistema distribuido, el cliente es el elemento que solicita y usa el servicio que proporciona una funcionalidad específica o dato. El cliente tiene una postura proactiva, esto quiere decir que está trabajando en una tarea específica y cuando necesita cierto dato o una operación específica invoca al servidor para obtenerlo. Generalmente es por medio de la aplicación cliente que un usuario accede y mantiene un diálogo con el sistema. El usuario realiza el diálogo vía una interfaz gráfica de usuario. La operación del cliente consiste en arrancar, realizar su trabajo y terminar.

4.4 Servidores

El servidor es el elemento que proporciona la funcionalidad o servicio en un sistema distribuido. Este servicio puede consistir en compartir datos, informar sobre una solicitud, compartir recursos físicos, imprimir, etc. Generalmente se considera que un servidor tiene una posición reactiva en el sistema, ya que se encuentra inactivo hasta que recibe una petición de servicio. Cuando recibe la petición, la procesa y envía la respuesta al solicitante, para después quedar nuevamente inactivo en espera de una nueva petición. En un sistema, el modo de ejecución de un servidor es continuo, ya que se inicializa al arrancar el sistema y está en operación hasta que el sistema se apaga. Un servidor tiene dos modos de arranque:

- Estático.
- Dinámico.

Un arranque estático sucede cuando el servidor se arranca como parte del arranque general del sistema donde se encuentra localizado. En contraste, un arranque dinámico sucede cuando un servidor es activado por un proceso del cliente que solicite sus servicios.

Los servidores forman parte importante del middleware básico de un sistema distribuido. Entre las características fundamentales de los servidores destaca que pueden ser reusables y relocalizarse. Los servidores pueden ser usados en diferentes tareas, entre las que se pueden destacar las siguientes:

- Servidores de datos.
- Servidores de archivos.

- Servidores de impresión.
- Servidores de correo.
- Servidores de programas.
- Servidores de bases de datos.
- Servidores de fecha y hora.
- Servidores de multimedia.
- Servidores de transacciones.
- Servidores web.

4.5 COMUNICACIÓN ENTRE PROCESOS

La comunicación entre procesos es un factor clave para construir sistemas distribuidos, los paradigmas de comunicación más usados en sistemas distribuidos son:

- Cliente - servidor.
- Llamada a un procedimiento remoto (RPC).
- Comunicación en grupo.

Los conceptos fundamentales que deben ser considerados para la comunicación son:

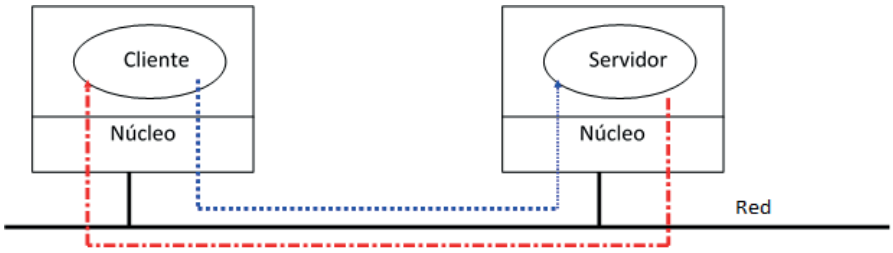
- Los datos tienen que ser aplanados antes de ser enviados.
- Los datos tienen que ser representados de la misma manera en la fuente y destino.
- Los datos tienen que empaquetarse para ser enviados.
- Usar operaciones de *send* para enviar y *receive* para recibir.
- Especificar la comunicación, ya sea en modo bloqueante o no bloqueante.
- Abstracción del mecanismo de paso de mensaje.
- La confiabilidad de la comunicación. Por ejemplo, usar TCP en lugar de UDP.

4.5.1 Modelo cliente - servidor (C-S)

El término *cliente - servidor* (C-S) hace referencia a la comunicación en la que participan dos aplicaciones. Es decir que está basado en la comunicación de uno a uno. La aplicación que inicia la comunicación enviando una

petición y esperando una respuesta se llama cliente. Mientras los servidores esperan pasivos, aceptan peticiones recibidas a través de la red, realizan el trabajo y regresan el resultado o un código de error porque no se generó la petición. Una máquina puede ejecutar un proceso o varios procesos cliente. La transferencia de mensaje en el modelo C-S se ejecuta en el núcleo. Una operación general del funcionamiento del *cliente - servidor* se muestra en la figura 4.2, donde un servidor espera una petición sobre un puerto bien conocido que ha sido reservado para cierto servicio. Un cliente reserva un puerto arbitrario y no usado para poder comunicarse.

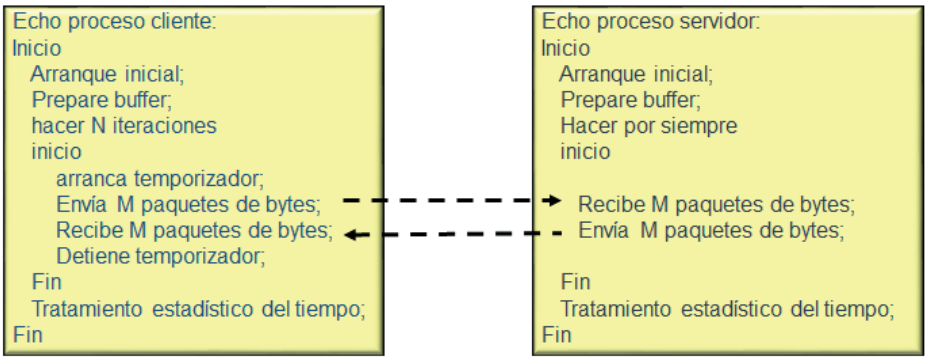
Figura 4.2. Modelo cliente - servidor



Características del software cliente - servidor

El software cliente es un programa de aplicación que se vuelve cliente temporal cuando se necesita acceso remoto. Este programa llama directamente al usuario y se ejecuta sólo durante una sesión. Se ejecuta localmente en la computadora del usuario y se encarga de iniciar el contacto con el servidor. Este programa puede acceder a varios servicios y no necesita hardware especial ni un complejo sistema operativo. Por su parte, el software servidor es un programa especial de propósitos dedicados a ofrecer un servicio pero que puede manejar varios clientes remotos al mismo tiempo. Este programa se inicia automáticamente al arranque del sistema y continuará ejecutándose en varias sesiones. El servidor espera pasivamente el contacto de los clientes remotos. Por lo general, el software servidor necesita ser instalado en un sistema operativo más robusto y contar con mucha capacidad de recursos de hardware. En la figura 4.3 se muestra un seudocódigo para un eco (echo) para un *cliente - servidor* (C-S).

Figura 4.3. Seudocódigo para un eco entre cliente
y servidor [Lin, Hsieh, Du, Thomas & McDonald, 1995]



La comunicación *cliente - servidor* puede ser implementada a través de librerías PVM (Parallel Virtual Machine) o sockets que permiten implementar comunicaciones bloqueantes y no bloqueantes, además de que también pueden usar TCP para tener comunicaciones confiables.

Deficiencias del modelo cliente - servidor

- El paradigma de comunicación es la entrada/salida (E/S), ya que estos no son fundamentales en el cómputo centralizado.
- No permite la transparencia requerida para un ambiente distribuido.
- El programador debe de atender la transferencia de mensajes o las E/S, tanto del lado del cliente como del lado del servidor.

4.5.2 Llamada de procedimiento remoto (RPC)

La llamada de procedimiento remoto, mejor conocido como RPC, es una variante del paradigma *cliente - servidor* y es una vía para implementar en la realidad este paradigma. En el RPC, un programa llama a un procedimiento localizado en otra máquina. El programador no se preocupa por las transferencias de mensajes o de las E/S. La idea de RPC es que una llamada a un procedimiento remoto se parezca lo más posible a una llamada local, esto le permite una mayor transparencia. Para obtener dicha transparencia, el RPC usa un *resguardo de cliente*, que se encarga de empacar los parámetros en un mensaje y le solicita al núcleo que envíe el mensaje al servidor, posteriormente se bloquea hasta que regrese la respuesta.

Algunos puntos problemáticos del RPC son:

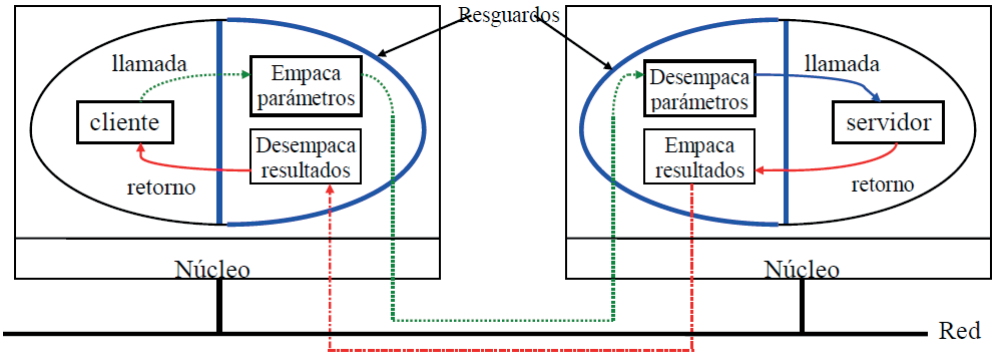
- Que se deben de usar espacios de direcciones distintos, debido a que se ejecutan en computadoras diferentes.
- Como las computadoras pueden no ser idénticas, la transferencia de parámetros y resultados puede complicarse.
- Ambas computadoras pueden descomponerse.

Modo de operación del RPC

En la figura 4.4 se muestra una llamada a un procedimiento remoto, el cual se realiza considerando los siguientes pasos:

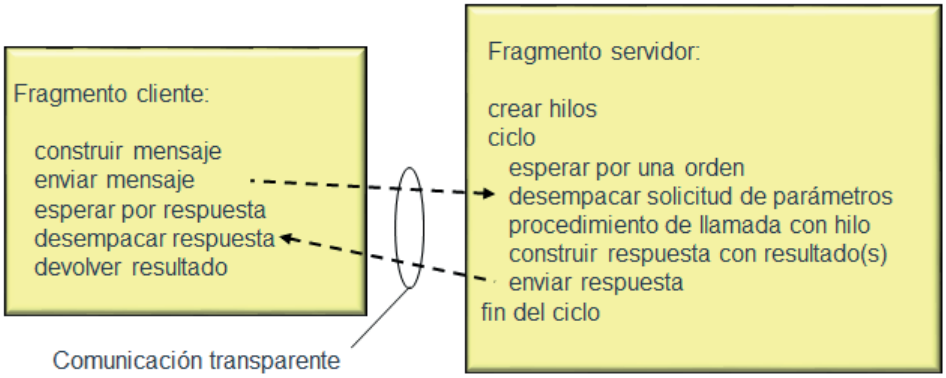
1. El procedimiento cliente llama al resguardo del cliente de la manera usual.
2. El resguardo del cliente construye un mensaje y realiza un señalamiento al núcleo.
3. El núcleo envía el mensaje al núcleo remoto.
4. El núcleo remoto proporciona el mensaje al resguardo del servidor.
5. El resguardo del servidor desempaca los parámetros y llama al servidor.
6. El servidor realiza el trabajo y regresa el resultado al resguardo.
7. El resguardo del servidor empaca el resultado en un mensaje y hace un señalamiento mediante el núcleo.
8. El núcleo remoto envía el mensaje al núcleo del cliente.
9. El núcleo del cliente da el mensaje al resguardo del cliente.
10. El resguardo desempaca el resultado y lo entrega al cliente.

Figura 4.4. Modo de operación de un RPC entre un proceso cliente y uno servidor



En el seudocódigo de la figura 4.5 se indica un ejemplo de fragmentos de un RPC.

Figura 4.5. Fragmentos de cliente - servidor en RPC

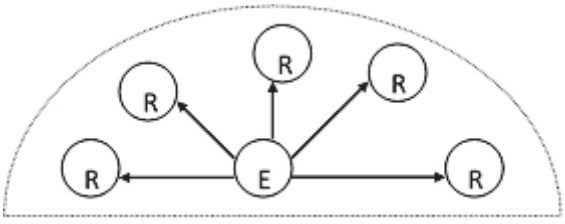


4.5.3 Comunicación en grupo

En un sistema distribuido puede haber comunicación entre procesos de uno a muchos o de uno a muchos, como se indica en la figura 4.6. Los grupos son dinámicos, lo cual implica que se pueden crear nuevos grupos y destruir grupos anteriores. Las técnicas para implantar la comunicación en grupos pueden ser:

- Transmisión de multidifusión (multicast)
- Transmisión completa
- Transmisión punto a punto

Figura 4.6. La comunicación en grupo se da entre un emisor (E) y varios receptores (R)



Con respecto a los aspectos de diseño de los grupos, se tienen las siguientes opciones [Tanenbaum, 1996]:

- *Grupos cerrados*: Por ejemplo, el procesamiento paralelo.
- *Grupos abiertos*: Por ejemplo, el soporte de servidores redundantes.
- *Grupo de compañeros*: Permite que las decisiones se tomen de manera colectiva.
  - Ventaja: el grupo de compañeros es simétrico y no tiene punto de falla.
  - Desventaja: la toma de decisiones por votación es difícil, crea retraso y es costosa.
- *Grupo jerárquico*: Existe un coordinador y varios trabajadores.
- *Membresía de grupo*:
  - Permite crear, eliminar grupos, agregar o eliminar procesos de grupos.
  - Utiliza técnicas como la del servidor del grupo o membresía distribuida.
  - Existen problemas de detección cuando un miembro ha fallado.
- *Direccionamiento al grupo*: Los grupos deben de poder direccionarse. Estas pueden ser:
  - Dirección única grupal
  - Dirección de cada miembro del grupo
  - Direccionamiento de predicados
- *Primitivas*: Las primitivas de comunicación son:
  - *send* y *receive*, de la misma forma que en una comunicación puntual
  - *group_send* y *grop_receive* para comunicación en grupo
- *Atomicidad*: Se refiere a que cuando se envía un mensaje a un grupo, este debe de llegarles a todos los miembros o a ninguno, así como garantizar la consistencia.
- *Ordenamiento de mensajes*: Está conjuntada con la atomicidad y permite que la comunicación en grupo sea fácil de comprender y utilizar. Los criterios son:
  - Ordenamiento con respecto al tiempo global
  - Ordenamiento consistente
  - Ordenamiento con respecto a grupos traslapados
- *Escalabilidad:* Permite que el grupo continúe funcionando, aun cuando se agreguen nuevos miembros.

4.6 Interfaz de programación de aplicaciones (API)

Las aplicaciones clientes - servidor se valen de protocolos de transporte para comunicarse. Las aplicaciones deben de especificar los detalles sobre el tipo de servicio (cliente o servidor), los datos a transmitir y el receptor donde situar los datos. La interfaz entre un programa de aplicación y los protocolos

de comunicación de un sistema operativo se llama interfaz de programación de aplicaciones (API).

Un API define un conjunto de procedimientos que puede efectuar una aplicación al interactuar con el protocolo. Por lo general, un API tiene procedimientos para cada operación básica. Los programadores pueden escoger una gran variedad de APIs para sistemas distribuidos. Algunas de estas son BSD socket, Sun's RPC/XDR, librería de paso de mensajes PVM y Windows Sockets. La API de *socket*, que tuvo sus orígenes en el UNIX BSD, se convirtió en una norma de facto en la industria y muchos sistemas operativos la han adoptado, tanto para computadoras basadas en Widows (winsock) como para algunos sistemas UNIX.

4.6.1 La interfaz de socket

Un socket es un punto de referencia hacia donde los mensajes pueden ser enviados, o de donde pueden ser recibidos. Al llamar la aplicación a un procedimiento de socket, el control pasa a una rutina de la biblioteca de sockets que realiza las llamadas al sistema operativo para implementar la función de socket. UNIX BSD y los sistemas derivados contienen una biblioteca de sockets, la cual puede ofrecer a las aplicaciones una API de socket en un sistema de cómputo que no ofrece sockets originales.

Los sockets emplean varios conceptos de otras partes del sistema, pero en particular están integrados con la E/S, por lo que una aplicación se comunica con un socket de la misma forma en que se transfiere un archivo. Cuando una aplicación crea un socket, recibe un descriptor para hacer referencia a este. Si un sistema usa el mismo espacio de descriptor para los sockets y otras E/S, es posible emplear la misma aplicación para transferir datos localmente como para su uso en red. La ventaja del método de socket es que la mayor parte de las funciones tienen tres o menos parámetros, sin embargo, se debe llamar a varias funciones a este método.

4.6.2 Funciones de la API de sockets

Las funciones relacionadas a la API sockets que permiten establecer una comunicación entre dos computadoras son:

socket( )

Esta rutina se usa para crear un socket y regresa un descriptor correspondiente a este socket. Este descriptor es usado en el lado del cliente y en el lado del servidor de su aplicación. Desde el punto de vista de la aplicación, el descriptor de archivo es el final de un canal de comunicación. La rutina retorna -1 si ocurre un error.

close( )

Indica al sistema que el uso de un socket debe de ser finalizado. Si se usa un protocolo TCP (orientado a conexión), *close* termina la conexión antes de cerrarlo. Cuando el socket se cierra, se libera al descriptor, por lo que la aplicación ya no transmite datos y el protocolo de transportación ya no acepta mensajes de entradas para el socket.

bind( )

Suministra un número a una dirección local a asociar con el socket, ya que cuando un socket es creado no cuenta con dirección alguna.

listen( )

Esta rutina prepara un socket para aceptar conexiones y solo puede ser usada en sockets que utilizan un canal de comunicación virtual. Esta rutina se deberá usar del lado del servidor de la aplicación antes de que se pueda aceptar alguna solicitud de conexión del lado del cliente. El servidor encola las solicitudes de los clientes conforme estas llegan. La cola de solicitudes permite que el sistema detenga las solicitudes nuevas mientras que el servidor se encarga de las actuales.

accept( )

Esta rutina es usada del lado del servidor de la aplicación para permitir aceptar las conexiones de los programas cliente. Después de configurar una cola de datos, el servidor llama *accept*, cesa su actividad y espera una solicitud de conexión de un programa cliente. Esta rutina solo es válida en proveedores de transporte de circuito virtual. Cuando llega una solicitud al socket especificado *accept( )* llena la estructura de la dirección, con la dirección del cliente que solicita la conexión y establece la longitud de la dirección, *accept( )* crea un socket nuevo para la conexión y regresa su descriptor al que lo invoca, este nuevo socket es usado por el servidor para comunicarse con el cliente y al terminar se cierra. El socket original es usado por el servidor para aceptar la siguiente conexión del cliente.

connect( )
Esta rutina permite establecer una conexión a otro socket. Se utiliza del lado del cliente de la aplicación permitiendo que un protocolo TCP inicie una conexión en la capa de transporte para el servidor especificado. Cuando se utiliza para protocolos sin conexión, esta rutina registra la dirección del servidor en el socket, esto permite que el cliente transmita varios mensajes al mismo servidor. Usualmente el lado cliente de la aplicación enlaza a una dirección antes de usar esta rutina, sin embargo, esto no es requerido.

send( )
Esta rutina es utilizada para enviar datos sobre un canal de comunicación tanto del lado del cliente como del lado servidor de la aplicación. Se usa para sockets orientados a conexión, sin embargo, podría utilizarse para datagramas pero haciendo uso de *connect( )* para establecer la dirección del socket.

sendto( )
Permite que el cliente o servidor transmita mensajes usando un socket sin conexión (usando datagramas). Es exactamente similar a *send( )* solo que se deberán especificar la dirección destino del socket al cual se quiere enviar el dato. Se puede usar en sockets orientados a conexión pero el sistema ignorará la dirección destino indicada en *sendto( )*.

recv( )
Esta rutina lee datos desde un socket conectado y es usado tanto en el lado del cliente como del lado del servidor de la aplicación.

recvfrom( )
Esta rutina lee datos desde un socket sin conexión. En este caso, el sistema regresa la dirección del transmisor con los mensajes de entrada y permite registrar la dirección del socket transmisor en la misma forma que espera *sendto( )*, por lo que la aplicación usa la dirección registrada como destino de la respuesta.

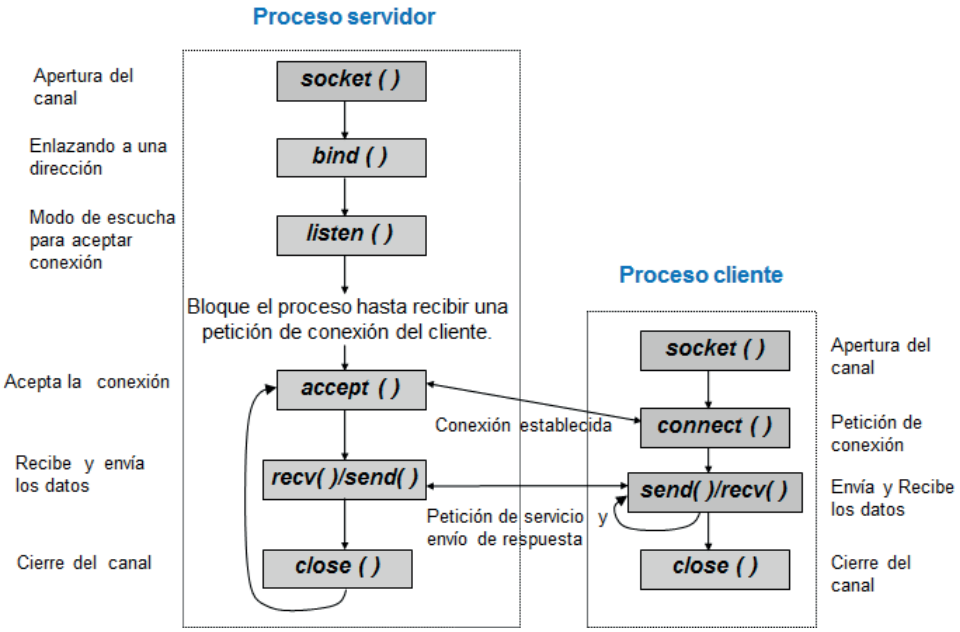
4.6.3 Ejemplo de cliente servidor usando socket

4.6.3.1 Comunicación orientada a conexión (TCP)

Un servicio orientado a conexión requiere que dos aplicaciones establezcan una conexión de transportación antes de comenzar el envío de datos. Para establecer la comunicación, ambas aplicaciones primero interactúan local-

mente con protocolo de transporte y después estos protocolos intercambian mensajes por la red. Una vez que ambos extremos están de acuerdo y se haya establecido la conexión, las aplicaciones podrán enviar datos. La secuencia de llamadas para un escenario orientado a conexión se indica en la figura 4.7.

Figura 4.7. Uso del API socket para una comunicación orientada a conexión [Jamsa & Cope, 1996]

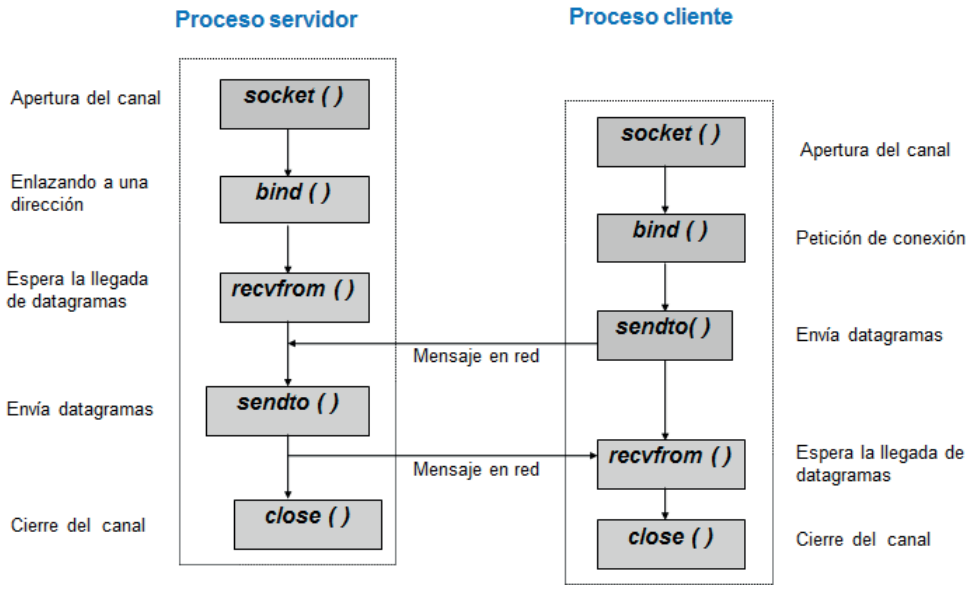


En la figura 4.7 se observa que, después de llamar a *listen( )*, el servidor se pone en modo pasivo hasta que recibe una solicitud del cliente. Cuando la solicitud de servicio del cliente llega al socket monitoreado por la función *accept( )*, se crea de modo automático un socket nuevo, que será conectado inmediatamente con el proceso cliente. Este socket se llama de servicio y se cierra al terminar la transferencia de datos. El socket original que recibió la solicitud de servicio permanece abierto en estado de escucha para seguir aceptando nuevas solicitudes.

4.6.3.2 Comunicación sin conexión o servicio de datagrama

El servicio de datagrama (ver figura 4.8) [Jamsa & Cope, 1996] es similar a enviar una carta postal. El lado cliente actúa como la persona que envía la carta y el lado servidor como la persona que lo recibe. El servidor usa las llamadas a *socket( )* y *bind( )* para crear y unir un socket. Como el socket es sin conexión, se deberán usar las llamadas a *recvfrom( )* y *sendto( )*. Se llama a la función *bind( )* pero no a *connect( )*, dejando esta última como opcional. La dirección destino se especifica en la función *sendto( )*. La función *recvfrom( )* no espera conexión sino que responde a cualquier dato que llegue al puerto que tiene enlazado.

Figura 4.8. Comunicación para aplicaciones que usan datagramas [Jamsa & Cope, 1996]



EJERCICIOS

1. Explica las tres cosas que debe garantizar un MUTEX.

2. Dadas las variables X y Z, con hilo H1 fijando ambas variables a 0 mientras hilo H2 fija a ambas variables en 1, escribe el seudocódigo de un MUTEX que garantice la consistencia de ambas variables.

3. Multicast es una tecnología para comunicación en grupo, explica su funcionamiento.

4. Cita al menos tres funciones relacionadas con los hilos y el uso de cada una de ellas.

5. Cita al menos tres desventajas del modelo cliente - servidor.

6. Explica la diferencia entre el RPC y el modelo cliente - servidor.

7. Explica la diferencia entre envío de datagrama y flujo de datos.

8. ¿Cuándo es importante usar una comunicación orientada a conexión?

9. ¿Cuándo es importante usar una comunicación sin conexión?

10. ¿Cuál es la diferencia entre un servidor de transacciones y un servidor de archivos?

11. ¿Cuál es la diferencia entre una función send( ) y sendto( )?

Actividad integradora

1. De la aplicación de los contenidos, así como de las habilidades y actitudes desarrolladas.
   Se recomienda exposiciones por parte del profesor, así como prácticas del laboratorio que permitan al estudiante entender el uso y programación de los hilos en un entorno distribuido. También se pueden realizar prácticas de programación de sockets en caso de que los alumnos carezcan de estos conocimientos, así como ejemplos con RPC o Java RMI. Por la importancia del tema, se recomienda desarrollar el contenido en un promedio de seis sesiones que incluyan trabajo intenso en el laboratorio. El alumno debe de mostrar un gran interés por programar aplicaciones distribuidas que involucren conceptos de hilos, objetos y diferentes niveles de transparencia.

2. Del logro de los objetivos establecidos en el capítulo o apartado del material.
   Se espera alcanzar los objetivos de este capítulo con el apoyo en la solución de los ejercicios, así como en la programación de diferentes modelos de comunicación que involucren algún nivel básico de transparencia.

Capítulo 5. Sincronización

Objetivo: Que el alumno analice y comprenda los diferentes algoritmos relacionados con sincronización, exclusión mutua, elección y consenso distribuidos, así como la importancia de estos en el diseño de sistemas distribuidos.

5.1 Introducción

La sincronización en sistemas distribuidos es más complicada que en un sistema centralizado, ya que se debe de considerar algunos de los siguientes puntos:

- Que la información se distribuye en varias máquinas.
- Los procesos toman decisiones con base en información local.
- Se debe evitar un punto único de falla.
- No existe un reloj común, como tampoco otra fuente de tiempo global.

5.2 Sincronización de relojes

Para la sincronización de relojes existen las siguientes alternativas:

Relojes lógicos

- Según Lamport, la sincronización de relojes no debe ser absoluta, debido a que si dos procesos no interactúan entre sí, no requieren que sus relojes estén sincronizados.
- La distorsión de reloj es la diferencia entre los valores de tiempo de los diferentes relojes locales.
- Aquí importa el orden de ocurrencia de los eventos, no la hora exacta.

Relojes físicos

- Usan el tiempo atómico internacional (TAI) y el tiempo coordenado universal (UTC).

- Se pueden sincronizar por medio de radios de onda corta.
- También se puede usar satélite para sincronizar.

5.3 ALGORITMOS PARA LA SINCRONIZACIÓN DE RELOJES

5.3.1 El Algoritmo de Lamport

Lamport define una relación temporal llamada: *ocurre antes de*. Por ejemplo, (a → b) se interpreta como "a ocurre antes de b". Para ilustrar esto, consideremos la siguiente situación:

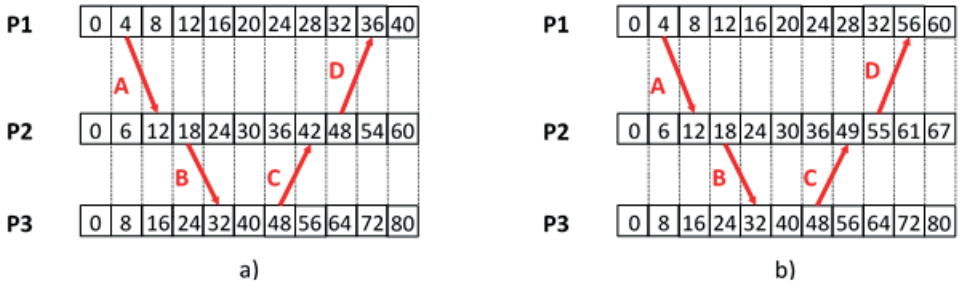
1. Si *a* y *b* son eventos del mismo proceso y *a* ocurre antes que *b*, entonces $a \rightarrow b$ es verdadero.
2. Si *a* es un envío de un mensaje por un proceso y *b* es la recepción del mensaje por otro proceso, entonces $a \rightarrow b$ es verdadero.
3. Eventos concurrentes: si $a \rightarrow b$ no es verdadero ni $b \rightarrow a$ tampoco.

Se requieren valores de tiempo para medir el tiempo:

- Si $a \rightarrow b$, entonces $C(a) < C(b)$.
- C siempre es creciente (se pueden hacer correcciones hacia adelante pero no hacia atrás).
- Para todos los eventos *a* y *b*, tenemos que $C(a) \neq C(b)$.

Un ejemplo de tres procesos (P1, P2 y P3), donde cada uno tiene su propio reloj a diferentes velocidades se muestra en la figura 5.1a, mientras que en la figura 5.1b se muestra la corrección de los relojes usando el algoritmo de Lamport.

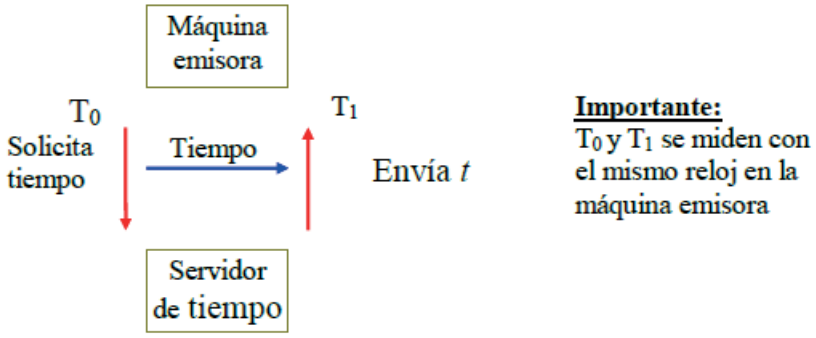
Figura 5.1. Un ejemplo del algoritmo de Lamport en tres procesos



5.3.2 El algoritmo de Christian

En este algoritmo un servidor central provee el tiempo por petición del usuario, este servidor es conocido como servidor del tiempo (ver figura 5.2).

Figura 5.2. Actualizando la hora a través de un servidor de tiempo



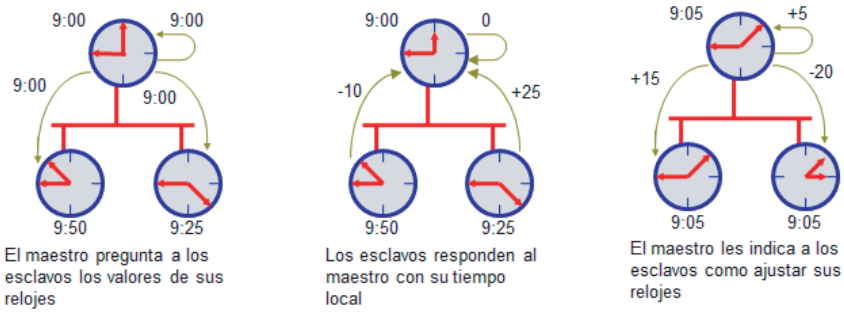
La operación de este algoritmo se resume a continuación:

- Periódicamente cada máquina envía un mensaje para solicitar el tiempo actual a este servidor.
- El servidor de tiempos responde con un mensaje que contiene el tiempo actual *t*.
- La máquina emisora puede poner su tiempo en $t + T_{trans}$.
- Un problema grave es que, como el tiempo nunca corre hacia atrás, el valor actual del tiempo del emisor podría ser mayor al valor de $t+T_{trans}$, lo que perjudicaría a los archivos compilados.
- Otro problema es el tiempo de propagación (T_{trans}), el cuál es distinto de 0 y varía según la carga en la red.
- Una estimación del tiempo de propagación sería realizado por el reloj de la máquina emisora calculando $(T_1 - T_0)/2$ e incrementándolo al llegar el valor *t*.
- Si se conoce el tiempo que tarda el servidor en manejar la interrupción (t_l), la estimación de tiempo puede mejorar con base en la siguiente expresión: $(T_1 - T_0 - t_l) / 2$.
- Christian sugiere hacer varias mediciones para mejorar la precisión y descartar los valores límites de (T_1-T_0), ya que estos están en función de la operación de la red.

5.3.3 El algoritmo de Berkeley

En este algoritmo una computadora maestra pide periódicamente a las computadoras esclavas sus relojes y efectúa la sincronización (ver figura 5.3). Se calcula un tiempo promedio, el cual es tomado como base si este no se diferencia más de un valor base. La computadora maestra envía los desfases de tiempo a los esclavos para que avancen su reloj o disminuyan la velocidad del mismo hasta lograr una reducción específica.

Figura 5.3. Representación del algoritmo de Berkeley [Tanenbaum & Van Steen, 2008]



5.4 EXCLUSIÓN MUTUA

En los sistemas distribuidos existen situaciones en que hay recursos compartidos los cuales no pueden ser utilizados por más de un proceso al mismo tiempo. En sistemas que comparten memoria se pueden utilizar semáforos o monitores para garantizar el uso de la región crítica de manera exclusiva. Sin embargo, en los sistemas distribuidos, los procesos ya no comparten la memoria física, por ello se deben de idear otros algoritmos que garanticen la exclusión mutua. Los requisitos básicos que debe cumplir un algoritmo de exclusión mutua son:

- Inanición.
- Seguridad.
- Orden.

La inanición cuida que un proceso que desea entrar en una región crítica pueda entrar en algún tiempo, lo cual implica que no se deben producir interbloqueos ni inanición. La seguridad se refiere a que la exclusión mutua debe de garantizar que en la región crítica, cuando mucho, solo se ejecute

un proceso en un momento dado. El orden implica que el acceso a la región crítica debe de realizarse siguiendo el orden casual propuesto por Lamport "sucedió antes". Para garantizar que ningún proceso entre en una región crítica mientras esta sea ocupada por otro proceso en sistema distribuido, se usan los siguientes algoritmos:

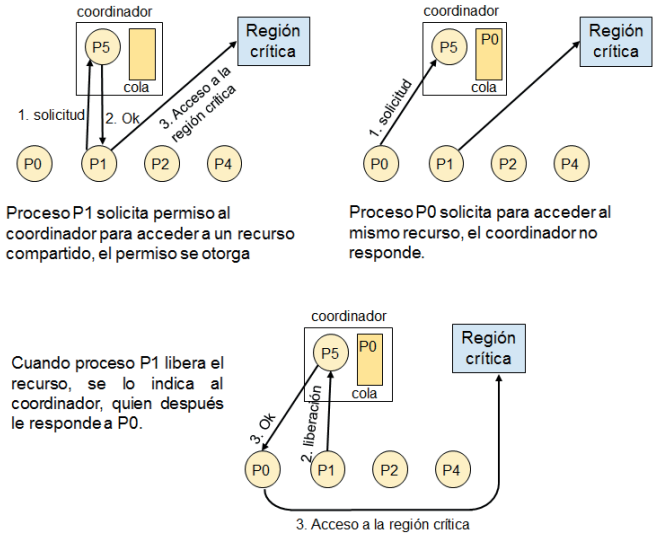
- Algoritmo centralizado.
- Algoritmo distribuido (Ricart y Agrawala).
- Algoritmo de anillo de token.

5.4.1 Algoritmo de servidor centralizado

Este algoritmo simula un solo procesador para realizar la exclusión. El algoritmo nombra a un proceso coordinador. En este algoritmo el coordinador da acceso a la región crítica al que posea un token (lo devuelve al coordinador). Cuando un proceso sale de la región crítica libera el token. El coordinador permite la entrada enviando el token o no (para lo cual no envía algo o puede enviar un permiso denegado).

Los problemas de este algoritmo se presentan cuando el coordinador falla o cuando falla el poseedor del token. Un solo coordinador en un gran sistema puede ocasionar un cuello de botella. La figura 5.4 muestra un ejemplo de cómo opera este algoritmo.

Figura 5.4. Algoritmo de servidor centralizado [Tanenbaum & Van Steen, 2008]

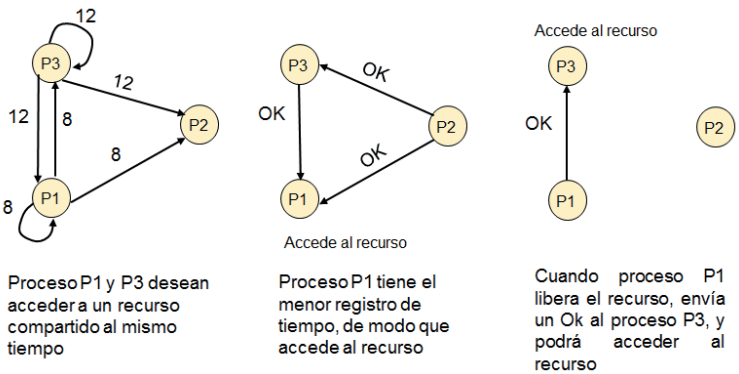


5.4.2 Algoritmo de Ricart y Agrawala

Este algoritmo está basado en un consenso distribuido y requiere de un orden para todos los eventos en el sistema [Tanenbaum & Van Steen, 2008]. Cuando un proceso desea entrar a la región crítica, construye un mensaje formado por su nombre, su número de proceso y la hora actual. Después, este mensaje es mandado a todos los procesos, incluyéndose el mismo. Así, se considera que la comunicación es confiable. La figura 5.5 muestra un escenario de este algoritmo.

- Cuando un proceso recibe el mensaje se pueden presentar tres alternativas:
  a) Si el proceso receptor no está en la región crítica y tampoco desea entrar, envía un mensaje de OK al proceso emisor.
  b) Si el receptor se encuentra en la región crítica, no responderá y solo colocará la solicitud en una cola.
  c) Si el receptor no está en la región crítica pero desea entrar, compara la etiqueta de tiempo del mensaje recibido con la suya, la menor gana. Si el mensaje recibido es menor al suyo, ejecuta el paso a), pero si es mayor al suyo ejecuta el paso b).
- El proceso solicitante espera el permiso de los otros, entonces entra a la región crítica (recurso compartido), si no ocurre así se bloquea.
- Cuando sale de la región crítica envía un OK y saca a todos los procesos en su cola.
- Los problemas que presenta este algoritmo son:
  - Que requiere mucho más mensajes que el algoritmo de servidor centralizado.
  - Que no existe la tolerancia a fallas para ningún proceso.

Figura 5.5. Ejemplo de Algoritmo de Ricart y Agrawala



5.4.3 Algoritmo de anillo de token

Este algoritmo distribuido fue propuesto por Lann [1977] para alcanzar exclusión mutua en un sistema distribuido. En este algoritmo los procesos pueden estar desordenados (ver figura 5.6) y opera como sigue:

Premisas:

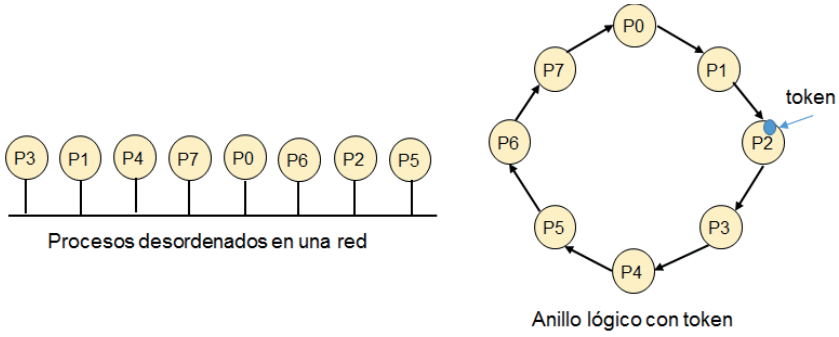
- Todos los equipos conectados forman un anillo lógico.
- Cada proceso es un nodo del anillo y se le asigna una posición en el anillo.
- Cada nodo del anillo debe de saber cuál es la dirección de su vecino.

Operación:

- Al arrancar el sistema, al proceso de posición 1 se le da un token, el cual irá circulando por el anillo.
- Cuando el proceso *k* tenga el token, debe transferirlo mediante un mensaje al proceso *k+1*.
- Así, el token irá pasando por todos los nodos del anillo.
- Cuando un proceso recibe el token, si quiere entrar a la región crítica, retiene el token y entra en la región crítica.
- Cuando el proceso sale de la región crítica le pasa el token al siguiente nodo del anillo.

Este algoritmo no respeta el orden y puede requerir de 1 mensaje (en el mejor de los casos) a *n-1* mensajes (en el peor de los casos) para conseguir entrar a la región crítica. En caso de perderse un mensaje con el token sería difícil detectarlo, porque se puede suponer que algún proceso está usando el token. Si falla cualquier proceso del anillo estando en la región crítica también es un problema en este algoritmo, ya que es difícil de detectarlo.

Figura 5.6. Algoritmo de anillo de token [Tanenbaum & Van Steen, 2008]



5.5 ALGORITMOS DE ELECCIÓN

Una elección es un procedimiento que se lleva a cabo por los procesos de cierto grupo para escoger a alguno para una tarea específica (ejemplo: coordinación). El objetivo del algoritmo de elección es asegurar que cuando se comienza una tarea, se concluya que todos los procesos están de acuerdo en quién es el nuevo coordinador. Un algoritmo de elección para realizar este procedimiento emplea tres tipos de mensaje:

- Anunciación del evento de elección.
- Votación.
- Anunciación del resultado de la elección

5.5.1 El algoritmo del Grandulón

El algoritmo del "Grandulón" es un algoritmo para elegir un proceso coordinador y fue propuesto por García Molina, en 1982 [Coulouris *et al.*, 2012], [Tanenbaum, 1996]. En general, el método a seguir es el siguiente:

Premisas:
- El sistema es síncrono y utiliza tiempo de espera para la identificación de fallas en los procesos.
- Se permite que los procesos se bloqueen durante la ejecución del algoritmo.
- La entrega de mensajes entre procesos se supone fiable y dentro de un periodo máximo.
- Los procesos están ordenados, tienen un único identificador (IDs) conocido y se sabe cuántos procesos existen.

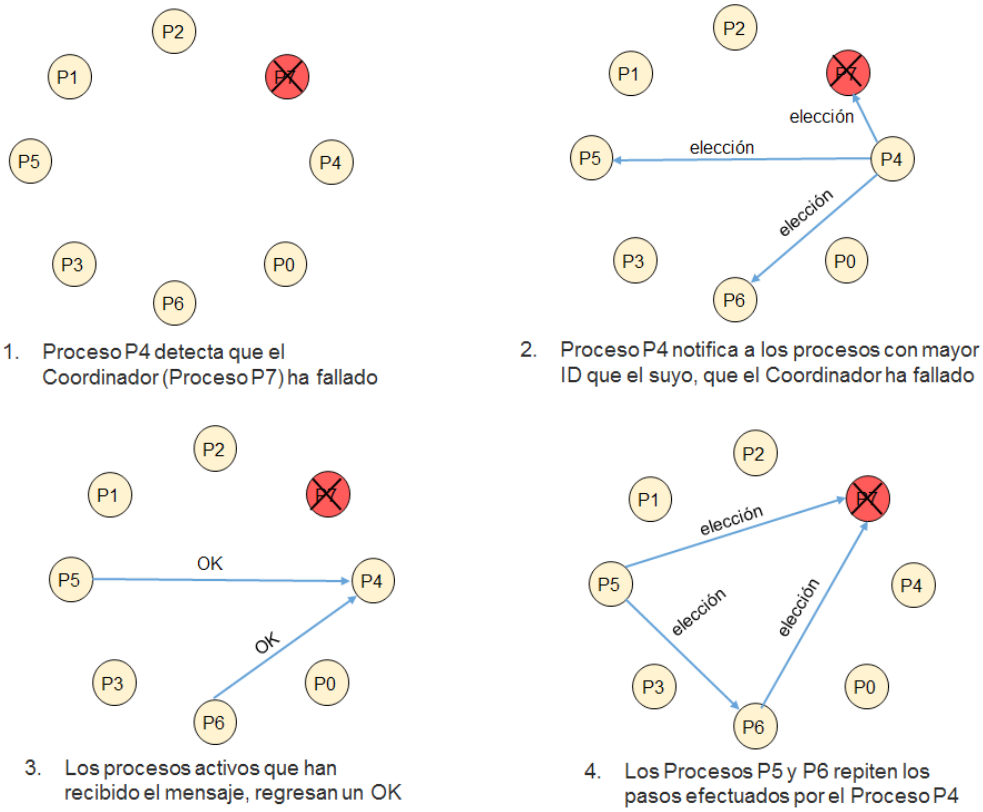
Tipo de mensajes:
- Mensaje de Elección: comunica que se seleccionará un nuevo coordinador.
- Mensaje de Respuesta: es una respuesta al mensaje de elección.
- Mensaje de Coordinador: comunica el ID del proceso seleccionado como coordinador.

Operación:
1. Un proceso x manda un mensaje de elección a todos aquellos procesos que tengan un identificador más grande cuando detecta que el coordinador ha fallado.

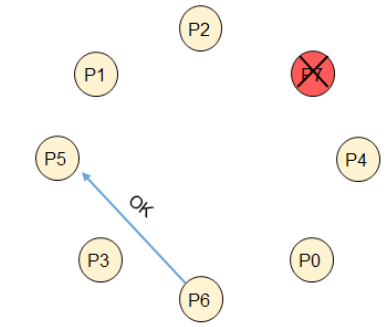
2. El proceso x espera los votos y, si ningún voto (ok) llega después de un cierto tiempo, el proceso x se declara coordinador, y envía el mensaje de coordinador a los procesos con identificador más pequeño que el suyo.
3. Si un voto llega (aunque pueden llegar varios votos), puede ser que otro coordinador sea declarado ganador.
4. Si un proceso recibe un mensaje de elección, envía un voto y otra elección empieza (paso 4 de la figura 5.7).
5. Cuando un coordinador que había fallado regresa, empieza una elección y puede volver a readquirir el control, aunque exista un coordinador actual.

Un ejemplo de cómo trabaja el algoritmo del "Grandulón" se muestra en la figura 5.5.

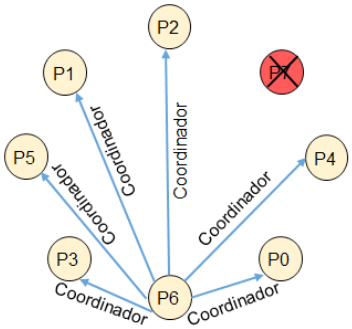
Figura 5.7. Ejemplo del algoritmo del "Grandulón" para elegir a un nuevo coordinador



1. Proceso P4 detecta que el Coordinador (Proceso P7) ha fallado
2. Proceso P4 notifica a los procesos con mayor ID que el suyo, que el Coordinador ha fallado
3. Los procesos activos que han recibido el mensaje, regresan un OK
4. Los Procesos P5 y P6 repiten los pasos efectuados por el Proceso P4



5. El Proceso P6 envía OK al proceso P5 y detecta que es el proceso con mayor ID



6. Proceso P6 tiene el mayor ID, se proclama Coordinador y lo comunica a los otros

5.5.2 Algoritmo de anillo

Otros algoritmos de elección se basan en topologías de anillo, ya sea lógico o físico. Existen varios algoritmos de elección basados en anillo. Sin embargo, aquí se describe el algoritmo de anillo de Chang & Roberts [1974] basado en el principio de extinción selectiva. Este algoritmo se usa cuando:

- Los procesos están física o lógicamente ordenados en anillo.
- No se conoce el número total de procesos (n).
- Cada proceso se comunica con su vecino (izquierda o derecha).

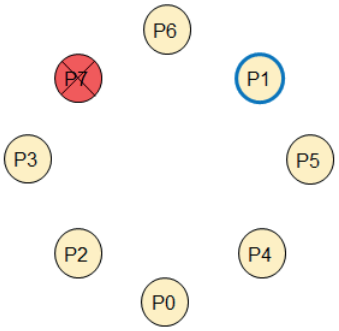
Operación (la figura 5.8 ilustra la operación de este algoritmo):

1. Inicialmente todos los procesos son "no participantes".
2. Cualquier proceso *P* decide arrancar una elección en cualquier momento.
3. Este proceso *P* se pone en estado "participante" y envía un mensaje de elección *M* a su vecino.
4. El mensaje *M* enviado contiene el *ID* (identificador) del proceso que ha iniciado la elección.
5. Cuando el vecino recibe el mensaje de elección *M*, establece su estado como participante y comprueba el *ID* del mensaje.
6. Si es mayor que su propio *ID*, entonces se lo envía directamente a su vecino.
7. Si su *ID* es mayor al *ID* recibido, entonces lo coloca en el mensaje *M* y lo envía a su vecino.
8. Así se circula el mensaje *M* sucesivamente hasta que llega a un proceso P_n que comprueba que el *ID* recibido es el propio. Eso indica que solo ha sobrevivido el mayor *ID*, que es la del proceso P_n.

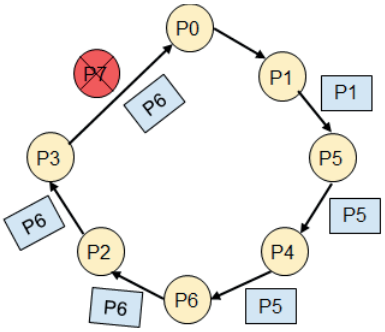
9. Entonces, este proceso P_n es el coordinador y lo notifica a su vecino. Cuando un proceso recibe un mensaje de coordinador debe de poner su estado como "no participante" y enviar el mensaje a su vecino.
10. Cuando el mensaje de coordinador retorna al proceso que lo emitió (el coordinador), entonces todos los procesos saben quién es el coordinador y todos quedan en estado "no participantes".

En caso de que dos procesos inicien al mismo tiempo una elección y se envíen mensajes de elección, un proceso en estado de "participante" debe de verificar el ID del proceso que envía el mensaje de elección. Si es menor al propio, el mensaje se descarta. Así, todos los mensajes de elección se extinguirán, excepto el que lleva el ID más alto.

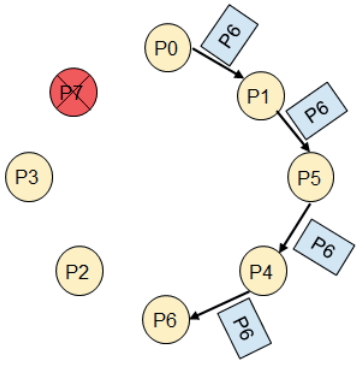
Figura 5.8. Algoritmo de elección que utiliza un anillo



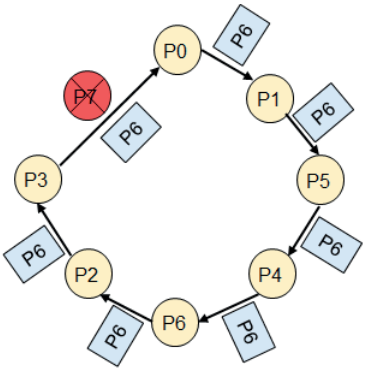
Todos los nodos en estado "no participante". Nodo P7 falla y lo detecta P1



Nodo 2 envía un "mensaje de elección". "Participantes": P1, P5, P4, P6, P2, P3 y P0.



La circulación del mensaje se detiene al llegar al nodo P6



P6 difunde "mensaje-coordinador. Todos los nodos en estado "no participante"

5.6 ALGORITMOS DE CONSENSO

La gran mayoría de los investigadores del área considera que el problema fundamental del cómputo distribuido es el consenso distribuido. Es decir, cómo lograr que un conjunto de procesos, ejecutándose en distintos nodos, se pongan de acuerdo con respecto al valor de un dato. Para que cualquier protocolo de coordinación funcione correctamente, se basan en que todos los procesadores correctos se pongan de acuerdo con respecto al valor del dato que intercambian. El problema del consenso consiste en conseguir que, aun en presencia de procesadores erróneos, los procesadores correctos sean capaces de ponerse de acuerdo con respecto a un valor, sin importar que este valor no sea el óptimo.

5.6.1 Problema de los generales bizantinos

El *problema de los generales bizantinos* fue planteado originalmente por Lamport, Shostak y Peace [1982]. Este problema plantea que un grupo de generales sitia una ciudad y deben de ponerse de acuerdo a través de un plan de ataque precisamente para atacar o retirarse, independientemente de que existan generales traidores. Los generales solo se comunican a través de mensajes a los otros generales. Uno de ellos, el general comandante, da las órdenes. Los otros, tenientes generales, deben de decidir si atacar o retirarse. Sin embargo, uno o más de los generales puede ser un traidor o pueden fallar. Esta traición puede verse de dos formas:

- Los mensajes pueden no llegar o, dicho de otra manera, las comunicaciones no son confiables.
- Un general traidor puede mentir, es decir, un nodo puede fallar de manera impredecible.

A continuación se revisa las circunstancias bajo las que se puede lograr un acuerdo entre los generales leales y cuando este acuerdo es imposible.

Caso 1: Tres generales

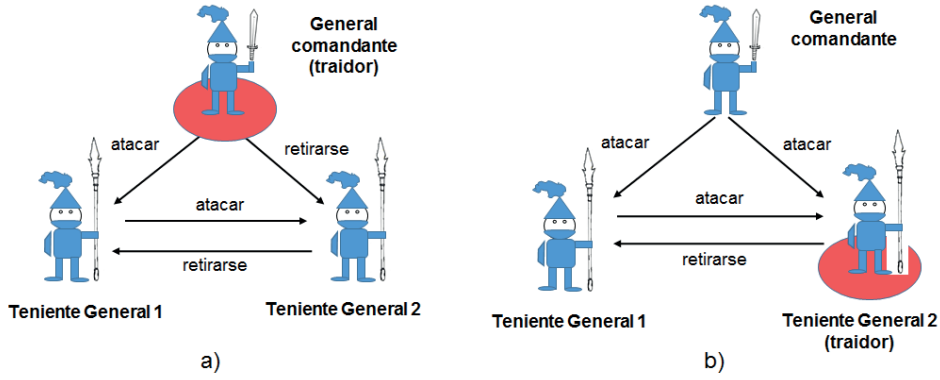
Este caso ilustra el escenario para cuando hay un general comandante y dos tenientes generales, y uno de ellos es traidor. ¿Pueden los generales leales llegar a un consenso?, es decir, ¿acordar "atacar" o "retirarse"?

Asume que el general comandante es el traidor (como se ilustra en la figura 5.9a). Entonces, el general comandante indicará al teniente general 1 "atacar" y al teniente general 2 "retirarse". Ambos tenientes tratarán de verificar la orden recibida comunicándose entre ellos. El teniente general 1 indicará al teniente general 2 que recibió la orden de "atacar", mientras que el teniente general 2 indicará al teniente general 1 que recibió la orden de "retirarse". Ambos tenientes generales podrán solo deducir que el general comandante es traidor pero no podrán tomar una decisión consensada.

Si se consideran los mismos participantes, ahora asume que el teniente general 2 es el traidor, el general comandante es leal, al igual que el teniente general 1. Este escenario se ilustra en la figura 5.9b, donde el general comandante da la orden de "atacar" a ambos tenientes generales, quienes intercambian la orden recibida. El teniente general 1 indica al teniente general 2 "atacar". Sin embargo, el teniente general 2 es traidor, por ello cambia la orden recibida y le indica al teniente general 1 "retirarse".

Para ambos escenarios, el teniente general escucha dos órdenes distintas, tanto del general comandante como del teniente general 2, sin saber cómo actuar, por lo que en ambos escenarios no existe consenso.

Figura.5.9. Caso de tres generales donde uno es traidor. a) El general comandante es el traidor, b) El teniente general es el traidor. No se llega a un consenso



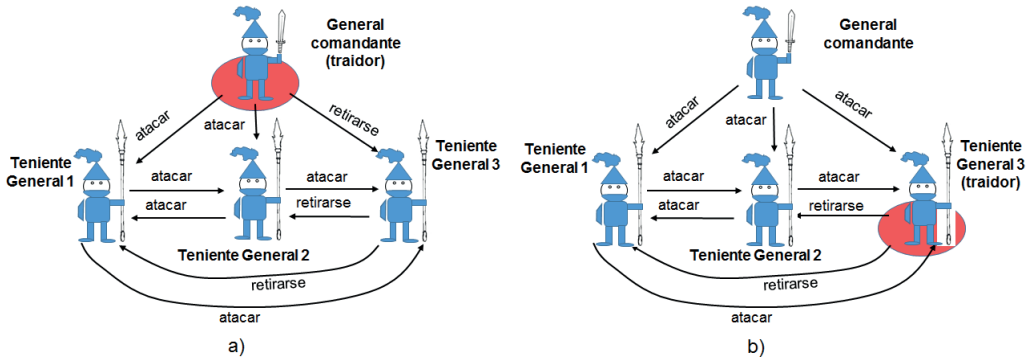
Caso 2: Cuatro generales

Ahora se muestra un caso similar al anterior pero con cuatro generales. Hay un general comandante y tres tenientes generales, y uno de ellos es traidor. ¿Pueden los generales leales llegar a un consenso?, es decir, ¿pueden acordar "atacar" o "retirarse"?

Asume que el general comandante es el traidor y los tres tenientes generales son leales. Independientemente de las órdenes que reciban los tenientes generales del general comandante, ellos las intercambian entre sí, de tal manera que los tres recibirán tres mensajes. Al decidir por la mayoría de las órdenes recibidas, los tres tenientes generales tomarán la misma decisión. En este caso, "atacar". Este escenario se muestra en la figura 5.10a.

Si se asume que el comandante general, y los tenientes generales 1 y 2 son leales, excepto el teniente general 3, quien es traidor, se tendría un escenario como el plateado en la figura 5.10b. El comandante general envía a todos los tenientes generales la orden de "atacar". Esta orden es retransmitida por los tenientes generales 1 y 2 a los demás, sin embargo, el teniente general, como es un traidor, cambia la orden recibida y envía a los tenientes generales 1 y 2 la orden de "retirarse". Esto no cambia la decisión consensada de los tenientes generales 1 y 2, ya que por mayoría deciden "atacar". También se puede notar que el general traidor pudo haber enviado cualquier mensaje sin que esto afectara el consenso del resultado.

Figura 5.10. Caso de cuatro generales donde uno es traidor.
a) El general comandante es el traidor, b) Un teniente general es el traidor.
En ambos casos, el consenso es "atacar"



El problema de los generales bizantinos es frecuentemente referido en la tolerancia a fallas de los sistemas distribuidos, también se conoce como problema de acuerdo bizantino. Lamport *et al.* [1982] demostraron que en un sistema con *k* procesos defectuosos, el consenso se puede lograr solo si están presentes *2k+1* procesos que funcionen correctamente, para un total de *3k+1* procesos.

Ejercicios

1. ¿Por qué es conveniente el uso de relojes lógicos en lugar de los relojes físicos?

2. Cita la diferencia entre el algoritmo de Christian y el algoritmo de Berkeley.

3. Dado tres procesos P1, P2 y P3, representa la planificación de los procesos siguiendo el algoritmo de "ocurre antes de".

4. ¿Cuál es la complejidad del algoritmo de anillo de Chang y Roberts?

5. ¿Cuáles son las desventajas del algoritmo de Ricart y Agrawala?

6. ¿Cuál es la principal limitante del algoritmo centralizado? Sugiere una variante.

7. En el algoritmo del Grandulón, ¿qué pasa si dos procesos detectan la caída del coordinador de forma simultánea y ambos deciden hacer una elección?

8. ¿Cómo se podría tolerar la falla del coordinador en un sistema centralizado?

9. ¿Cómo son afectadas las memorias cachés cuando se sincronizan los relojes internos de los procesadores de un sistema distribuido?

10. En el algoritmo de Ricart y Agrawala, ¿cómo se puede interpretar cuando un proceso falla y no responde a la solicitud de otro proceso para entrar a una región crítica?

Actividad integradora

1. De la aplicación de los contenidos, así como de las habilidades y actitudes desarrolladas.
   Se recomienda exposiciones por parte del profesor, así como una explicación detallada de cada algoritmo por medio de talleres prácticos donde los estudiantes demuestren diferentes casos y posibles escenarios de estos algoritmos. Se recomienda encargar como trabajo extraclase y en equipo el

desarrollar demos de algunos de estos algoritmos, además de exponer en clase para comprobar la apropiación del conocimiento de este capítulo. Debido a que este tema es fundamental en un curso de sistemas distribuidos, se recomienda desarrollar el contenido en un promedio de seis sesiones que incluyan trabajo intenso de ejemplos demostrativos de la manera en que operan estos algoritmos, así como para explicar su complejidad.

El alumno debe de mostrar un gran interés por abstraer y modelar problemas que pueden resultar cotidianos pero que tienen gran aplicabilidad en los sistemas de cómputo distribuido, también de demostrar habilidades para programar algún algoritmo de sincronización en algún lenguaje de programación de alto nivel. Se recomienda Java.

2. Del logro de los objetivos establecidos en el capítulo o apartado del material.

   Se espera alcanzar los objetivos de este capítulo con el apoyo en la solución de los ejercicios, así como en la programación de diferentes algoritmos de sincronización que simulen su operación.

Capítulo 6. Transacciones distribuidas

Objetivo: Que el alumno analice y comprenda las particularidades de las transacciones en un entorno distribuido, así como sus modelos y aplicaciones.

6.1 Introducción

En general, en los sistemas distribuidos se realizan dos acciones que se pueden considerar como contrastantes. La primera acción consiste en que los clientes no deben intervenir en las acciones que realiza otro cliente, mientras que la segunda acción consiste en que el servidor debe ser usado por los clientes para compartir e intercambiar información entre ellos.

Para el manejo de este tipo de situaciones en los datos compartidos existe una herramienta conceptual muy útil conocida como "transacción", la cual es una unidad indivisible de manipulación de información. Las transacciones en los sistemas distribuidos reciben el nombre de transacciones atómicas, siendo estas un mecanismo de alto nivel que oculta los aspectos técnicos de sincronización, como la exclusión mutua, interbloqueos y recuperación de fallas. Algunas características de las transacciones son:

- Que agrupa operaciones en una transacción en las que todas terminan o ninguna es realizada, bajo esta última situación se regresa al estado inicial.
- Que para implementarlas, el servidor debe de encapsular (aislar) los recursos que maneja y administra.
- Que las transacciones intentan eliminar algunos problemas debidos a las concurrencias. Por ejemplo, si dos usuarios ejecuten al mismo tiempo Rename ("x","y") y Rename ("x", "z") en un servicio de directorio.

6.2 MODELO DE TRANSACCIONES

Por medio de este se puede modelar con mayor precisión las transacciones a través del uso de procesos independientes. Entre los puntos a considerar están:

- *Almacenamiento estable*: Permite que la información perdure a todo, con excepción de las catástrofes, para lo cual se realiza un disco espejo.
- *Primitivas de transacción*: Son proporcionadas por el sistema operativo o compilador, como *begin - transaction, end - transaction, abort- transaction, read, write*.
- *Propiedades de las transacciones*: Son las propiedades fundamentales de las transacciones y se indican como ACID, si se cumple con lo siguiente:
  - *Atomicidad*: Las transacciones se ejecutan completamente o se abortan.
  - *Consistencia*: El resultado de una transacción no debe de depender del número de clientes concurrentes.
  - *Aislamiento*: Los efectos intermedios de las transacciones no deben de ser visibles a otras.
  - *Durabilidad*: La información debe de guardarse en disco una vez que la transacción se ejecuta exitosamente.
- Si una transacción se aborta, es necesario un procedimiento de recuperación para eliminar las operaciones ya realizadas.

6.3 PROBLEMAS DEBIDO A LA CONCURRENCIA DE TRANSACCIONES

Entre los principales objetivos del servidor de transacciones se encuentran maximizar la concurrencia y usar la serie equivalente. Las operaciones atómicas tienen que ser capaces de bloquear el acceso de varios usuarios a una región crítica. El problema de modificaciones y pérdidas sucede cuando una transacción aborta. El problema de reportes inconsistentes ocurre cuando una transacción lee un dato antes que otra transacción actualice. Una solución sería usar la serie equivalente de transacciones por las entrelazadas de instrucciones. La equivalencia en serie es usada como criterio para la derivación de diferentes mecanismos de control de concurrencia. El servicio de transacciones asegura que el resultado será el mismo si se realizan las transacciones de manera secuencial.

6.4 RECUPERACIÓN DE TRANSACCIONES

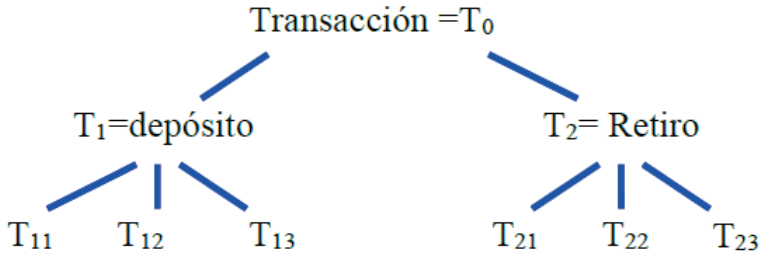
Un servicio de transacciones tiene que ser capaz de abortar transacciones y recuperarse de los efectos ocasionados por ello. Entre algunos de los problemas relacionados al aborto de las transacciones, se pueden citar los siguientes: lecturas sucias, cascada de aborto, y escrituras prematuras. El problema de "lecturas sucias" ocurre cuando una transacción exitosa lee datos que han sido afectados por una transacción abortada. Una solución a este problema es retrasar la decisión de declarar exitosa una transacción hasta que todas las transacciones relacionadas sean exitosas. Para el problema de "cascada de aborto" se propone como solución que se retrasen las lecturas hasta que las transacciones que se encuentran escribiendo al mismo dato, aborten o tengan éxito. El problema de las "escrituras prematuras" se presenta cuando dos transacciones escriben al mismo dato al mismo tiempo. Para esto se propone como solución retrasar las operaciones de escritura. Usando versiones tentativas normalmente se pueden evitar retrasos.

6.5 TRANSACCIONES ANIDADAS Y DISTRIBUIDAS

6.5.1 Transacciones anidadas

Una transacción anidada (TA) está formada por un conjunto de transacciones, cada una de las cuales puede estar a su vez formada por transacciones, y así sucesivamente, como se observa en la figura 6.1.

Figura 6.1. Ejemplo de transacción anidada



Las transacciones anidadas son útiles para:

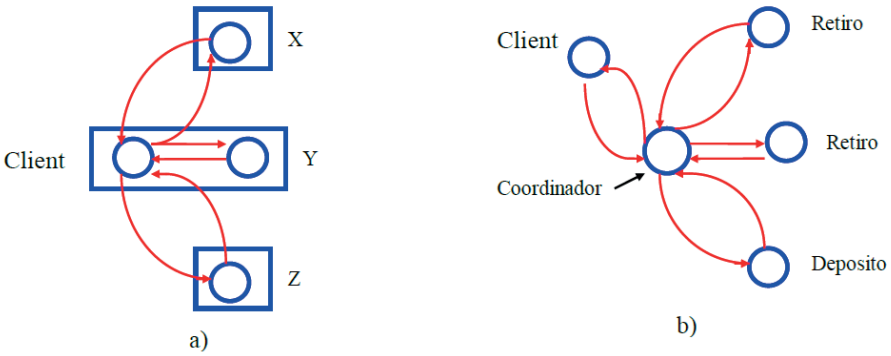
- Dar una jerarquía a las transacciones según su importancia.
- Especificar la concurrencia.
- Mejorar el control de las transacciones. Por ejemplo, una transacción anidada puede abortar sin que su padre tenga que hacerlo.
- Modelar transacciones distribuidas, las cuales son transacciones cuyas partes se realizan en diferentes sitios.

6.5.2 Transacciones distribuidas

En este tipo de transacciones, una transacción implica a muchos servidores (ver figura 6.2a), por lo que existe la posibilidad de datos replicados. Un coordinador (ver figura 6.2b) es el primer servidor que atiende las transacciones y es responsable de [Coulouris *et al.*, 2001]:

- Abortar o declarar exitosa la transacción.
- Utilizar nuevos servidores para la transacción.

Figura 6.2. a) Esquema de la transacción distribuida,
b) Coordinación de transacciones



6.6 IMPLANTACIÓN DE SISTEMAS DISTRIBUIDOS

Para la implantación de sistemas distribuidos se usan comúnmente dos métodos:

- *Espacio de trabajo individual:* Consiste en que cuando un proceso inicia su transacción se le asigna un espacio de trabajo individual, que contiene

los archivos que va a acceder. Uno de los inconvenientes de este método es el costo de copiar todo.
- *Bitácora de escritura anticipada*: Aquí los archivos se modifican, pero antes de hacerlo se deberá de escribir en una bitácora un registro indicando la transacción que realiza el cambio, el archivo y bloque modificado, así como los valores anteriores y nuevos.

6.7 TRANSACCIONES CON REPLICACIÓN

Existen tres modelos para atender peticiones en un sistema de transacciones con replicación:

- *Asíncrono:* Son modelos donde las peticiones son procesadas por servidores de réplicas locales.
- *Síncrono:* En este modelo las peticiones de modificación son procesadas en el mismo orden en todas las réplicas (orden total).
- *Mixto:* En este modelo ciertas réplicas pueden ser contactadas y procesadas por el mismo orden.

El modelo seleccionado depende de la razón entre el número promedio de lecturas y escrituras. El orden en que pueden ser procesadas dos peticiones r_1 y r_2 en los servidores de réplicas son:
- *Total*: Si r_1 es procesada antes que r_2 en todos los servidores de réplicas o viceversa.
- *Casual*: Si $r_1 \rightarrow r_2$, lo que implica que r_1 sea procesado antes que r_2 en todos los servidores de réplicas.

El ejemplo anterior sería síncrono si la ejecución de r_1 implica que todas las peticiones fueron realizadas antes en todos los servidores de réplicas.

Una petición no será procesada por un servidor de réplicas hasta que las restricciones de orden sean cumplidas. La petición es guardada en una cola. Entre las propiedades de la computadora solicitante se pueden mencionar las siguientes:

- *Seguridad*: Se refiere a que ningún mensaje puede ser procesado fuera de orden.

- *Vida limitada*: Ningún mensaje permanecerá esperando en la cola indefinidamente.

Ejemplos de arquitecturas de replicación son [Coulouris *et al.*, 2001]:

- *Gossip*: Esta arquitectura proporciona servicios altamente disponibles por medio del uso de réplicas.
- *Isis*: En esta arquitectura los grupos de procesos se clasifican de acuerdo con el patrón de comunicación seguido, que puede ser del mismo rango, de servidores, de cliente - servidor y de difusión.

6.8 PROBLEMA DEL "DEADLOCK" EN LOS SISTEMAS DISTRIBUIDOS

El problema del deadlock o candado mortal en los sistemas distribuidos es similar al que se presenta en los sistemas de un solo procesador, pero con mayor dificultad, ya que evitarlos, prevenirlos, detectarlos y remediarlos se complica aún más al tener la información dispersa en varias computadoras.

Para afrontar el problema del deadlock en los sistemas distribuidos, se consideran las siguientes estrategias [Coulouris *et al.*, 2001]:

1. *Ignorar el problema*.
2. *Detección:* Esta estrategia permite que el deadlock se presente, lo detecta e intenta recuperarse de ellos. Dos son los tipos de algoritmos a resaltar:
   - Algoritmos centralizados
   - Algoritmos distribuidos
3. *Prevenir*: Esta estrategia no permite que el deadlock se presente estructuralmente, para lo cual asume transacciones atómicas y tiempo real.
4. *Evitarlos*: Se realiza una asignación cuidadosa de recursos pero casi nunca se utiliza.

6.9 SERVICIOS WEB

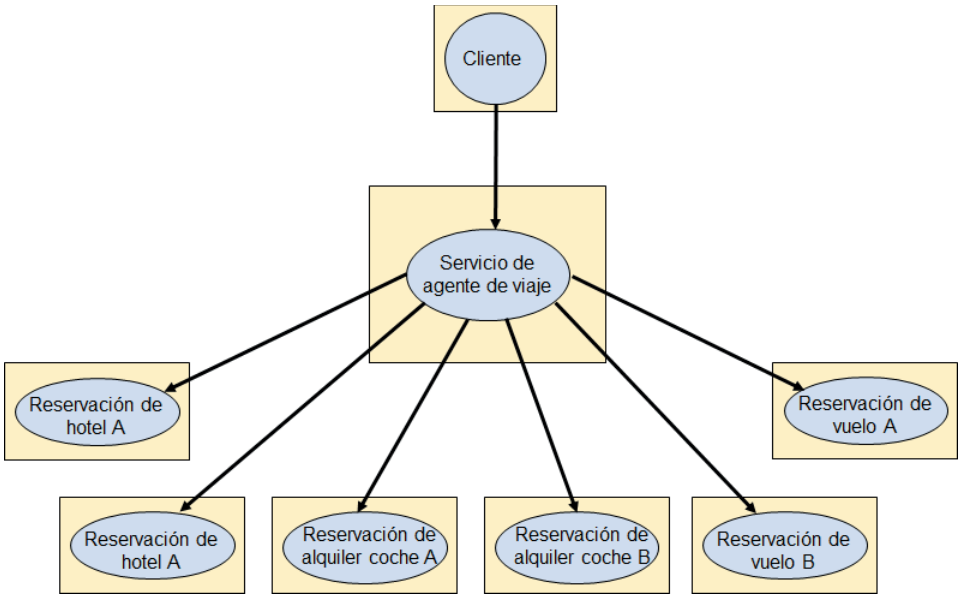
Un servicio web ofrece una interfaz de servicio que permite a los clientes interactuar con los servidores de una manera más general que como los navegadores lo hacen. Los clientes acceden a las operaciones en la interfaz de un servicio web a través de solicitudes y respuestas con formato en XML

(eXtensible Markup Language) y generalmente se transmite a través protocolos HTTP (Hypertext Transfer Protocol) [Coulouris *et al.*, 2012]. La interfaz de servicio web consiste generalmente en una colección de operaciones que puede ser utilizada por un cliente a través de Internet. Las operaciones en un servicio web pueden ser proporcionadas por una variedad de recursos, por ejemplo, los programas, los objetos o las bases de datos. Un servicio web puede ser manejado por un servidor web junto con las páginas web o ser un servicio independiente. La representación de datos externos y de clasificación de mensajes intercambiados entre los clientes y los servicios web se hace en XML.

Los servicios web proporcionan una infraestructura para mantener de una forma más rica y estructurada la interoperabilidad entre clientes y servidores. Asimismo, los servicios web proporcionan una base por la que un programa cliente en una organización puede interactuar con un servidor en otra organización sin supervisión humana. En particular, los servicios web permiten que aplicaciones complejas a ser desarrolladas por prestación de servicios integren otros servicios. Es decir, los servicios web pueden actuar como componentes independientes que se integrar entre sí para formar sistemas distribuidos complejos.

Un escenario típico de un servicio web es la integración de un conjunto de aplicaciones de distintas empresas u organizaciones. Un ejemplo es un agente de reservación de viajes. Para viajar, muchas personas realizan la reservación de vuelos, hoteles y alquiler de coches en línea por medio de diferentes sitios web. Si cada uno de estos sitios web es para proporcionar una interfaz de servicios web estándar, entonces un "servicio de agente de viajes" podría utilizar sus operaciones para proporcionar a un cliente viajero una combinación de estos servicios. Este escenario se ilustra en la figura 6.3.

Figura 6.3. Ejemplo de uso de servicios web [Coulouris *et al.*, 2012]



Es posible que un servicio de agente de viajes use patrones de comunicación alternativos disponibles en el servicio web. En este caso, el primer patrón sería para el soporte a documentos de consulta con el fin de agilizar la reserva. Esto se logra con el intercambio asíncrono de diferentes documentos basados en fechas y destinos. Por ejemplo, diferentes hoteles en determina ciudad, diferentes líneas aéreas a un destino o diferentes compañías para rentar autos. Otro patrón de comunicación podría ser para el control de los datos de la tarjeta de crédito, en este caso las interacciones con el cliente deben de ser apoyadas por un protocolo de petición-respuesta.

Entre las características clave de un servicio web se pueden citar las siguientes [Coulouris *et al.*, 2012]:

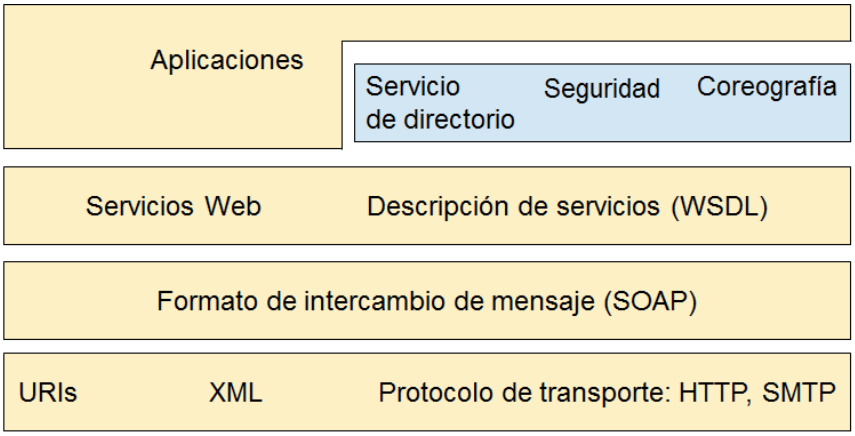
- Interoperabilidad.
- Estándares abiertos.
- Acoplamiento débil.

La interoperabilidad implica que distintas aplicaciones desarrolladas en diferentes lenguajes de programación y desplegadas en cualquier plataforma de cómputo pueden intercambiar datos a través de los servicios web.

Los servicios web soportan interoperabilidad a través de la Internet global, incluyendo el área clave de integración de empresa-a-empresa y también proporcionan el middleware subyacente tanto para el cómputo grid como para el cómputo en la nube.

El éxito de la interoperabilidad en los servicios web se logra gracias a la adopción de protocolos y estándares abiertos. Los responsables de la estandarización y arquitectura de los servicios web son el World Wide Web Consortium [W3C, 2014] y The Organization for the Advancement of Structured Information Standards [OASIS, 2014]. En la figura 6.4 se muestra un conjunto (pila) de servicios y protocolos de los servicios web.

Figura 6.4. Componentes e infraestructura de los servicios web [Coulouris *et al.*, 2012]



Donde:

- *XML (Extensible Markup Language)*: Es el lenguaje estándar para los datos que se intercambian. Es flexible y extensible.
- *SOAP (Simple Object Access Protocol)*: Son los protocolos sobre los que se establece el intercambio entre extremos.
- *WSDL (Web Services Description Language)*: Es el lenguaje de la interfaz pública para los servicios web. Está basada en XML e incluye la información necesaria para suplir la ausencia de un middleware.
- Protocolos *HTTP (Hypertext Transfer Protocol), FTP (File Transfer Protocol)* o *SMTP (Simple Mail Transfer Protocol)*: Son usados para enviar los datos XML de una aplicación a otra.

- *UDDI (Universal Description, Discovery and Integration)*: es el protocolo para publicar la información de los servicios web.
- *WS-Security (Web Service Security)*: es el protocolo de seguridad para garantizar la autenticación de los actores y la confidencialidad de los mensajes enviados.
- *URI (Uniform Resource Identificator)*: es una secuencia compacta de caracteres que identifca un recurso físico o abstracto. Está definido por el RFC 3986. Ejemplos de URI pueden ser URL (Uniform Resource Locator) o URN (Uniform Resource Name).

Con el propósito de reducir el riesgo de que un cambio en un servicio web tenga un efecto en cadena sobre otros servicios, hay un interés en la articulación flexible o acoplamiento débil en los sistemas distribuidos. En el contexto de los servicios web, la articulación flexible se refiere a minimizar las dependencias entre servicios, con el fin de tener una arquitectura subyacente flexible.

Los servicios web se pueden utilizar sobre cualquier protocolo de comunicación, sin embargo, el más usado es el TCP (Transmission Control Protocol). EL TCP usa el puerto 80, el cual no es bloqueado por el cortafuego (firewalls) de las organizaciones. Este es generalmente el puerto usado por los navegadores de Internet. De esta manera, los servicios web pueden usar este puerto para no ser bloqueados.

EJERCICIOS

1. Explica la propiedad A.C.I.D. en las transacciones.

2. Cita las dos reglas de operación de los candados en las transacciones.

3. ¿Qué pasa con un dato cuando se aborta una transacción?

4. Cita algunos ejemplos del uso de los servicios web.

5. Desarrolla un ejemplo de escritura prematura para una concurrencia.

6. Indica la razón principal por la que los servicios web son soportados por el TCP.

7. Explica el caso de una lectura sucia en una transacción.

8. ¿Por qué el "acoplamiento débil" es una característica deseada en los servicios web?

9. Describe un ejemplo de transacción anidada.

10. Describe un ejemplo de transacción distribuida.

11. ¿Cuál es la función de los candados en una transacción?

Actividad integradora

1. De la aplicación de los contenidos, así como de las habilidades y actitudes desarrolladas.
   Se recomienda exposiciones por parte del profesor, así como una explicación de las características fundamentales de las transacciones distribuidas y su vinculación con la replicación, consistencia, localización de contenido, candados, recuperación y los deadlock. A modo de taller, se recomienda la realización de ejemplos sobre diversos casos a través de los cuales una transacción pueda caer en inconsistencia, tanto en el aula como de modo extraclase. Se recomienda desarrollar el contenido en un promedio de dos a tres sesiones que incluyan trabajo intenso de ejemplos demostrativos sobre la manera en que trabajan las transacciones, así como de los servicios web.
   El alumno debe de mostrar un fuerte interés por abstraer y modelar las transacciones distribuidas que pueden caer en estado de inconsistencia en los sistemas de cómputo distribuido. También tiene que demostrar habilidades para programar algoritmos relacionados al deadlock en las transacciones distribuidas en algún lenguaje de programación de alto nivel.
2. Del logro de los objetivos establecidos en el capítulo o apartado del material.
   Se espera alcanzar los objetivos de este capítulo con el apoyo en la solución de los ejercicios, así como en talleres con diversos casos de transacciones distribuidas que eviten inconsistencia o el deadlock en los sistemas distribuidos.

Capítulo 7. Sistemas operativos distribuidos

Objetivo: Que el alumno comprenda la importancia de los sistemas operativos distribuidos y la funcionalidad de sus componentes para el soporte de aplicaciones distribuidas de cómputo.

7.1 Introducción

Los sistemas operativos distribuidos presentan algunas de las siguientes ventajas [Sánchez, 1995]:

- Facilitan la implementación de sistemas distribuidos.
- Proveen abstracciones de los recursos en un sistema distribuido, por ejemplo, canales de comunicación y procesos en lugar de redes y procesadores.
- En los sistemas abiertos no existe una clara división entre el sistema operativo distribuido y las aplicaciones que se ejecutan en él.

7.2 Núcleo y servidores

Los núcleos y servidores de un sistema operativo distribuido administran recursos los cuales deben proveer las siguientes funciones [Sánchez, 1995]:

- *Encapsulamiento*: Deben de tener una interfaz de servicio útil para que el cliente pueda acceder sus recursos sin necesidad de que conozca los detalles de implementación.
- *Procesamiento concurrente*: implica que los recursos y accesorios del sistema pueden ser compartidos de manera simultánea por los clientes.

- *Protección*: Se deben de proteger los recursos contra accesos no autorizados.
- *Invocación*: se refiere al acceso a un recurso encapsulado y tiene las siguientes tareas relacionadas:
  - *Resolución de nombres*: El servidor que administra un recurso debe de ser localizado por medio del identificador del recurso.
  - *Comunicación*: Los parámetros y resultados se tienen que enviar a los administradores de recursos a través de una red.
- *Itinerario*: Cuando se invoca una operación, su procedimiento debe de ser itinerado dentro del núcleo o servidor.

Características del núcleo

El núcleo de un sistema operativo se caracteriza porque es un programa que se ejecuta con privilegios de acceso a los dispositivos físicos de la computadora huésped, como son los registros de memoria. El núcleo se responsabiliza de separar y proteger los espacios de memoria de todos los procesos, así como de la gestión de los registros. Los recursos del núcleo pueden ser accedidos por los procesos por medio de llamadas al sistema.

Los núcleos pueden ser [Sánchez, 1995]:

- *Monolíticos*: Cuando realizan todas las funciones básicas del sistema operativo, por ejemplo el UNIX. Un problema de estos núcleos es que ocupan mucho espacio (aproximadamente 200 MB).
- *Microkernel:* En este enfoque el núcleo es muy pequeño y sólo contienen las funciones más básicas tales como la comunicación entre procesos, servidores y memoria. El resto de funciones es previsto por servidores cargados dinámicamente (por ejemplo: MACH).

7.3 NOMBRAMIENTO Y PROTECCIÓN DE RECURSOS

7.3.1 Nombrado de recursos

Un servicio que administra recursos debe de dar un nombre a cada uno de estos recursos. Este nombre debe de ser independiente de su localización (memoria, computadora). Por ejemplo, el uso de recursos en Chorus y Amoeba requiere [Coulouris *et al.*, 2001], [Tanenbaum, 1996]:

- Puerto del servidor
- Identificación del recurso
- Conocer el puerto y el identificador del recurso
- Los nombres deben de ser únicos y accesibles para ser localizados
- Usar mensajes a puertos para manejar recursos

La ventaja del nombramiento de recursos es que existe una transparencia de red que es la que busca un sistema operativo distribuido, porque el servidor puede estar en otra máquina o en la máquina local y sin embargo debe ser transparente.

7.3.2 Protección de recursos

La protección de recursos se refiere a que los clientes solo podrán acceder a aquellos recursos para los cuales tengan permisos de acceso. La protección de recursos se logra implementando un dominio de protección, el cual es un conjunto de reglas de acceso de recursos compartidos por un conjunto de procesos. Existen dos formas de realizar la implementación:

1. Los procesos contienen un conjunto de capacidades o identificadores de servicios (con sus respectivos derechos) que pueden usar, y cada proceso conoce qué recurso usar.
2. Cada recurso tiene una lista de control de acceso, la cual es concentrada en el servidor como una lista de las personas que pueden tener acceso al sistema.

7.4 Casos de sistemas operativos distribuidos

Diferentes sistemas operativos distribuidos han sido desarrollados por consorcios industriales, centros de investigación y universidades. Algunos ejemplos de sistemas distribuidos son:

- *Mach*: Sistema de punta con un diseño flexible de memoria.
- *Chorus*: Sistema de punta con diseño de microkernel.
- *DCE*: Sistema comercial con ambiente coherente para aplicaciones distribuidas.

A continuación se revisa brevemente las principales características de estos sistemas.

7.4.1 Mach

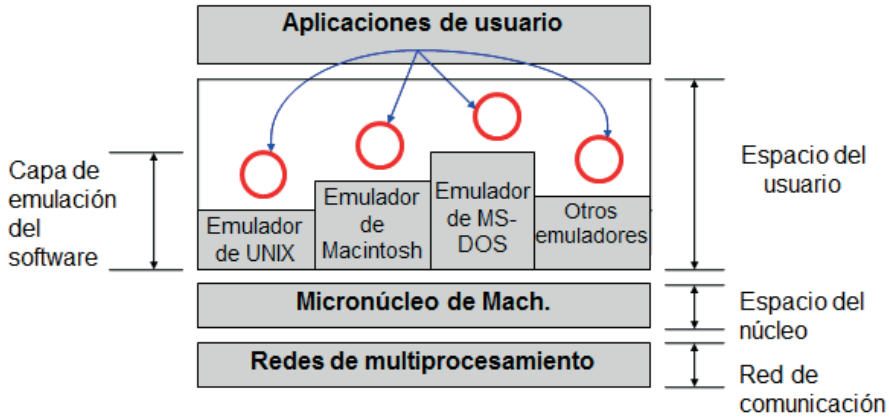
7.4.1.1 Generalidades

El sistema operativo Mach es un proyecto de la Universidad Carnegie Mellon (CMU), desarrollado en 1986. Entre los objetivos de este sistema se encuentran:

- Crear una base para emular otros sistemas.
- Proveerlo de facilidades avanzadas en el núcleo.
- Permitir la transparencia en el acceso a recursos en red.
- Explorar el paralelismo.

La versión 2.5 de Mach fue usada por el sistema OSF/1 como base. La versión 3 eliminó la emulación de UNIX del núcleo. Mach puede ejecutarse en máquinas con procesadores 386, 486, estaciones DEC y SPARC, además puede funcionar en máquinas con un solo procesador o con múltiples procesadores. El acceso al núcleo puede ser un problema en Mach si este ha sido mal diseñado. Mach permite manejar memoria virtual como si fuera memoria compartida. Los recursos son manejados a través de mensajes de puertos en los servidores a nivel usuario. Mach permite la portabilidad, ya que el código dependiente de máquina puede ser asilado. En la figura 7.1 se muestra una arquitectura de Mach.

Figura 7.1. Arquitectura de Mach [Tanenbaum, 1996]



7.4.1.2 El micronúcleo y los puertos en Mach

El micronúcleo en Mach controla las siguientes abstracciones:

- *Puertos:* Canal de comunicación unidireccional que tiene asociado un buffer. A través de estos puertos se puede enviar o recibir mensajes.
- *Proceso:* Ambiente donde se lleva a cabo la ejecución, contiene un espacio de direcciones protegidas y un conjunto de capacidades para acceder a puertos.
- *Hilos:* Los procesos están constituidos de diferentes hilos que se pueden ejecutar en paralelo a diferentes procesadores de una máquina con memoria compartida.
- *Mensajes:* Es un conjunto de datos o peticiones para acceder a recursos.
- *Objetos de memoria*: Es una instancia de un tipo de dato abstracto, por cada objeto de memoria existe un objeto del núcleo que contiene un caché de las páginas residentes en memoria.

Características de los puertos en Mach

Los puertos sirven para acceder a recursos. Existen tres diferentes tipos de derechos para acceder a un puerto:

- De envío.
- De envío una sola vez.
- De recepción.

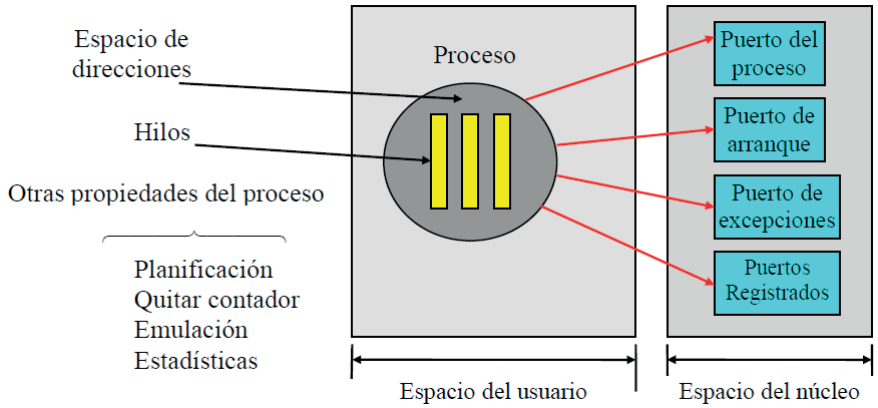
En un puerto varias tareas pueden tener derechos de envío pero solo una tarea tendrá los de recepción.

7.4.1.3 Administración de procesos en Mach

Procesos en Mach

Los procesos son solo medios de ejecución, ya que solo los hilos que se encuentran en él pueden ejecutar operaciones. En la figura 7.2 se muestra la operación de un proceso.

Figura.7.2. Un proceso en el sistema Mach [Tanenbaum, 1996]



Los puertos que se muestran en la figura 7.2 tienen las siguientes funciones:

- *Puerto de arranque*: Usado en la inicialización de todos los parámetros al comenzar el proceso.
- *Puerto de excepciones*: Se utiliza para informar de las excepciones ocasionadas por el proceso.
- *Puerto de proceso*: Permite la comunicación con el núcleo.
- *Puerto de registrados*: Permite al proceso una forma de comunicación con otros servidores estándar del sistema.

Hilos en Mach

Los hilos son entidades activas en Mach que ejecutan instrucciones, controlan sus registros y espacios de direcciones. Cada hilo pertenece exactamente a un proceso específico, el cual no podrá realizar algo si no tiene por lo menos un hilo. La creación de hilos en Mach sigue el modelo de UNIX y usa la llamada *fork*, mientras que con la llamada *exit* termina un hilo su trabajo. Cuando un hilo es creado se le asigna un puerto para que pueda acceder al núcleo.

7.4.1.4 Modelo de comunicación en Mach

La comunicación en Mach está basada en los puertos, los cuales son objetos del núcleo que contienen mensajes. Un mensaje está constituido de un encabezado de tamaño fijo, seguido por una lista variable de datos. El encabezado de un mensaje está compuesto de los siguientes elementos:

- Puerto destino
- Puerto respuesta
- Identificador de operación
- Tamaño de la lista de datos

Cada dato en la lista puede ser un apuntador a datos, un dato tipo o tipos de derechos sobre puertos. Se usa la llamada *mach_msg* para el envío y recepción de mensajes. Los puertos tienen un buffer cuyo tamaño puede ser ajustado. Si se intenta enviar un mensaje por un puerto que tenga su buffer lleno, el hilo se bloqueará. Los mensajes pueden ser recibidos en un conjunto de puertos. La confiabilidad en la comunicación del sistema existe y, además, permite la entrega ordenada de los mensajes. Servidores de mensaje son usados para realizar la comunicación a través de la red. Estos servidores están constituidos por varios hilos, los cuales llevan a cabo distintas funciones. La comunicación externa se realiza por los puertos de red, los cuales soportan el protocolo TCP/IP.

7.4.1.5 Manejo de memoria en Mach

Mach cuenta con un sistema de memoria flexible basado en la paginación, el cual separa las partes dependientes de la máquina de las partes independientes, lo que le permite una portabilidad. Las páginas solo se copian cuando han sufrido modificación y para esto se usa la llamada *copy*. Debido a que Mach no soporta archivo, debe usar el paginador externo, el cual tiene, entre sus funciones:

1. El manejo del respaldo de datos eliminados por el núcleo de su caché.
2. El mantener un registro de las páginas virtuales en uso.
3. El definir las restricciones necesarias que permitan mantener la consistencia de los datos.

Mach maneja una memoria virtual grande y lineal. Cuando se crea una tarea, se hereda memoria, la cual podrá ser compartida o copiada.

7.4.2 Chorus

7.4.2.1 Generalidades

Este sistema es un proyecto del INRIA, desarrollado en 1980 [Tanenbaum, 1996]. En 1997 el Chorus fue adquirido por Sun Microsystems. Algunos de los objetivos de Chorus son:

- Permitir una emulación de UNIX de alto rendimiento.
- Para usar en sistemas distribuidos.
- Aplicaciones en tiempo real.
- Integrar programación orientada a objetos.
- Permite servidores de grupo y ser reconfigurable.

Algunos aspectos que guardan en común Chorus y Mach son:

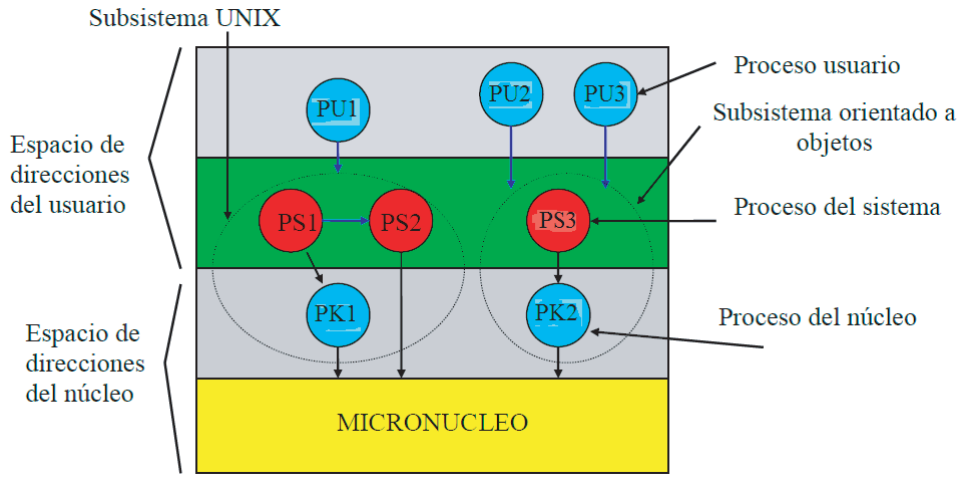
- Operación paralela con memoria compartida.
- Transparencia a nivel de red.
- Emulación de UNIX y otros sistemas operativos.
- Flexibilidad al implementar la memoria virtual.
- Portabilidad.
- Tienen servidores de grupo reconfigurable.

Estructura

Chorus se estructura en capas, con el *micronúcleo* en la parte inferior, que lleva una mínima administración de los nombres, procesos, hilos, memoria y mecanismos de comunicación. Los *procesos del núcleo* se sitúan por encima del microkernel y se cargan de manera dinámica para cada nuevo programa permitiendo adicionar funcionalidad al microkernel sin aumentar su tamaño y complejidad. Los *procesos del sistema* se encuentran en la capa superior siguiente, se ejecutan en modo usuario y envían mensajes a los procesos del núcleo. Finalmente, en la capa superior se encuentran los procesos de usuario, los cuales no podrán llamar directamente al microkernel, excepto los definidos para tal fin. La figura 7.3 ilustra la estructuración de estas capas en el sistema Chorus. El micronúcleo conoce qué subsistema ocupa cada proceso

usuario, por lo que las llamadas de este proceso se restringirán únicamente a las ofrecidas por este.

Figura 7.3. Estructura de capas en Chorus [Tanenbaum,1996]



7.4.2.2 Micronúcleo del sistema Chorus

El Micronúcleo en el sistema Chorus controla las siguientes abstracciones:

- *Actores*: Son en esencia equivalentes a los procesos en otros sistemas operativos y son un medio que encapsula a los hilos.
- *Hilos*: Son similares a un proceso, ya que tienen una pila, apuntador a la pila, contador de programas y registros, pero comparten el mismo espacio de direcciones.
- *Puertos:* Es el canal de comunicación que tiene asociado un buffer para recepción de mensajes. Los puertos se pueden agrupar para que una tarea pueda mandar o recibir datos.
- *Mensajes*: Datos o peticiones para poder acceder a un recurso, los cuales tienen una parte fija y otra de tamaño variable. El cuerpo del mensaje contiene la información enviada por el emisor.
- *Regiones:* Es un rango consecutivo de direcciones que se asocian con alguna pieza de datos, ya sea programa o archivo.
- *Segmento*: Es el conjunto lineal de bits en la cual podemos asociar a regiones y que son guardadas en el núcleo en un caché local.

7.4.2.3 Administración de procesos en Chorus

Actores en Chorus

Un actor (o proceso) en Chorus es un conjunto de elementos activos y pasivos que se ejecutan en grupo para realizar cierto trabajo. En el sistema Chorus, los actores y los hilos pueden ser cargados de manera dinámica en el espacio del núcleo. Del mismo modo, los servidores se pueden cargar activamente, ya sea en el espacio del núcleo o del usuario. Existen tres tipos de privilegios para los actores en Chorus:

- *De usuario*: Solo algunas llamadas a sistema.
- *De sistema*: Permite las llamadas al sistema.
- *De núcleo o supervisor*: Tiene acceso completo.

Cada actor en Chorus tiene asociado un *identificador de protección*, el cual les proporciona un mecanismo de autentificación.

Hilos

En Chorus, los hilos son los que ejecutan el código en un actor y tienen su propio contexto privado. Siempre permanecen unidos al actor en el que fueron creados. Es función del núcleo conocer y planificar cada hilo. Los hilos se comunican entre sí, ya sea enviando o recibiendo mensajes, sin importar que los hilos receptores se encuentren en otros actores. Un hilo distingue los siguientes estados no excluyentes (un hilo puede estar en más de un estado al mismo tiempo):

- *Activo*: El hilo se puede ejecutar.
- *Suspendido*: El hilo ha sido intencionalmente suspendido.
- *Detenido*: El actor donde se encuentra el hilo se ha suspendido.
- *Espera*: El hilo está en espera a que un evento se realice.

Chorus usa dos registros de software para solucionar el problema de administrar los datos particulares y su pila en un hilo. En cada registro está un apuntador a los datos particulares cuando opera en modo usuario y otro apuntador cuando el hilo envía una señal al núcleo.

7.4.2.4 Manejo de memoria en Chorus

Chorus guarda varias semejanzas con respecto al manejo de memoria con Mach. Chorus usa los siguientes conceptos [Tanenbaum, 1996]:

- *Regiones*: Que corresponde a un rango en serie de direcciones virtuales.
- *Segmento:* Es una colección adyacente de bytes que recibe el nombre y protección de una posibilidad. Ejemplos de segmentos son los archivos y las áreas de intercambio.
- *Asociadores*: Son usados para controlar uno o más segmentos a asociar con regiones.

Chorus soporta la memoria compartida distribuida clásica usando un algoritmo descentralizado dinámico, donde el segmento es la unidad a ser compartida. Las llamadas más importantes soportadas por el administrador de memoria se relacionan con *administración de regiones*, *administración de segmentos*, *asociadores*.

7.4.2.5 Comunicación en Chorus

La comunicación en Chorus está basada en la transferencia de mensajes. Un mensaje está constituido por un encabezado que contiene la fuente y destino, una parte fija opcional y un cuerpo opcional, estas dos últimas partes están bajo el control del usuario. Existen tres tipos de mensajes a grupos:

- *Por difusión*: Llega a todos los miembros del grupo.
- *Funcional*: Dirigido a algún miembro del grupo.
- *Selectivo:* A algún miembro del grupo donde se encuentra un recurso específico.

Chorus permite el agrupamiento de puertos para diferentes servidores y proporciona dos formas de comunicación: asíncrono y RPC.

7.4.2.6 Emulación de UNIX en Chorus

Aunque Chorus no tuvo como objetivo inicial la emulación con UNIX, se creó un subsistema (MiX) que permite la compatibilidad binaria. Los procesos, tanto en Chorus como en MiX, comparten el mismo espacio de memoria. El Modulo UNIX en Chorus permite una simulación de llamadas al sistema y

usa una protección de datos similar a la usada en UNIX. En este sistema las señales son controladas por el propio proceso a través de un hilo de control y permite la creación de procesos en computadoras remotas.

MiX está construido con base en los siguientes componentes:

- Administrador de procesos.
- Administrador de objetos.
- Administrador de control de flujo.
- Administrador de comunicación.

7.4.3 DCE

7.4.3.1 Generalidades y objetivos

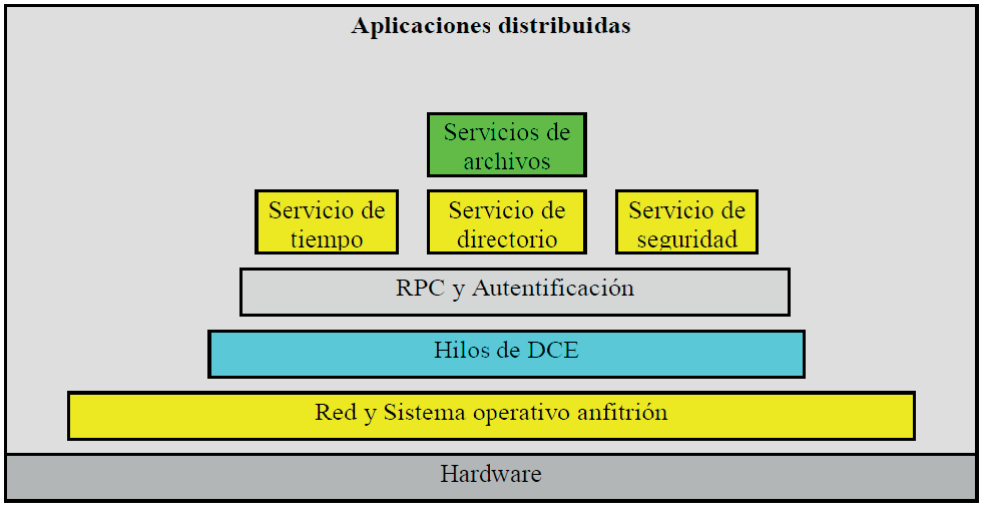
DCE (Distributed Compuyting Enviroment) fue desarrollado por OSF (Open Systems Foundation) que agrupa a compañías como IBM, Digital y Hewlett-Packard, en 1980 [Tanenbaum, 1996]. El objetivo de DCE es ofrecer un ambiente coherente como plataforma para el desarrollo de aplicaciones distribuidas. La base de DCE fueron los sistemas operativos existentes, en principio UNIX para después considerar VMS, WINDOWS y OS/2.

DCE tiene diversas herramientas y servicios, así como la infraestructura para que estos trabajen de manera integrada con el fin de facilitar aplicaciones distribuidas. DCE permite crear un mercado potencial al ejecutarse en distintas arquitecturas, sistemas operativos y redes, además de ofrecer transparencia al usuario. Mecanismos de autentificación y protección integrados en DCE permiten que este sistema brinde seguridad en red a un grupo de trabajo. DCE utiliza los protocolos TCP/IP u OSI para realizar la comunicación en red.

Estructura de DCE

El cliente - servidor es el modelo de DCE, ya que los procesos usuarios operan como clientes para obtener un servicio de un proceso servidor. Los servicios distribuidos proporcionados por DCE son, principalmente, servicio de tiempo, directorio, seguridad y sistema de archivos. Entre otras prestaciones de DCE, se encuentran los hilos y las RPC (llamadas a procedimientos remotos). La figura 7.4 indica el modelo DCE y cómo se relacionan las partes que la componen.

Figura 7.4. Modelo del DCE [Tanenbaum, 1996]



Las celdas en DCE son agrupamientos de usuarios, máquinas y otros recursos sobre la cual están basados aspectos como asignación de nombres, seguridad y administración. Para agrupar las celdas, se consideran las siguientes restricciones:

- *Finalidad*: Los agrupados en una celda deben trabajar con un objetivo común.
- *Seguridad:* Agrupar a los usuarios que mejor confianza tengan entre sí.
- *Costo:* Agrupar a los usuarios que por su localización representen menos costos.
- *Administración:* El nombrar a un administrador de la celda se facilita cuando las restricciones anteriores son previstas.

Con el fin de minimizar el número de operaciones entre celdas en DCE y considerando las restricciones indicadas, es recomendable tener una pequeña cantidad de celdas.

7.4.3.2 Los hilos de DCE

Los hilos constituyen una parte fundamental en DCE, entre los puntos a considerar sobre ellos están la planificación, la sincronización y las llamadas. Están basados en POSIX P1003.4a. Un hilo maneja cuatro estados:

- *Ejecución*: El hilo está en actividad con el CPU.
- *Listo*: El hilo se encuentra en espera de entrar en actividad.
- *En espera*: El hilo se bloquea mientras espera que un evento ocurra.
- *Terminado*: El hilo ha finalizado su actividad pero sigue existiendo.

La planificación en los hilos es lo que determina el tiempo y orden de ejecución de cada hilo. Los algoritmos de planificación a soportar por DCE son:

- *FIFO*: Ejecuta el primer hilo de la cola de máxima prioridad de un grupo de colas.
- *Round-robin*: Ejecuta cada hilo durante un *quantum* fijo para la cola más poblada.
- *Por omisión*: Usa round-robin pero mientras la prioridad sea mayor el hilo tendrá un *quantum* mayor.

Para realizar la sincronización de hilos en DCE se usan:

- *Mutex*: Existen los rápidos, recursivos y no recursivos.
- *Variable de condición*: Son usados con los mutex como otro mecanismo de sincronización.

7.4.3.3 RPC en DCE

Entre los objetivos del uso de RPC en DCE se encuentran:

- Permitir que un cliente acceda a un servicio remoto a través de un procedimiento local.
- Facilitar la escritura de programas cliente y su transparencia.
- Tener pocas modificaciones en el código al ejecutarse.

El RPC de DCE consta de distintos componentes, incluyendo lenguajes, bibliotecas, demonios y programas de utilerías, en la cual la interfaz se escribe en IDL (Interface Definition Language). La salida del compilador IDL en su salida está constituido por los archivos:

- *Archivo de encabezado*: Con el identificador, tipos, constantes y los prototipos de función.
- *Resguardo del cliente*: Se encarga de almacenar los procedimientos reales.

- *Resguardo del servidor:* Aloja los procedimientos llamados por el sistema cuando un mensaje está arriba.

Para conectarse al servidor, el cliente primero localiza cuál es la máquina servidora y después localiza el proceso correcto en el servidor usando una dirección que se asigna dinámicamente.

7.4.3.4 Los servicios en RPC

Servicio de tiempo

DCE tiene un servicio distribuido de tiempo (DTS) que sincroniza los relojes de las máquinas distantes, este registro de tiempos se realiza por intervalos. Las funciones de DTS se centran en mantener:

- La consistencia mutua de los relojes: El mismo valor de tiempo en todos.
- Manejar tiempo real: Implica que el valor coincida con el del mundo real.

Cuando un programa solicita a DTS una comparación de tiempos, las respuestas pueden ser las siguientes:

- Primer tiempo es menor.
- Segundo tiempo es menor.
- No se sabe cuál es menor.

El empleado del tiempo es un demonio de DTS que se ejecuta en el cliente manteniendo el tiempo sincronizado con los relojes remotos.

Servicio de directorios

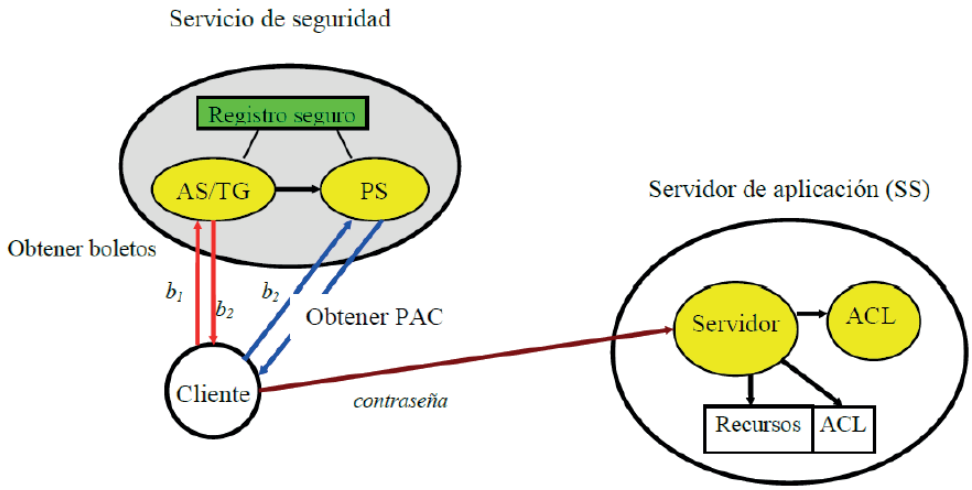
En DCE el servicio de directorio que nombra y registra todos los recursos está constituido por celdas, cada una con un servicio de directorio de celdas (CDS). Cada recurso tiene un nombre constituido por el nombre de la celda y el suyo en ella. Los mecanismos GDS (Global Directory Service) y DNS (Domain Name System) son usados por DCE para localizar celdas. CDS ordena los nombres de las celdas por jerarquía, aunque también permite los enlaces simbólicos. El GDS es un servicio secundario de DCE que le permite localizar celdas remotas, así como información arbitraria relativa a los directorios.

Servicio de seguridad

La seguridad en los sistemas distribuidos es esencial y DCE la proporciona en forma eficaz [Hu, 1995], [Tanenbaum, 1996]. Algunos conceptos de seguridad que involucra DCE son:

- *Principal*: Es un proceso que requiere seguridad en la comunicación. Tiene asociado un número binario como *identificador único de usuario*.
- *Autentificación*: Rutina para determinar si un *principal* es realmente quien dice ser, para lo cual DCE se basa en el sistema Kerberos del MIT (Massachusetts Institute of Technology).
- *Autorización*: Es usado para determinar a cuáles recursos tiene acceso el *principal* y se utiliza una *lista de control de acceso* (ACL).
- *Criptografía:* Transforma datos de tal manera que sea imposible recuperar los datos originales si no se posee una llave.
- El sistema de seguridad está constituido principalmente por los componentes que se indican en la figura 7.5.

Figura 7.5. Componentes elementales de la seguridad en DCE [Tanenbaum, 1996]



El *servidor de autentificación (AS)* es usado al iniciar el sistema o cuando un cliente entra en él. El *servidor de emisor de boletos (TGS)* se encarga de entregar el boleto al cliente o principal. El *servidor de privilegios (PS)* emite los certificados PAC. Cada usuario tiene una clave secreta solo conocida por él y el registro en el servicio de seguridad. Un *boleto* es un conjunto de

datos cifrados, que básicamente tiene las siguientes opciones: servidor pretendido, clave de sesión, cliente y tiempo de expiración, estos tres últimos cifrados mediante una clave temporal K_n.

Los pasos para obtener una RPC autentificada son [Tanenbaum, 1996]:

1. El cliente se autentifica con el Servidor de Autentificación (AS) usando una contraseña, el AS le entrega su llave secreta Kc y la clave C_1 cifradas en un boleto b_1 a usar con TGs.
2. El cliente envía el boleto b_1, cifrado con la clave C_1 al TGS, solicitando un boleto b_2 para comunicarse con el servidor de privilegios (PS). TGS verifica el boleto b_1, la clave C_1 y la marca de tiempo y, si esto es correcto, TGS envía al cliente un boleto b_2 cifrado que solo el cliente que cuenta con la clave K_c podrá abrir.
3. El cliente solicita con el boleto b_2 cifrado con C_2 al PS un certificado de atributos de privilegios (PAC) que contiene la identidad del usuario y grupo al que pertenece. El PAC es enviado al cliente en forma cifrada con C_2.
4. El PAC es enviado por el cliente al servidor emisor de boletos (usando C_3), quien descifra el PCA para corroborar su autenticidad y lo vuelve a cifrar con una nueva clave C_4 para enviarlo al cliente.
5. Finalmente el PAC cifrado con C_4 es enviado al servidor de aplicaciones (SS), quien lo descifrará exhibiendo la clave C_4. Entonces, el servidor responde con la clave C_5 que solo es conocida por el cliente y el servidor para permitir una comunicación segura.

Los recursos en DCE pueden estar protegidos por una lista de control de acceso (ACL) para indicar quién puede acceder y los derechos de acceso. Los derechos estándares soportados son: leer, escribir, ejecutar, modificar ACL, insertar recipiente, eliminar recipiente y probar.

Sistema distribuido de archivos (DFS)

El DFS en DCE permite que los procesos puedan acceder a archivos autorizados, aunque estos se localicen en celdas distantes. El DFS está constituido principalmente por la parte local y la parte de área amplia. El DFS usa hilos para el servicio simultáneo a varias solicitudes de acceso a un archivo, considerando a cada nodo como un cliente o servidor de archivo; guarda similitud con respecto a la interfaz con UNIX para abrir, leer y escribir archivos, así como en el montaje de archivos remotos. El DFS soporta archivos individuales, directorios, conjunto de archivos y particiones de disco (agregados en DCE).

EJERCICIOS

1. ¿Cuál es la función de los hilos en Mach?
2. ¿Cuál es la principal aportación de Mach a los sistemas distribuidos?
3. ¿Cómo trabaja el algoritmo LRU?
4. Cita tres aspectos que guardan en común Mach y Chrorus.
5. ¿Cómo se realiza la implementación de UNIX en Chorus?
6. Explica el concepto de micronúcleo.
7. ¿Para qué se extendieron la semántica de las señales de UNIX en Chorus?
8. ¿Cuáles son las principales diferencias del sistema operativo distribuido DCE con respecto a Chrorus y Mach?
9. ¿Cómo estructura su modelo de seguridad DCE?
10. ¿Cuál es la función del servicio de distribución de tiempo en el sistema operativo distribuido DCE? y ¿cómo opera?

ACTIVIDAD INTEGRADORA

1. De la aplicación de los contenidos, así como de las habilidades y actitudes desarrolladas.
   Se recomienda realizar una exposición general del tema para que el alumno comprenda la importancia de los sistemas operativos distribuidos, como el soporte para aplicaciones distribuidas, sin profundizar en detalles que extiendan el tema. Dado las generalidades del tema por el profesor, se puede desarrollar cada caso específico de sistema operativo distribuido en equipos formados por tres o cuatro estudiantes, con el fin de realizar una exposición de los casos en una sesión. Se recomienda desarrollar el contenido en dos sesiones que incluyan trabajo intenso con ejemplos demostrativos de la manera en que trabajan las transacciones, así como de los servicios web.

El alumno debe de mostrar un gran interés por entender los componentes básicos de los sistemas operativos distribuidos, así como profundizar la investigación de estos sistemas en la literatura científica existente.

2. Del logro de los objetivos establecidos en el capítulo o apartado del material.

   Se espera alcanzar los objetivos de este capítulo con el apoyo en la solución de los ejercicios, en las exposiciones realizadas por los alumnos, así como en la lectura de artículos de investigación por parte de los alumnos.

Capítulo 8. Sistemas de archivos distribuidos

Objetivo: Que el alumno comprenda la importancia de los sistemas de archivos distribuidos, sus componentes, estructura y las operaciones sobre ellos, así como algunos casos de sistemas de archivos distribuidos.

8.1. Introducción

Un archivo es una colección no contigua de bloques o conjunto de apuntadores a bloques del archivo en el índice de bloques que guardan una interfaz con el módulo cliente. Un archivo es la abstracción de datos guardados en almacenamiento secundario. Un directorio es un archivo especial que permite organizar los archivos. El sistema de archivos en un sistema operativo distribuido se encarga de la organización, almacenamiento, nombrado, repartición y protección de archivos. Importa notar que un sistema de archivos no es igual a una base de datos. Las transparencias requeridas en un sistema de archivos distribuidos son [Coulouris *et al.*, 2001]:

- Acceso.
- Localización.
- Concurrencia.
- Falla.
- Desempeño.

8.2 Servicios de archivos

Otros requerimientos para los sistemas de archivos distribuidos son heterogeneidad de replicación y de migración. Los módulos que constituyen el sistema de archivos son:

- *Servicio de archivos planos:* Está compuesto de módulos de archivos y de acceso a archivos. Se encarga de modificar el contenido de los archivos.
- *Servicio de directorios:* Está constituido por módulos de directorios y de control de acceso. Se encarga de definir un alfabeto y una sintaxis para formar los nombres de archivos.
- *Módulo cliente:* Está compuesto de módulo de bloques y dispositivos. Se encarga de proveer al usuario de una interfaz de programación. También se ocupa de las rutinas de bajo nivel para acceder a bloques de memoria y dispositivos.

8.3. SERVICIO DE ARCHIVOS PLANOS

Un servicio de archivos plano es un conjunto de operaciones simple y de propósito general. Los archivos contienen datos y atributos. La tolerancia a fallas se debe considerar para:

- Operaciones repetibles.
- Servidores sin estado.
- La atomicidad de operaciones como*:*
  - Read ( ).
  - Write ( ).
  - Create ( ).
  - Truncate ( ).
  - Delete ( ).
  - GetAttributes ( ).
  - SetAttributes ( ).

8.4 SERVICIO DE DIRECTORIO

El servicio de directorio proporciona una transformación entre nombres de texto para los archivos y sus UFID (Unique File Identifiers). Una UFID tiene el formato indicado en la figura 8.1. El UFID puede ser obtenido por los clientes de un archivo indicando su nombre de texto al servicio de directorio. Un UFID es único entre todos los archivos de todas las computadoras y es difícil de duplicar por personas externas. El UFID no es reusable y contiene información sobre la dirección física. Permite que los archivos sean divididos en grupos. Los archivos contienen campos de permisos encriptados.

Figura 8.1. Formato de la UFID

Identificador de grupo	Número de archivo	Número aleatorio	Permiso

Funciones delservicio de directorio

- Mapeo de nombres a UFIDs.
- Organización de los archivos planos en un árbol de directorios.
- Control del acceso a archivos.

8.5 MÓDULO CLIENTE

En cada computadora cliente se ejecuta un módulo conocido como módulo cliente. Este módulo cliente integra y extiende las operaciones del servicio plano de archivos con una interfaz de programación de aplicación disponible para los programas en las computadoras cliente. Un módulo cliente tiene las siguientes funciones:

- Se encarga del acceso a bloques de disco.
- Normalmente se tiene un sistema de caché para reducir tiempos de acceso.
- Usa el algoritmo LRU (Least Recently Used) para escoger un bloque que se regresa a disco.
- El cliente puede reducir retrasos de red usando un caché local.
- Uno de los inconvenientes es con la coherencia del caché.

8.6 SISTEMAS NFS Y AFS

Dos de los sistemas de archivos distribuidos más difundidos son los sistemas AFS y NFS [Coulouris *et al.*, 2001]. Las principales características de estos sistemas son descritas a continuación.

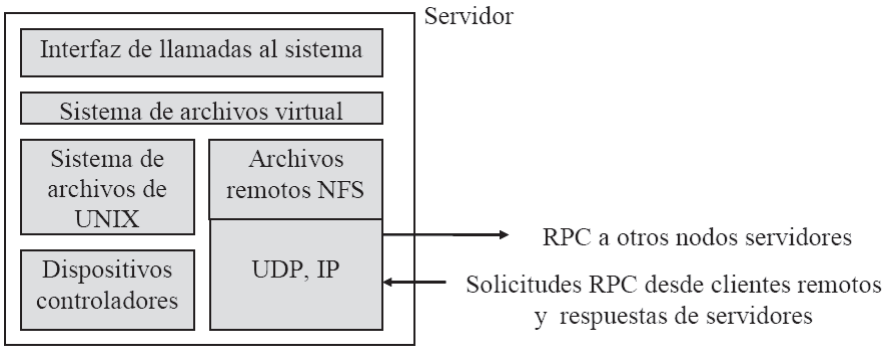
8.6.1 Sistema NFS (Network File Systems)

El sistema de archivos NFS fue diseñado por la compañía Sun Microsystems en 1985, con el propósito de ejecutarse en una red local (LAN). Entre las características de un sistema NFS se puede destacar lo siguiente:

- Es el estándar para tener acceso a archivos distribuidos en UNIX.
- Utiliza el modelo cliente - servidor y ambos se encuentran en el núcleo del sistema operativo.
- Permite la heterogeneidad de arquitecturas.
- Permite las transparencias de acceso, localización, fallas y desempeño.
  - Alcanza transparencia de localización.
  - La transparencia de migración es implementada parcialmente.
  - Las transparencias de replicación, concurrencia y escalabilidad no son implementadas.
  - NFS utiliza RPC. Es un servidor sin estados, por lo que no mantiene abiertos los archivos ni guarda información sobre ellos.

Para su implementación en UNIX, NFS define nuevas capas en el sistema de archivos de UNIX, como se muestran en la figura 8.2.

Figura 8.2. Estructuras de capas de NFS embebidas en el núcleo de UNIX



NFS utiliza un sistema de archivos virtuales VFS que puede distinguir entre archivos locales y archivos remotos. Usa un *v-node* en lugar de un *i-node*. Al *v-node* se le añade un identificador de sistema. NFS emula la semántica del sistema de archivos de UNIX integrándolo en el núcleo, por lo que no se requiere recompilar librerías. El caché de clientes tiene el problema de las diferentes versiones de archivos, ya que la escritura de un cliente no se traduce en la inmediata modificación de copias del caché en los otros clientes. Las estampillas de tiempo son usadas por el NFS para validar bloques de caché.

Inconvenientes del NFS

Entre los inconvenientes que presenta NFS se pueden mencionar las siguientes:

- No es eficiente para sistemas distribuidos grandes.
- Las implementaciones caché inhiben a tener un pobre desempeño en las operaciones de escritura.

8.6.2 Sistema AFS (Andrew File Systems)

Este sistema de archivos distribuido fue desarrollado en Carnegie Mellon University (CMU) en 1986. El objetivo de AFS es compartir información a gran escala (10,000 estaciones de trabajo). Al igual que en Network File Systems (NFS), en AFS no hay que modificar programas de UNIX. La estrategia clave para alcanzar escalabilidad en AFS es a través del manejo de caché de archivos completos en los nodos clientes.

El mecanismo de AFS considera lo siguiente [Sánchez, 1995]:

1. Un proceso cliente realiza un operación "open" en un espacio compartido, entonces el servidor envía una copia del archivo al cliente.
2. El cliente realiza todas las operaciones en la copia local.
3. Una vez que el archivo es cerrado, se envía la versión modificada al servidor, mientras que el cliente mantiene la copia local.

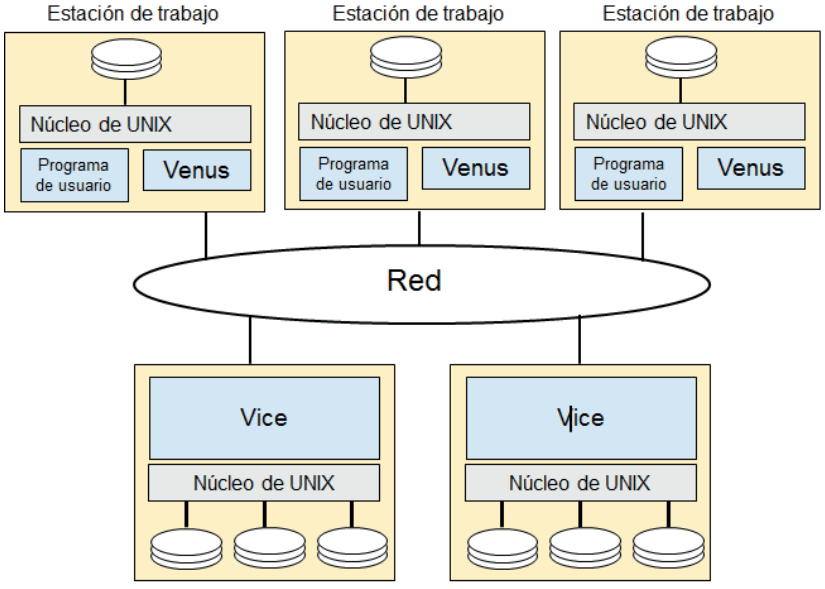
La estrategia de diseño de AFS está basado en UNIX, y considera las siguientes observaciones [Coulouris et al., 2001]:

1. Usar archivos pequeños.
2. Las lecturas se realizan con más frecuencia que las escrituras en una relación de seis a una.
3. El acceso a los archivos es secuencial.
4. Se comparten pocos archivos.
5. Generalmente un solo usuario es el responsable de modificar los archivos compartidos.

Entre las principales características del sistema AFS, se pueden indicar las siguientes [Coulouris et al., 2001]:

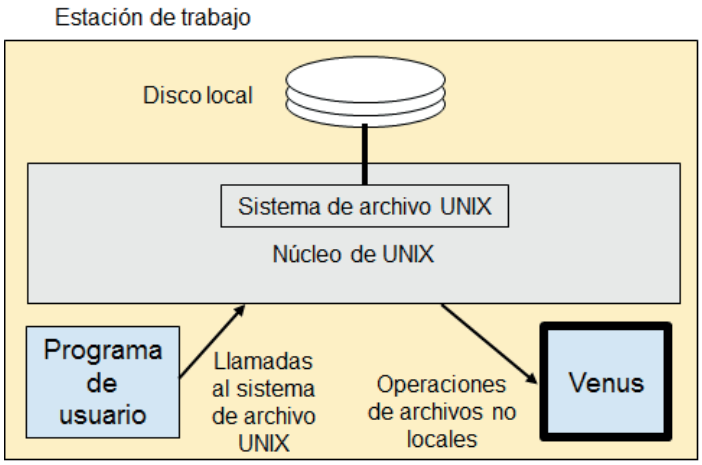
- Los cachés usados son del orden de 100 MB.
- El tamaño de los paquetes para la transferencia de archivos es de 64 KB.
- El uso de bases de datos no está considerado.
- Se usan directorios locales o compartidos para colocar los archivos.
- Está constituido principalmente por dos componentes de software a nivel usuario (ver figura 8.3):
  - Vice, en el servidor.
  - Venus, en el cliente.
- Cuando muchos clientes usan el AFS, se puede comparar favorablemente con respecto al NFS.
- Intercepta las llamadas a operaciones de archivos.
- Permite guardar información de estado sobre las copias de los archivos.
- Emplea réplicas de archivos, por lo que una nueva escritura deberá de ser hecha en todas las copias distribuidas.
- Soporta seguridad, protección y heterogeneidad.
- Los usuarios pueden cambiar de máquinas y usar el mismo nombre. Este proceso deberá ser transparente, excepto por algunas fallas de ejecución iniciales que son necesarias al poner los archivos de usuarios en las máquinas locales.

Figura 8.3. Arquitectura de distribución de procesos en AFS [Coulouris *et al.*, 2012]



AFS intercepta las llamadas al sistema de operaciones de archivos (ver figura 8.4) y los archivos forman grupos llamados volúmenes. Cada archivo y directorio en el sistema es identificado por un FID (File Identifier) único, que es usado para la comunicación entre Vice y Venus.

Figura 8.4. Intercepción de llamadas al sistema en AFS [Coulouris *et al.*, 2012]



8.7 MEMORIA COMPARTIDA DISTRIBUIDA

8.7.1 Generalidades

La *memoria compartida distribuida (DSM)* fue propuesto por Li [1986] como una solución en la cual un conjunto de máquinas conectadas en red comparten un solo espacio de direcciones virtuales con sus páginas. El desempeño es pobre, debido a la latencia y tráfico de la red como resultado de la localización de las páginas en la red.

Ejemplos de tipos de memoria compartida son:

- *Memoria en circuitos*: Varios procesadores comparten una memoria en un circuito.
- *Memoria usando un bus con o sin caché*: Diferentes CPU comparten una memoria que usa o no caché por medio de un bus.

- *Multiprocesadores basados en anillo*: Varios CPU comparten una parte de su espacio de direcciones usando un anillo.
- *Múltiples CPU con conmutador*: Permite incrementar la capacidad de comunicación variando la topología cuando existe una gran cantidad de procesadores.
- NUMA: Este acceso no uniforma a memoria, permite resaltar las diferencias entre un acceso local a uno remoto no ocultándolas con el uso de caché.
- *Basadas en páginas*: Aquí cada CPU tiene una memoria privada y no pueden realizar llamadas directas a memorias remotas.
- *Basadas en objeto*: Los programas deben de recorrer métodos protegidos para poder usar datos compartidos. Todo esto se realiza en software y permite mantener la consistencia.

8.7.2 Consistencia en la DSM

Adve y Hill [1990] definieron al modelo de consistencia como las reglas negociadas entre el software y la memoria compartida distribuida. Ejemplos de modelos de consistencia usados en la DSM son [Tanenbaum, 1996]:

- *Consistencia estricta*: Aquí toda lectura a una localidad de memoria *x* devolverá el valor registrado por la última escritura en *x*.
- *Consistencia secuencial*: Define que el resultado de cualquier ejecución es igual que si las operaciones de todos los procesos fueran realizadas de manera secuencial y las operaciones de cada proceso aparecen en esta secuencia en el orden indicado en el programa.
- *Consistencia casual*: Aquí todas las escrituras potenciales relacionadas en forma casual son vistas en ese orden por todos los procesos. Escrituras concurrentes podrán ser vistas en diferente orden desde diferentes máquinas.
- *Consistencia PRAM*: Define que las escrituras de un proceso se reciben por otros procesos en el orden realizado, sin embargo, las escrituras de procesos diferentes se pueden ver en orden diferentes desde distintas computadoras.
- *Consistencia débil*: El acceso a las variables de sincronización son de consistencia secuencial, no se permite este acceso hasta que se terminen las escrituras anteriores en todos los sitios ni se permite un acceso a datos sin haber terminado todos los accesos anteriores a las variables de sincronización.

- *Consistencia de liberación*: Se debe de terminar con éxito todas las adquisiciones anteriores antes de realizar un acceso normal a una variable compartida, así como terminar lecturas y escrituras anteriores del proceso antes de liberar, ambas deben ser consistentes.
- *Consistencia de entrada*: Define que un proceso que realiza una adquisición no podrá concluir hasta no actualizar todas las variables compartidas protegidas, para lo que entrará en una región crítica en modo exclusivo.

8.7.3 DSM basada en paginación

En este modelo ningún procesador tiene acceso directo a la memoria de otro procesador, por lo que estos sistemas también se dominan sin acceso a memoria remota (NORMA) [Tanenbaum, 1996]. La idea en este modelo se centra en emular el caché de un multiprocesador a través de un administrador de memoria y el sistema operativo para relacionar los pedazos de memoria. La opción de réplica ayuda a mejorar el desempeño, duplicando los pedazos exclusivos para lectura, por lo que un aspecto importante del diseño es el tamaño de estos pedazos, que puede ser una palabra, un bloque, una página o un segmento.

Un problema a considerar en este modelo es que para transferencias grandes de datos la red tiende a bloquearse por más tiempo, pero también se podría presentarse la compartición de manera falsa. Otro problema se presenta con la consistencia secuencial cuando existen réplicas de página de lectura-escritura para la página local y la página remota. Para este problema de consistencia se presentan dos soluciones:

- *Actualización*: Permite la escritura local y actualiza de manera simultánea todos los cachés.
- *Invalidación*: Aquí se transmite por el bus la dirección de la palabra a actualizar, pero no la nueva palabra, por lo que cuando un caché ve que una de sus palabras está siendo actualizada, invalida el bloque que contiene la palabra, eliminándola del caché.

Por lo general, los sistemas DSM basados en paginación adoptan la solución por invalidación. Para la búsqueda de copias en estos DSM se proponen dos posibilidades:

- Transmitir el número de página solicitando su invalidación.
- Usar una lista de conjunto de páginas.

Una DSM puede requerir reemplazar páginas cuando no existan marcos libres, para lo que podrán usar los algoritmos tradicionales, como LRU (*Least Recently Used*, El más reciente usado). Una página duplicada es por lo general la elegida a retirar de la memoria o una página duplicada de un proceso saliente. En caso de que ninguna página duplicada pueda ser elegida, se podrá retirar la página no duplicada de menor uso. La sincronización es otro punto a observar en estos sistemas, para lo cual se prevé la exclusión mutua apoyada con algún controlador de sincronización.

EJERCICIOS

1. Describe tres funciones del módulo cliente en un sistema de archivos.

2. Describe las funciones de los módulos "Venus" y "Vice" en el sistema AFS.

3. Cita cinco operaciones que son posibles realizar en el sistema NFS.

4. ¿Cuál es la diferencia entre un sistema de archivos y una base de datos?

5. ¿Cuál es el principal beneficio de la memoria distribuida?

6. ¿Cuáles son las principales desventajas del sistema NFS?

7. Cita tres características de UNIX que toma AFS en su estrategia.

8. Indica las principales diferencias entre NFS y AFS.

9. Investiga sobre los sistemas de archivos posteriores a NFS y AFS.

10. ¿Por qué es importante la consistencia en la memoria compartida distribuida?

11. Cita tres ejemplos de uso de memoria compartida.

Actividad integradora

1. De la aplicación de los contenidos, así como de las habilidades y actitudes desarrolladas.
   Se recomienda exposición general del tema por parte del profesor, así como la realización de algunas prácticas sobre operaciones en archivos distribuidos por parte de los alumnos en el laboratorio. Se puede usar para tal fin el sistema NFS que tienen los sistemas operativos Linux o Unix. El contenido de este capítulo podría ser cubierto con una sesión teórica y dos sesiones prácticas en el laboratorio, realizando manipulación de archivos en un ambiente distribuido.
   El alumno debe mostrar un gran interés por programar y explorar el uso de comandos preexistentes en un sistema de archivos distribuidos.
2. Del logro de los objetivos establecidos en el capítulo o apartado del material.
   Se espera alcanzar los objetivos de este capítulo con el apoyo en la solución de los ejercicios y la exposición del profesor sobre el tema, complementadas con las prácticas de laboratorio correspondiente.

Capítulo 9. Replicación, consistencia y tolerancia a fallas

Objetivo: Que el alumno analice y comprenda los diferentes esquemas relacionados a la replicación, consistencia y tolerancia a fallos, así como su impacto en el diseño de sistemas distribuidos.

9.1. Introducción

La replicación juega un importante rol en los sistemas distribuidos y, por lo general, se utiliza para incrementar la confiabilidad y mejorar el rendimiento de un sistema. Sin embargo, uno de los principales retos en los sistemas distribuidos es hacer que estas replicas se mantengan consistentes, lo que implica garantizar que todas las copias del sistema sean actualizadas.

Un reto importante en la consistencia es con respecto a los datos compartidos, que son accedidos por varios procesos simultáneamente, ya que implementar la consistencia crece en complejidad conforme el sistema distribuido es escalado. En este escenario, dos cuestiones relacionados a la consistencia deben ser considerados [Tanenbaum & Van Steen, 2008]. La primera está relacionada con la administración de la réplica, donde se consideran aspectos como la ubicación de los servidores de réplicas y distribución del contenido entre estos servidores. La segunda cuestión a considerar es sobre el mantenimiento de consistencia de las réplicas, lo cual implica que las actualizaciones de todas las réplicas se deben de realizar de una manera rápida (casi inmediata). Por otro lado, también los sistemas distribuidos son diseñados de manera que puedan recuperarse automáticamente de fallas parciales, sin que se afecte seriamente el desempeño total. Esto es una característica sobresaliente de los sistemas distribuidos que los distingue de los sistemas de una sola máquina o centralizados. Una falla se puede

presentar en un sistema distribuido y afectar el funcionamiento de algunos componentes, sin embargo, el sistema puede seguir operando. Esto no es posible en un sistema no distribuido.

Este capítulo revisa los temas relacionados a replicación, consistencia y tolerancia a fallas en los sistemas distribuidos.

9.2 REPLICACIÓN

La replicación significa mantener copias de una información en múltiples computadoras. Este recurso es ampliamente usado en los sistemas distribuidos, debido a que proporciona un mejor rendimiento, alta disponibilidad y tolerancia a fallas. Algunos ejemplos donde se usa la replicación son [Coulouris *et al.*, 2012]:

- Almacenamiento en caché en los servidores web y servidores proxy web.
- El servicio de nombres DNS.

9.2.1 BENEFICIOS DE USAR REPLICACIÓN EN LOS SISTEMAS DISTRIBUIDOS

9.2.1.1 Mejoras del rendimiento

Entre las características con respecto al rendimiento del sistema distribuido que deben de ser observadas al replicar datos se pueden mencionar las siguientes [Coulouris *et al.*, 2012]:

- La replicación se implementa principalmente a través de cachés en clientes o servidores.
- Es importante mantener copias de los resultados obtenidos en llamadas anteriores al servicio para evitar o reducir el coste de llamadas idénticas.
- Se evita el tiempo de latencia derivado del cálculo del resultado o de realizar consultas a otros servidores.
- La replicación de datos con actualizaciones muy frecuentes en la red genera un costo en el uso de protocolos de intercambio y actualización, además de limitar la efectividad de la réplica.

9.2.1.2 Alta disponibilidad

Entre las características con respecto a la disponibilidad del sistema distribuido que deben de ser observadas al replicar datos se pueden mencionar las siguientes [Coulouris *et al.*, 2012]:

- La proporción de tiempo que un servicio está accesible con tiempos de respuesta razonables, que debe de ser cercana al 100%.
- Existen pérdidas de disponibilidad debido a los siguientes factores:
  - Fallas en el servidor. Si se replica *n* veces el servidor, la disponibilidad será $1\text{-}p^n$. Por ejemplo, para $n = 2$ servidores, la disponibilidad es $1 - 0.05^2 = 1 - 0.0025 = 99.75\%$.
  - Particiones de red o desconexiones.
  - Operación desconectada (son aquellas desconexiones de comunicación que a menudo no se planifican y son un efecto secundario de la movilidad del usuario).

9.2.1.3 Tolerancia a fallas

Entre las características respecto a la tolerancia a falla en los sistemas distribuidos, se pueden mencionar las siguientes [Coulouris *et al.*, 2012]:

- Una alta disponibilidad no implica necesariamente corrección, esto se debe a que puede haber datos no actualizados, o inconsistentes, originados por procesos concurrentes.
- Se puede utilizar la replicación para alcanzar una mayor tolerancia a fallas.
  - Ante una caída o suspensión de servidores; por ejemplo, si se tienen *n* servidores, pueden fallar *n-1* sin que el servicio sufra alteración.
  - Ante fallas bizantinas, es decir, si *f* servidores presentan fallas bizantinas, un sistema con *2f+1* servidores proveería un servicio correcto.

9.2.2 Requisitos para realizar la replicación

Entre los requisitos más importantes que se deben considerar al implementar la replicación en un sistema distribuido, están:

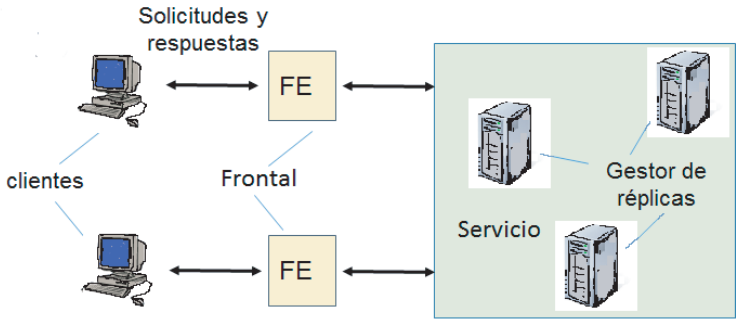
- Transparencia.
- Consistencia.

El requisito de transparencia implica que los clientes no son conscientes de que hay múltiples copias del recurso al que ellos están accediendo. Para los usuarios, solo existen recursos lógicos individuales. Por otro lado, la consistencia implica que las operaciones sobre un conjunto de objetos replicados deben de dar resultados que sigan la especificación de corrección definida para esos objetos. De acuerdo con el tipo de aplicación, la consistencia puede ser más o menos estricta.

9.2.3 Modelo general de gestión de réplica

Esta sección presenta un modelo de sistema general para la gestión de réplicas. Además, describe el papel de los sistemas de comunicación grupal en la consecución de la tolerancia a fallas a través de la replicación. El modelo general de gestión de réplica se muestra en la figura 9.1.

Figura 9.1. Modelo de arquitectura básica para la gestión de los datos replicados



Los componentes principales de esta arquitectura de gestión de datos replicados son:

- Gestor de réplicas.
- Frontal (front-end).

El gestor de réplicas son los componentes que almacenan las réplicas de un determinado objeto o servicio y operan sobre ellas. Frontal (FE) es el componente o interfaz que atiende las llamadas de los clientes y se comunica con los gestores de réplicas.

En general, cinco fases están involucradas en el desempeño de una única solicitud a los objetos replicados [Wiesmann, Pedone, Schiper, Kemme & Alonso, 2000]:

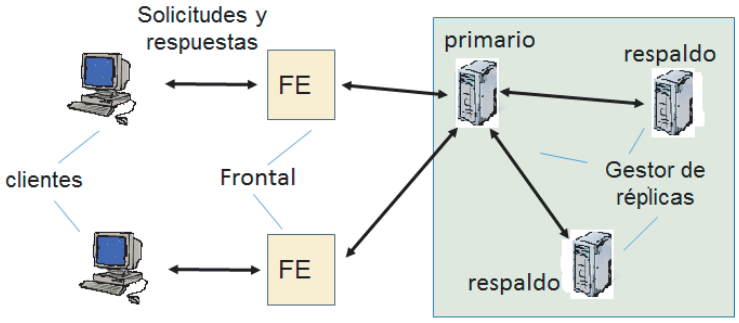
1. La *petición* es enviada por el frontal a uno o más gestores con las siguientes opciones:
   - Se envía la petición a un gestor y este la reenvía a otros.
   - Multidifundir la petición a varios gestores.
2. Los gestores se *coordinan* para ejecutar la petición de manera consistente. Los tipos de ordenación para que los gestores manipulen las réplicas son:
   - Ordenamiento FIFO.
   - Ordenamiento casual.
   - Ordenamiento total.
3. Se *ejecuta* la petición, la cual podría ser realizada de manera tentativa.
4. Se llega a un *acuerdo* o *consenso* antes de consumar la ejecución.
5. Uno o más gestores de réplicas entrega una *respuesta* al frontal.

9.2.4 Servicios de tolerancia a fallas basados en replicación

9.2.4.1 Replicación pasiva

En la replicación pasiva existe un gestor de réplicas primario y uno o más gestores secundarios también conocidos como "respaldos" o "esclavos". Los frontales se comunican con el gestor primario, quien ejecuta las operaciones y manda copias a los respaldos. Si el gestor de réplicas primario falla, entonces un gestor de réplicas de respaldo lo sustituye. La figura 9.2 muestra este escenario.

Figura 9.2. Modelo de replicación pasiva para tolerancia a fallas



Una replicación pasiva tiene las siguientes fases de ejecución [Coulouris *et al.*, 2012]:

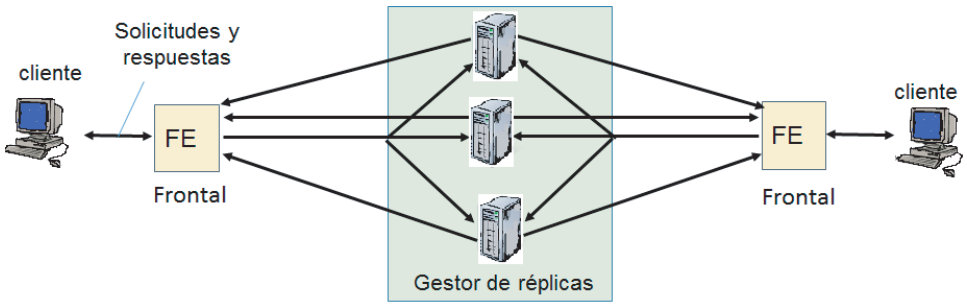
1. Petición.
2. Coordinación.
3. Ejecución.
4. Acuerdo.
5. Respuesta.

Durante la fase de petición, el frontal envía la petición al gestor primario, quien activa la fase de coordinación y ejecuta las peticiones siguiendo una ordenación FIFO. Se realiza la fase de ejecución de la petición y se almacena la respuesta. Si es una petición de actualización, entonces en la fase de acuerdo el gestor primario envía la actualización a todos los respaldos, quienes confirman la recepción. Finalmente, en la fase de respuesta, el gestor primario envía la respuesta correspondiente al frontal. La replicación pasiva tolera fallas de procesos pero no tolera fallas bizantinas. El frontal requiere poca funcionalidad, sin embargo, este esquema de replicación es propenso a problemas de cuellos de botella en el gestor primario.

9.2.4.2. Replicación activa

En la replicación activa todos los gestores de réplicas tienen el mismo rol. Los frontales multidifunden las peticiones a todos los gestores y todos los frontales procesan la petición de manera idéntica pero independiente. En la figura 9.3 se muestra un escenario de replicación activa.

Figura 9.3. Modelo de replicación activa para tolerancia a fallas



Al igual que la replicación pasiva, una replicación activa tiene las siguientes fases de ejecución [Coulouris *et al.*, 2012]:

1. Petición.
2. Coordinación.
3. Ejecución.
4. Acuerdo.
5. Respuesta.

Sin embargo, las actividades que se realizan en cada fase difieren sustancialmente de las actividades que se hacen en la replicación pasiva. Durante la fase de petición, el frontal multidifunde la petición a los gestores usando multidifusión fiable y de ordenación total. No se envía otra petición hasta que se reciba la respuesta a la petición actual. El sistema de comunicación durante la fase de coordinación entrega la petición a todos los gestores según una ordenación total. Entonces, cada gestor realiza la fase para ejecutar la petición. La fase de acuerdo no es necesaria, debido a que se utiliza multidifusión. Finalmente, en la fase de respuesta, cada gestor manda su respuesta al frontal. El número de respuesta que recibe el frontal está en función de las asunciones de falla y del algoritmo de multidifusión.

En la replicación activa, la tolerancia a fallas está soportada principalmente por la multidifusión fiable y totalmente ordenada. La multidifusión es equivalente a un algoritmo de consenso, por lo que la replicación activa permite resolver fallas bizantinas. Esto se debe a que el frontal recoge *f+1* respuestas iguales antes de responder al cliente.

9.3 Consistencia

La consistencia en los sistemas de cómputo es una necesidad vital, ya que sin ella el sistema simplemente no funciona. El problema de la consistencia está presente tanto en los sistemas de cómputo centralizados como en los sistemas distribuidos. El acceso a los datos en un sistema centralizado puede caer en estado inconsistente ante accesos concurrentes, por lo que se requiere prever mecanismos de exclusión que eviten estos escenarios. Sin embargo, en los sistemas distribuidos el problema de la inconsistencia es de mayor dimensión, tanto por su importancia como por la gran cantidad de situaciones en que puede producirse. Debido a que un sistema distribuido es un único sistema, entonces debe tener un único estado global compartido por todas las computadoras que la componen. Este estado global comprende las tablas de mantenimiento del sistema, la hora actual o los datos que están siendo compartidos por distintas computadoras.

9.3.1 Tipos de inconsistencias

En un sistema distribuido, los diferentes tipos de consistencias más frecuentemente estudiadas son:

- Consistencia de actualización.
- Consistencia de replicación.
- Consistencia de caché.
- Consistencia de reloj.

9.3.1.1 Consistencia de actualización

La consistencia de actualización es importante observarla cuando varios procesos acceden concurrentemente a un dato para actualizarlo, ya que la actualización de todo el dato en su conjunto no se realiza como una única operación atómica y se puede caer en una inconsistencia de actualización. Este tipo de inconsistencia es un problema clásico en el acceso a bases de datos o datos compartidos. Sin embargo, en los sistemas distribuidos esto tiene una mayor relevancia debido a que estos sistemas generalmente tienen un gran número de usuarios. Este tipo de inconsistencia se evita usando *transacciones*.

Las transacciones son las primitivas equivalentes a las entradas y salidas de una región crítica usada para proteger la consistencia de un dato compartido. Asimismo, una transacción asegura que las operaciones incluidas en esta se ejecuten todas o no se ejecuta una sola.

9.3.1.2 Consistencia de replicación

Una situación de inconsistencia de replicación puede producirse cuando un conjunto de datos debe replicarse en diversas computadoras de la red, pudiendo ser modificado por cualquiera de estas. En este escenario, es muy probable que se puedan producir situaciones en que los datos no sean iguales en todas las computadoras al mismo tiempo. Los juegos interactivos multiusuarios pueden ser un ejemplo de inconsistencia de replicación, pues las acciones que introduzca un jugador deben de propagarse usando un protocolo de difusión de manera inmediata al resto de los jugadores en la red. En caso de que la red falle o tenga alta latencia, cada jugador tendrá una visión distinta del juego. La consistencia de replicación tiene gran importancia en

los sistemas distribuidos, más aun si estos son en ambientes colaborativos e interactivos.

9.3.1.3 Consistencia de caché

Cuando una aplicación cliente accede a archivo de datos, se pueden guardar estos datos en un espacio de la memoria local del lado del cliente para facilitar el acceso a este dato en futuras referencias. Esto se conoce como memoria caché y reduce la transferencia de datos por la red. La memoria caché tiene como objetivo mejorar los accesos locales mediante el modelo de jerarquías de memoria [Patterson & Hennessy, 1994]. Sin embargo, el problema de inconsistencia se puede presentar cuando el cliente también tiene que actualizar el mismo dato pero este reside en las memorias cachés de otros clientes. En este caso, existe el riesgo de que las copias del dato en las otras memorias cachés queden desactualizadas. Para evitar este escenario, se deben considerar técnicas que aseguren la consistencia de los cachés por parte de los sistemas operativos distribuidos o por las arquitecturas de sistemas multiprocesadores.

9.3.1.4 Consistencia de reloj

Diversas aplicaciones y programación en ambientes distribuidos basan su funcionalidad en algoritmos que muchas veces dependen de estampillas de tiempo. Estas estampillas de tiempo (ver capítulo 5. Sincronización) son usadas para indicar el momento en que un evento ha sucedido. Por ejemplo, algunos algoritmos de reemplazo de página en memoria virtual hacen uso de estampillas de tiempo. Una computadora en un sistema distribuido puede generar una estampilla de tiempo, la cual se puede enviar a cualquier otra computadora del sistema. Así, cada computadora puede comparar su estampilla de tiempo local con la estampilla de tiempo recibida. El problema de inconsistencia de reloj se presenta debido a que no es fácil mantener la misma hora física en todas las computadoras del sistema de manera simultánea.

9.3.2 Modelos de consistencia

Los modelos de consistencia para los datos compartidos son a menudo difíciles de implementar de manera eficiente en los sistemas distribuidos a gran escala [Tanenbaum & Van Steen, 2008]. Además, en muchos casos se pueden

utilizar modelos más simples, que también son a menudo más fáciles de implementar. Un modelo de consistencia es un contrato en los procesos y el almacenamiento de datos. Es decir, si los procesos acuerdan obedecer ciertas reglas, entonces el almacenamiento promete trabajar correctamente. Normalmente, una operación de lectura debe retornar la última actualización del dato. Los modelos de consistencia pueden ser:

- Centrados en los datos.
- Centrados en el cliente.

9.3.2.1 Modelo de consistencia centrado en los datos

Este modelo asume que un almacén de datos (base de datos distribuida o un sistema de archivos) puede estar físicamente distribuido en varias máquinas. Todo proceso que pueda acceder a datos del almacén tiene una copia local disponible de todo el almacén y todas las operaciones de escritura se propagan hacia las otras copias. Ante la ausencia de un reloj global, es difícil sincronizar y determinar cuál es la última operación de escritura. Lo anterior permite que una consistencia centrada en datos presente una gama de modelos de consistencia, entre los que se pueden mencionar los siguientes:

- Modelos de consistencia que no usan variables de sincronización:
  - Estricta.
  - Linealizada.
  - Secuencial.
  - Causal.
  - FIFO.

- Modelos de consistencia con operaciones de sincronización:
  - Débil.
  - Relajada.
  - Entry.

9.3.2.2 Modelo de consistencia centrado en el cliente

Este modelo de consistencia está orientado a un conjunto de datos que tienen un bajo número de actualizaciones simultáneas o, cuando estas ocurren, pueden resolverse fácilmente. Generalmente está orientado a operaciones

de lectura de datos, por lo que se puede catalogar como un modelo de consistencia bastante débil [Tanenbaum & Van Steen, 2008]. Los modelos de consistencias basados en el cliente permiten ver que es posible ocultar muchas inconsistencias de un sistema distribuido de una manera relativamente fácil.

Tanenbaum y Van Steen [2008] citan como ejemplos de modelos de consistencia centrados en el cliente los siguientes:

- Consistencia momentánea.
- Lecturas monotónicas.
- Escrituras monotónicas.
- Lea sus escrituras.
- Las escrituras siguen a las lecturas.

9.4. Tolerancia a fallas

Los sistemas distribuidos se distinguen principalmente de los sistemas centralizados de una sola computadora en su capacidad de falla parcial. La tolerancia a fallas ha sido un tema de investigación por muchos años en las ciencias de la computación. Un sistema tolerante a fallas es un sistema que posee la capacidad interna para asegurar la ejecución correcta y continuada a pesar de las presencia de fallas en hardware o software. El objetivo de este tipo de sistemas es ser altamente fiable.

9.4.1 Origen de una falla

El origen de las fallas en un sistema puede ser clasificado como:

- Fallas en hardware:
  - Son generalmente fallas permanentes o transitorias en los componentes del hardware.
  - Fallas permanentes o transitorias en los subsistemas de comunicación.

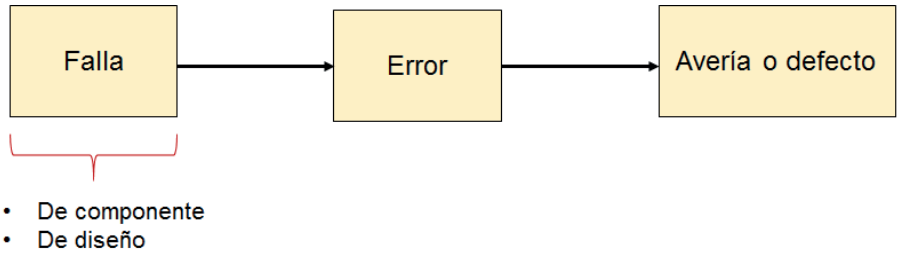
- Fallas en software:
  - Se originan por especificaciones inadecuadas.
  - Fallas introducidas por errores en el diseño y programación de componentes de software.

Se dice que un sistema distribuido es un sistema fiable cuando comprende varios requerimientos útiles, como los siguientes:

- Disponibilidad.
- Confiabilidad.
- Seguridad.
- Mantenimiento.

La *fiabilidad* de un sistema es una medida de su conformidad con una especificación autorizada de su comportamiento [Tanenbaum & Van Steen, 2008]. En contraste, una *avería* o *defecto* es una desviación del comportamiento de un sistema respecto a esta especificación. Las averías se manifiestan en el comportamiento externo del sistema pero son el resultado de *errores* internos. Las fallas pueden ser consecuencia de averías en los componentes del sistema. Las fallas pueden ser pequeñas pero los defectos muy grandes. En la figura 9.4 se muestran las relaciones causa-efecto de una falla.

Figura 9.4. Relación causa-efecto de una falla



9.4.2. Clasificación de fallas

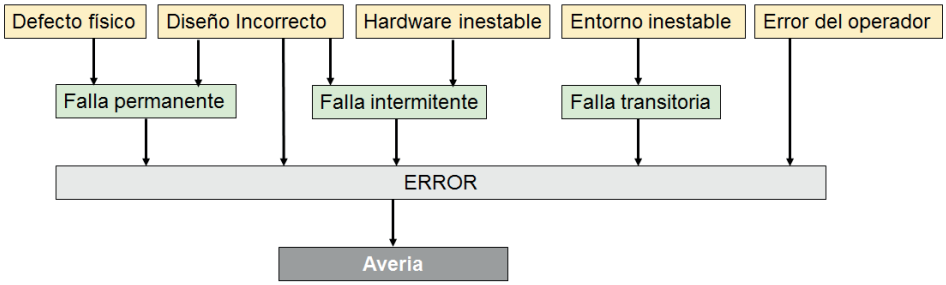
El objetivo de la tolerancia a fallas en un sistema distribuido es evitar que las fallas produzcan averías. Las fallas se clasifican generalmente como (ver figura 9.5):

- Transitorias.
- Intermitentes.
- Permanentes.

Es una falla transitoria, si esta desaparece sola al cabo de cierto tiempo. Ejemplos de este tipo de falla son las interferencias en las señales de comunicación o las fallas transitorias en los enlaces de comunicación. Por otro

lado, las fallas permanentes son aquellas fallas que permanecen hasta que el componente se repare o sustituya, por ejemplo, los errores de software o la rotura de una tarjeta de video. Finalmente, las fallas intermitentes se refieren a aquellas fallas transitorias que ocurren de vez en cuando, por ejemplo, el calentamiento de algún componente de la computadora.

Figura 9.5. Clasificación de fallas



9.4.3 Fallas en los procesos distribuidos

En un sistema distribuido, tanto los procesos como los canales de comunicación pueden fallar, es decir que pueden apartarse de lo que se considera el comportamiento correcto o deseable. En la figura 9.6 se ilustra un modelado de procesos y canales. El modelo de fallas define las formas en las que puede ocurrir una falla, con el fin de proporcionar una comprensión de los efectos de los fallas. Bajo este enfoque, Hadzilacos y Toueg [1994] proporcionaron una taxonomía que distingue entre las fallas de los procesos y canales de comunicación y los presenta como fallas por omisión, fallas arbitrarias y fallas temporales. La tabla 9.1 resume las principales características de este tipos de fallas.

Figura 9.6. Procesos y canales

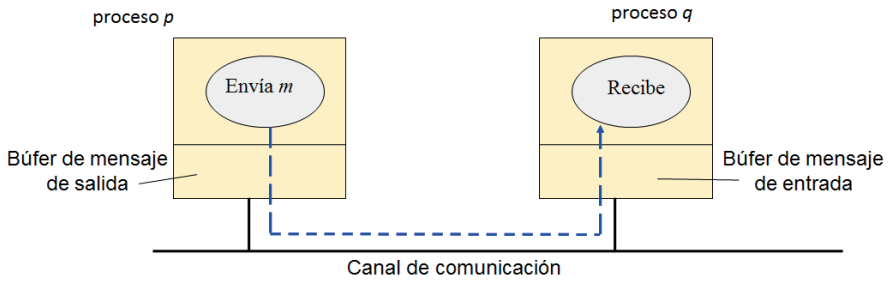


Tabla 9.1. Fallas por omisión y arbitrarias [Coulouris et al, 2012]

Tipo de falla	Efecto	Descripción
Falla de congelación	Proceso	El proceso para y permanece así. Otros procesos pueden detectar este estado
Crash (caída)	Proceso	El proceso para y permanece así. Otros procesos pueden no ser capaces de detectar este estado
Omisión	Canal	Un mensaje insertado en un buffer de mensajes de salida no llega al siguiente buffer de llegada de mensajes
Omisión de envío	Proceso	Un proceso completa un envío pero el mensaje no es puesto en su buffer de mensajes de salida
Omisión de recepción	Proceso	Un mensaje es puesto en el buffer de mensajes de llegada de un proceso pero este no lo recibe
Arbitraria (Bizantino)	Proceso o canal	El proceso/canal muestra un comportamiento arbitrario: podría enviar/transmitir arbitrariamente mensajes en tiempos arbitrarios, comete omisiones; un proceso puede detenerse o tomar un paso incorrecto

Las fallas más serias son las arbitrarias o bizantinas. Cuando estas fallas ocurren, los clientes deben de estar preparados para lo peor. Este problema suele conocerse en la literatura como *el problema de los Generales Bizantinos*, que fue planteado originalmente por Lamport *et al*. [1982].

9.4.4 Redundancia

Todas las técnicas de tolerancia a fallas se basan en el uso de la *redundancia*, de la cual pueden ser identificados los siguientes tipos:

- *Redundancia estática*: Los componentes redundantes se utilizan dentro del sistema para enmascarar los efectos de los componentes con defectos.
- *Redundancia dinámica*: La redundancia se utiliza solo para la detección de errores. La recuperación debe de realizarla otro componente.
- *Redundancia en el diseño*: Partes de un diseño pueden ser redundantes.
- *Redundancia temporal*: Mediante el uso repetido de un componente en presencia de fallas. Adecuado para fallas transitorias.

Ejercicios

1. Define el concepto de tolerancia a fallas en los sistemas distribuidos.

2. ¿Cuál es la importancia del problema de los generales bizantinos para la tolerancia a fallas?

3. Investiga sobre los esquemas de redundancia más usuales para soportar la tolerancia a fallas en los sistemas distribuidos.

4. Cita tres ejemplos de fallas transitorias, intermitentes y permanentes.

5. Explica las ventajas y desventajas de usar una replicación en los sistemas interactivos multiusuarios desplegados globalmente.

6. Explica qué es un modelo de consistencia.

7. Explica la diferencia entre la replicación pasiva y activa.

8. ¿Cuáles son los retos a que se enfrenta la consistencia de caché en los sistemas distribuidos?

9. Explica la relación entre transparencia, consistencia y replicación en los sistemas distribuidos.

10. ¿Qué requerimientos debe de cumplir un sistema distribuido para considerarlo fiable?

Actividad integradora

1. De la aplicación de los contenidos, así como de las habilidades y actitudes desarrolladas.
   Se recomienda exposición general del tema por parte del profesor, así como un análisis de los esquemas de replicación y tolerancia a fallas más usados en los sistemas distribuidos, además de su impacto en tipos de consistencia específicos. Se recomienda la discusión en grupo con alta participación de los alumnos para identificar las posibles soluciones que ofrece cada esquema de replicación y tolerancia a fallas, así como los

retos para implementarlas en la práctica. Este tema se puede cubrir en una semana de tres sesiones.

El alumno debe de mostrar un gran interés por la observación, análisis, abstracción e interpretación de esquemas de flujo de datos, necesidades de replicación y costos de comunicación.

2. Del logro de los objetivos establecidos en el capítulo o apartado del material.

   Se espera alcanzar los objetivos de este capítulo con el apoyo en la solución de los ejercicios, la exposición del profesor sobre el tema, así como un taller de análisis de diagramas sobre soluciones de replicación y tolerancia a fallas.

Capítulo 10. Seguridad

Objetivo: Que el alumno tenga una visión general sobre la importancia de la seguridad en los sistemas distribuidos desde una perspectiva del encriptado y la autenticación de acceso.

10.1 Introducción

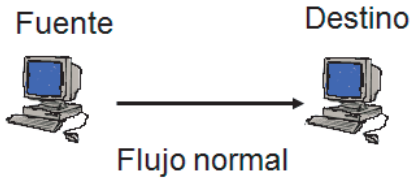
Seguridad se refiere en general a un conjunto de medidas de procedimiento, tanto lógicas como físicas, orientadas a la prevención, detección y corrección de casos de mal uso. Desde un enfoque informático, la seguridad se refiere a las características que debe de tener un sistema de cómputo para resistir ataques y mal uso, ya sea este intencional o no. En los sistemas distribuidos, la seguridad de la información juega un rol muy importante, ya que se debe de garantizar que los recursos de cómputo y la información estén protegidos. Para un enfoque de seguridad informática se recomienda observar los siguientes aspectos [Menezes, Van Oorschot & Vanstone, 1996]:

- *Integridad*: Solo los usuarios autorizados podrán modificar la información.
- *Confidencialidad*: Solo los usuarios autorizados tendrán acceso a los recursos y a la información que utilicen.
- *Disponibilidad*: La información debe de estar disponible cuando se necesite.
- *Vinculación (no repudio)*: El usuario no puede refutar o negar una operación realizada.

10.2 ATAQUES A LA SEGURIDAD

Los ataques a la seguridad en una red de comunicaciones se pueden caracterizar modelando el sistema como un flujo de información desde una fuente (usuario o archivo) a un destino (otro archivo o usuario), tal como se muestra en la figura 10.1 [Stalling, 2004].

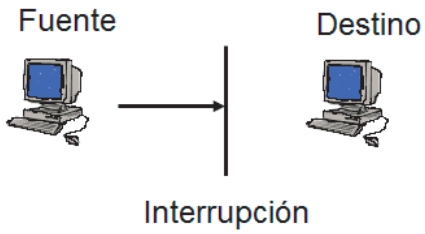
Figura 10.1. Flujo de información normal entre una fuente y un destino



Los cuatro tipos genéricos de ataques a la seguridad son los siguientes [Stalling, 2004]:

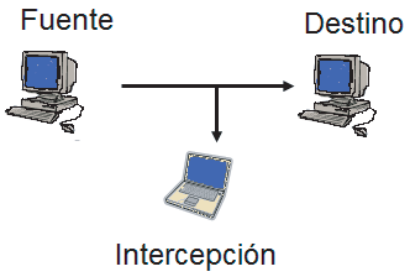
- *Interrupción*: Una parte del sistema resulta destruida o no disponible en un momento dado. Ejemplos de este tipo de ataque pueden ser la destrucción de una parte del hardware o el corte de una línea de comunicación.

Figura 10.2. Ataque del tipo interrupción entre una fuente y un destino



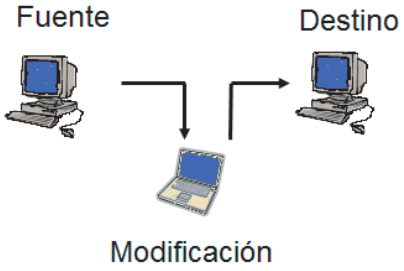
- *Intercepción*: Una entidad no autorizada accede a una parte de la información. Esta entidad no autorizada puede ser una persona, una computadora o un programa. Ejemplos de este tipo de ataques son la escucha del canal, la intercepción vía radio de comunicaciones móviles o la copia ilícita de archivos o programas transmitidos a través de la red.

Figura 10.3. Ataque del tipo intercepción entre una fuente y un destino



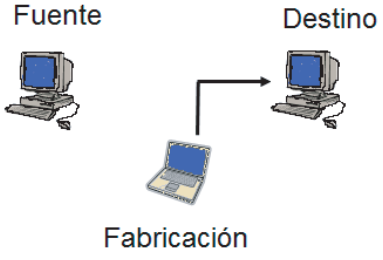
- *Modificación*: Una entidad no autorizada puede acceder a una parte de la información y, además, es capaz de modificar su contenido. Ejemplos de estas modificaciones son la alteración de archivos de datos, alteración de programas y la modificación de mensajes mientras estos son transmitidos por la red.

Figura 10.4. Ataque del tipo modificación entre una fuente y un destino



- *Fabricación*: Una entidad no autorizada envía mensajes y se hace pasar por un usuario legítimo.

Figura 10.5. Ataque del tipo fabricación entre una fuente y un destino



Otra clasificación usada es la división de los ataques en términos de *pasivos* y *activos*. En los *ataques pasivos*, el atacante no altera la comunicación o información, solo escucha o monitorea la red para obtener la información que se está transmitiendo. Entre los objetivos de un ataque pasivo se encuentra la intercepción de datos y el análisis de tráfico, que puede ser usada para obtener información de la comunicación. Los ataques pasivos son muy difíciles de detectar, ya que no provocan alteración en los datos. La información se puede proteger de este tipo de ataque usando encriptado y otros mecanismos de protección.

En los *ataques activos*, el atacante altera las comunicaciones. Este tipo de ataque puede subdividirse en las siguientes categorías:

- Suplantación de identidad.
- Reactuación.
- Modificación de mensajes.
- Degradación fraudulenta del servicio.

Durante la suplantación de identidad, el intruso simula ser una entidad diferente, mientras que en la reactuación uno o varios mensajes legítimos son capturados y repetidos para producir un efecto no autorizado. En el ataque de modificación de mensajes, el intruso varía los datos transmitidos. Finalmente, en un ataque de degradación fraudulenta del servicio, el intruso intenta impedir que las entidades dialogantes puedan funcionar de manera correcta.

10.3 SERVICIOS DE SEGURIDAD

Para afrontar las amenazas a la seguridad del sistema de una organización, se requiere definir una serie de servicios. Estos servicios hacen uso de uno o más mecanismos de seguridad. Una clasificación de los servicios de seguridad es la siguiente [Stalling, 2004]:

- *Confidencialidad*: Garantiza que la información es accesible únicamente por las entidades o personas autorizadas.
- *Autenticación*: Ofrece un mecanismo que permite una identificación correcta del origen del mensaje, asegurando que la entidad no es falsa.
- *Integridad*: Garantiza la corrección y completitud de la información. Es decir que la información solo pueda ser modificada por las entidades

autorizadas. Una modificación puede consistir en escritura, cambio, borrado, creación y reactuación de los mensajes transmitidos.
- *Vinculación*: Vincula un documento a una transacción, a una persona o a un sistema de gestión de información, de tal manera que ni el emisor ni el receptor puedan negar la transmisión.
- *Control de acceso*: Requiere que el acceso a la información sea controlado por el sistema destino.

10.4 MECANISMOS DE SEGURIDAD

No existe un único mecanismo que provea todos los servicios de seguridad antes descritos. Sin embargo, la mayoría de los mecanismos de seguridad usan técnicas criptográficas basadas en el cifrado de los datos. Las técnicas criptográficas más importantes son [Stalling, 2004]:

- *Intercambio de autenticación*: Se encarga de comprobar que la entidad, ya sea origen o destino de la información, sea la indicada.
- *Encriptado*: Altera la representación de los mensajes de manera que garantiza que el mensaje es ininteligible a receptores no autorizados.
- *Integridad de datos*: Garantiza la corrección y completitud de la información. Cifra una cadena comprimida de datos a transmitir y envía este mensaje al receptor junto con los datos ordinarios. El receptor repite la compresión y el cifrado posterior de los datos y compara el resultado obtenido con el recibido para verificar que los datos no han sido modificados.
- *Firma digital*: Cifra una cadena comprimida de datos que se va a transferir usando la clave secreta del emisor. La firma digital se envía al receptor junto con los datos ordinarios. Este mensaje se procesa en el receptor para verificar su integridad.
- *Control de acceso*: Permite que solo aquellos usuarios autorizados accedan a los recursos del sistema o a la red.
- *Tráfico de relleno*: Es el envío de tráfico falso junto con los datos válidos para que el enemigo no sepa si se está enviando información ni la cantidad de datos útiles que se están transfiriendo.
- *Control del enrutamiento*: Permite enviar cierta información por determinadas rutas clasificadas. Asimismo, posibilita la solicitud de otras rutas en caso de que se detecten persistentes violaciones de integridad en una ruta determinada.

10.5 SISTEMAS CRIPTOGRÁFICOS

La criptografía tiene una larga y fascinante historia, desde su uso inicial y limitada por los egipcios hace unos 4 mil años, hasta el siglo XX, cuando jugó un rol crucial en las dos guerras mundiales. La criptografía a través de la historia ha sido usada como una herramienta para proteger los secretos y estrategias nacionales [Menezes *et al.*, 1997]. A través de los siglos, se ha creado y elaborado una gran diversidad de protocolos y mecanismos para hacer frente a los problemas de seguridad de la información. Sin embargo, la proliferación de computadoras y sistemas de comunicación en la década de 1960 trajo consigo una demanda por parte del sector privado de los medios necesarios para proteger la información digital. Lo anterior ha incrementado la demanda por servicios de criptografía. La *criptografía* es el estudio de técnicas matemáticas relacionadas con los aspectos de seguridad de la información, tales como la confidencialidad, integridad de datos, autenticación de la entidad y autenticación del origen de datos [Menezes *et al.*, 1997]. Los sistemas criptográficos son muy útiles para la seguridad en los sistemas distribuidos, ya que permiten mitigar muchas de las posibles vulnerabilidades que estos presentan. Los sistemas de encriptado más usados son los siguientes:

- Encriptado simétrico.
- Encriptado asimétrico.

En el método de encriptado simétrico, tanto el emisor como el receptor usan una misma clave secreta para cifrar como descifrar. Por otro lado, en el encriptado asimétrico cada usuario tiene una pareja de claves, una pública y otra privada, con la característica de que aquello que se cifra con una de las claves solo se puede descifrar con la otra clave. La figura 10.6 muestra el modelo de un encriptado simétrico, mientras que la figura 10.7 muestra el modelo para un encriptado asimétrico o de llave pública.

Figura 10.6. Modelo simplificado de encriptado simétrico [Stalling, 2004]

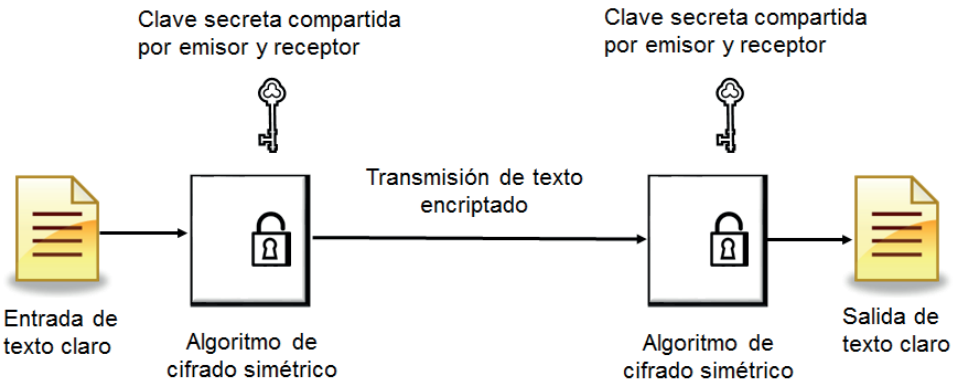
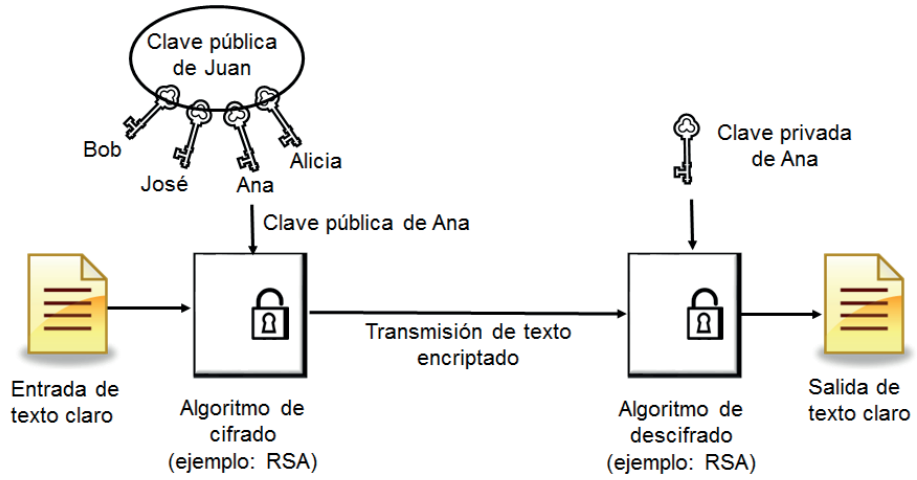


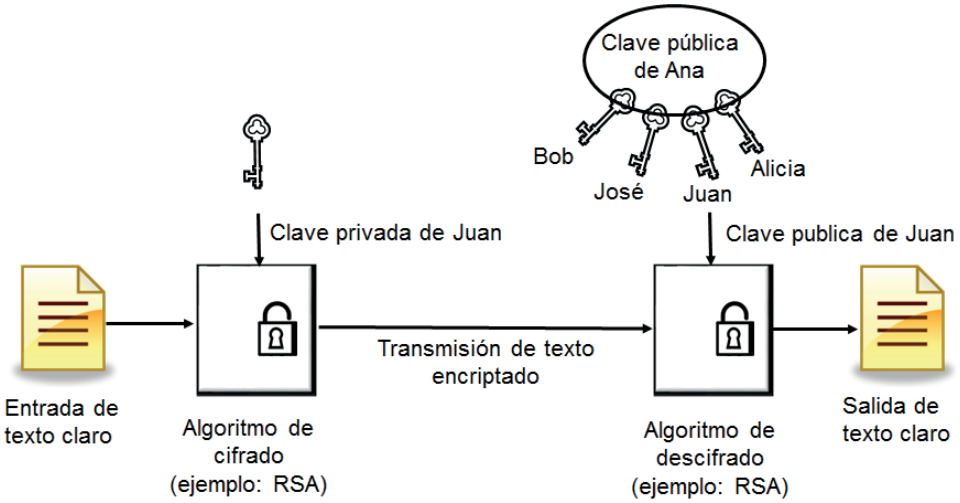
Figura 10.7. Modelo simplificado de encriptado asimétrico o público [Stalling, 2004]



En la figura 10.7, Juan se quiere comunicar secretamente con Ana, para lo cual usa la clave pública de Ana para cifrar su mensaje. Ana podrá abrir el mensaje cifrado usando su clave privada.

El método de clave asimétrica o pública también se puede usar para autenticar mensajes o firma digital. Este escenario es mostrado en la figura 10.8. En este caso, Juan quiere autenticarse ante Ana, para tal propósito necesita enviar su firma digital cifrada usando su clave privada. Ana autentica la firma usando la clave pública de Juan. Si Juan no fue quien envió la firma, entonces esta no podrá ser descifrada con su llave pública.

Figura 10.8. Modelo simplificado del uso de clave pública para autenticación [Stalling, 2004]



10.5.1 Requisitos para la criptografía de llave pública

Los requisitos que deben de ser observados para generar las llaves en el modelo de llave pública son [Stalling, 2004]:

1. Debe de ser computacionalmente fácil para generar un par de llaves (llave pública *KUb* y llave privada *KRb*).
2. Debe de ser fácil para el remitente generar texto cifrado (C):
$$C = E_{KUb}(M)$$
3. Debe de ser fácil para el receptor descifrar texto cifrado usando la llave privada:
$$M = D_{KRb}(C) = D_{KRb}[E_{KUb}(M)]$$
4. Debe de ser computacionalmente imposible determinar la llave privada (KRb) conociendo la llave pública (KUb).
5. Debe de ser computacionalmente imposible recuperar el mensaje M, sabiendo KUb y el texto cifrado C.
6. Cualquiera de las dos llaves se puede usar para cifrar, mientras que la otra llave se usa para descifrar el mensaje (M):
$$M = D_{KRb}[E_{KUb}(M)] = D_{Kun}[E_{KRb}(M)]$$

10.6 AUTENTICACIÓN

La autenticación es el acto que sirve para establecer o confirmar algo (o a alguien) como auténtico. La autenticación de un objeto puede significar la confirmación de su procedencia, mientras que la autenticación de una persona a menudo consiste en verificar su identidad. La autenticación está ligada a las preocupaciones de seguridad en los sistemas distribuidos.

Entre las preocupaciones de seguridad se pueden citar las siguientes:

- Confidencialidad y puntualidad.
- Para proporcionar confidencialidad, se debe de cifrar la identificación y la información de la clave de sesión.
- ¿Qué requiere el uso de llaves públicas o privadas previamente compartidas?
- La puntualidad es requerida para prevenir los ataques de repetición.
- La puntualidad puede alcanzarse al usar números de secuencia o estampillas de tiempo o reto/respuesta.

10.6.1 Kerberos

El sistema Kerberos es un proyecto de autenticación desarrollado en el Instituto Tecnológico de Massachusetts [Kerberos, 2014]. El nombre de este proyecto está inspirado en la mitología griega, donde un perro de tres cabezas es el guardián de la entrada al infierno.

Cuando los usuarios desean acceder a los servicios en los servidores, existen las siguientes posibles amenazas:

- El usuario puede pretender ser otro usuario.
- El usuario puede alterar la dirección de la red de una estación de trabajo.
- El usuario puede espiar los intercambios y usar un ataque de repetición.

Kerberos proporciona un servidor de autenticación centralizado para autenticar los usuarios a los servidores y los servidores a los usuarios. El protocolo se basa en el cifrado convencional, sin hacer uso del encriptado de clave pública. Existen dos versiones, la 4 y 5. La versión 4 hace uso de DES.

Un diálogo hipotético simple de autenticación es mostrado en la figura 10.9, donde los términos usados son indicados en la figura 10.10.

Figura 10.9. Diálogo hipotético simple de autenticación

(1)	C → AS:	IDc || P_c || IDv
(2)	AS → C:	Ticket
(3)	C → V:	IDc || Ticket

Ticket = E_{Kv}[IDc || P_c || IDv]

Figura 10.10. Términos empleados en diálogo hipotético simple de autenticación

C = Cliente
AS = Servidor de autenticación
V = Servidor
Idc = Identificador de usario en C
IDv = Identificador de V
P_c = Contraseña de usuario en C
ADc = Dirección de red de C
Kv = Clave de cifrado secreta compartida por AS en V
TS = Estampilla de tiempo
|| = Concatenacion

Riesgos a considerar durante el diseño:

- El tiempo de vida de un mensaje es asociado con el ticket de concesión.
- Si es demasiado corta, entones es solicitado repetidas veces por la contraseña.
- Si es demasiado larga, entonces existe una mayor oportunidad para un ataque de repetición.

La amenaza es que un oponente se robe el ticket y lo use antes de que el tiempo expire.

La figura 10.11 muestra el protocolo real de Kerberos. En este se considera el problema del ticket TGT (ticket que concede un ticket) capturado y la necesidad de determinar que quien presenta el ticket es el mismo que el cliente para quien se emitió dicho ticket. Aquí existe la amenaza de que un oponente pueda robar el ticket y usarlo antes de que expire. Kerberos soluciona este problema usando una clave se sesión (mensaje 2 de la figura 10.11).

Con el ticket y la clave de sesión, C está preparado para dirigirse al TGS. Para esto, C envía al TGS un mensaje que incluye el ticket y el ID del servicio solicitado (mensaje 3 de la figura 10.11). Adicionalmente, C transmite un autentificador que incluye el ID y la dirección del usuario de C y un sello de tiempo.

La respuesta desde el TGS (mensaje 4) sigue la forma usada en mensaje 2 [Stalling, 2004]. Es decir, el mensaje se cifra con la clave de sesión compartida por el TGS y C, e incluye una clave de sesión que comparten C y V, así como el ID de V y el sello de tiempo del ticket.

El usuario C envía este ticket de servicio reutilizable al servidor V (mensaje 5) junto con un autentificador. El servidor V descifra el ticket, recupera la clave de sesión y descifra el autentificador. En caso de que se requiera autentificación mutua, el servidor responde como se indica en el mensaje 6 de la figura 10.11. El esquema general de Kerberos se muestra en la figura 10.12.

Figura 10.11. Diálogo de autenticación de Kerberos [Stalling, 2004]

Intercambio de Servicio de Autenticacion: Para obtener un TGT (otorgamiento de entrada)

(1) C → AS: IDc || IDtgs ||TS1

(2) AS → C: $E_{Kc}[K_{c,tgs}|| IDtgs || TS_2 || Lifetime_2 || Ticket_{tgs}]$

Intercambio de TGS: Para obtener ticket de concesión de servicio

(3) C → TGS: IDv ||$Ticket_{tgs}$ ||Authenticatorc

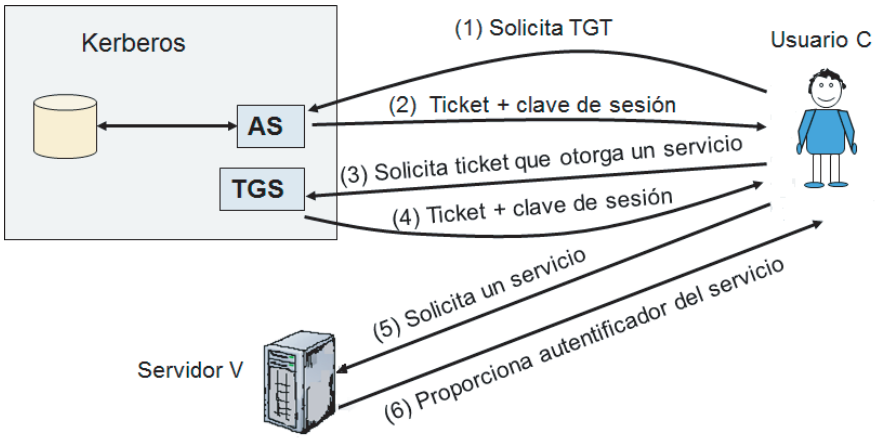
(4) TGS → C: $E_{Kc}[K_{c,v}|| IDv || TS_4 || Ticket_v]$

Intercambio de autenticación Cliente/Servidor: Para obtener un servicio

(5) C → V: $Ticket_v$ || $Authenticator_c$

(6) V → C: EKc,v[TS5 +1]

Figura 10.12. Esquema general de Kerberos



Existen casos en que las redes de clientes y servidores pertenecen a diferentes organizaciones constituyendo diferentes dominios. Sin embargo, los usuarios de un dominio pueden necesitar acceso a servidores de otro dominio y algunos servidores pueden estar dispuestos a ofrecer el servicio a esos usuarios, previa autenticación. En este entorno se propone un esquema conocido como Kerberos múltiple [Stalling, 2004]. Este esquema necesita lo siguiente:

1. El ID de usuario y la contraseña de todos los usuarios están registrados en la base de datos del servidor Kerberos.
2. Todos los servidores están registrados con el servidor Kerberos, quien comparte una clave secreta con cada uno de ellos.
3. Los servidores Kerberos de cada dominio se registran entre sí y comparten una clave secreta entre ellos.

La interoperabilidad de un servicio Kerberos múltiple para dos dominios es mostrado en la figura 10.13.

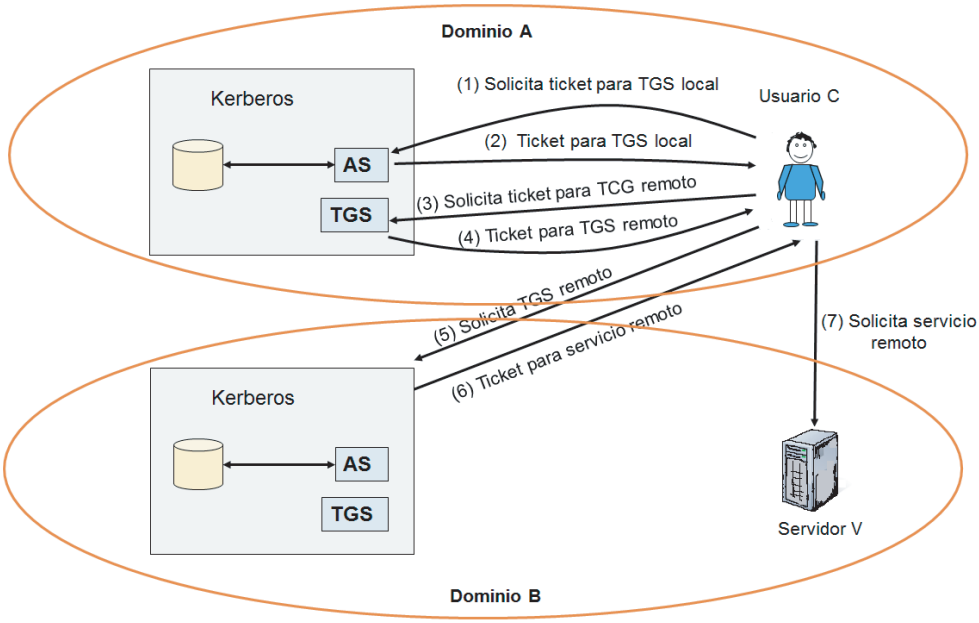
La versión 4 (Kerberos v4) está restringida a un solo dominio y la versión 5 (Kerberos v5) permite autenticación entre varios dominios. Kerberos v5 es un estándar de Internet especificado en RFC1510 y es usado por muchas aplicaciones.

Entre las principales diferencias entre la versión 4 y 5 se encuentran las siguientes:

- Dependencia del sistema de cifrado (V.4 DES).
- Dependencia del protocolo de Internet.

- Ordenamiento de bits de mensaje.
- Tiempo de vida del ticket.
- Reenvío de autenticación.
- Autenticación de intercampos.

Figura 10.13. Kerberos para solicitud de servicio en otro dominio



EJERCICIOS

1. Cita al menos tres requisitos que debe de cumplir la criptografía de llave pública.

2. Indica los componentes que constituyen la criptografía de llave pública y describe el escenario (usando un esquema) para un caso de encriptamiento y un caso de autenticación.

3. ¿Cuál es el proposito de la estampilla de tiempo en el sistema Kerberos?

4. Explica en qué consiste un ataque de interrupción.

5. Explica qué es un ataque a la confidencialidad.

6. ¿Qué tipo de ataque sería un ataque a la autenticidad?

7. Cita algunas recomendaciones relacionadas con las políticas en la seguridad.

8. Investiga el uso del algoritmo de llave pública (asimétrico) en la firma digital.

9. Cita dos requisitos para la criptografía de llave pública.

10. ¿Cuál es la función del tráfico de relleno en un mecanismo de seguridad?

11. ¿Cuáles son los aspectos que se deben de cuidar en un enfoque de seguridad en la información?

ACTIVIDAD INTEGRADORA

1. De la aplicación de los contenidos, así como de las habilidades y actitudes desarrolladas.
   Se recomienda exposición general del tema por parte del profesor, resaltando la importancia de los sistemas de seguridad en los sistemas distribuidos desde una perspectiva de los sistemas criptográficos y la autenticación de acceso a los sistemas. Debido a que los alumnos de la Licenciatura en Tecnologías y Sistemas de Información cursan una UEA llamada Seminario de seguridad, se recomienda no ahondar mucho en el tema, sino comprender los riesgos que implica tener la información en entornos distribuidos y las posibles soluciones de seguridad que se pueden ofrecer. El tema puede cubrirse en dos sesiones.
   El alumno debe de mostrar un gran interés por la observación, análisis, abstracción e interpretación de los riesgos de seguridad en los sistemas distribuidos, así como de algunos esquemas de seguridad para mitigar este tipo de riesgos.
2. Del logro de los objetivos establecidos en el capítulo o apartado del material.
   Se espera alcanzar los objetivos de este capítulo con el apoyo en la solución de los ejercicios, la exposición del profesor sobre el tema, así como un taller de análisis de diagramas sobre riesgos de seguridad y sus probables soluciones para mitigar estos riesgos.

Capítulo 11. Multimedia distribuida

Objetivo: Que el alumno comprenda aspectos del flujo de video respecto a sus estándares e infraestructuras de distribución, así como su importancia en los sistemas distribuidos actuales.

11.1. Introducción

Multimedia quiere decir "múltiples medios". Los medios pueden ser desde texto e imágenes hasta animación, sonido o video. Específicamente, la multimedia es una combinación de texto, sonidos, imágenes o gráficos ya sea estáticos o en movimiento. El video es un tipo de contenido que combina los tipos de medios anteriores, por lo que se constituye en un caso ideal de multimedia.

Los últimos avances de las tecnologías de Internet y la computación han abierto nuevas oportunidades para aplicaciones multimedia. En este nuevo escenario, la transmisión de video a través de Internet tiene una gran popularidad. Este hecho ha generado una revolución tecnológica y social, y diferentes sistemas de distribución de video han emergido. La transmisión de video se realiza desde un proveedor de servicio centralizado o desde múltiples proveedores de servicios a una gran multitud de personas, quienes utilizan diversas técnicas, tales como video bajo demanda o video IP para obtenerlos. La reproducción de video, en general, requiere de una alta velocidad de datos, baja latencia o de alto rendimiento, con el fin de ofrecer una buena calidad de transmisión. Sin embargo, la calidad de servicio (QoS) para la transmisión de video a través de Internet sigue siendo insuficiente e inconsistente. Esto se traduce en tiempos de transferencia muy largos y la imposibilidad de ver un video en tiempo real. Este capítulo revisa los con-

ceptos básicos relacionados con la difusión del video, como infraestructura y técnicas de codificación de video. El capítulo también analiza brevemente algunos sistemas de flujo de video.

Durante el flujo de video, el receptor puede ver el video mientras que una parte del archivo aún se sigue recibiendo y decodificando. De esta manera, la transmisión de video reduce el tiempo entre el inicio de la entrega y el inicio de la reproducción en el visor. Este retardo normalmente es de 5 a 15 segundos [Apostolopoulos, Tan & Wee, 2002]. Por otro lado, ya que solo una pequeña parte del video se almacena por el espectador durante la transmisión de video, los requisitos de almacenamiento son bajos. Sin embargo, la transmisión de video es sensible a la demora, debido a que los paquetes de video deben de llegar al receptor antes de sus plazos de emisión [Wang, Ostermann & Zhang, 2002]. Por lo tanto, este tipo de transmisión, como ya se señaló, requiere de alta velocidad de datos, baja latencia o de un alto rendimiento, lo cual es difícil de conseguir, ya que actualmente la Internet está basada en el protocolo IP [Wu *et al.*, 2001], que no garantiza una buena calidad de servicio (QoS).

11.2 Estándares de codificación de video

La compresión es un tema importante a considerar para la distribución de video digital, pues ayuda a reducir la alta tasa de bits resultante de la conversión digital a un nivel que puede ser soportado por las redes de comunicación. La compresión elimina las redundancias inherentes en señales de audio y de video digitalizadas con el fin de reducir la cantidad de datos que necesita ser almacenada y transmitida. Para alcanzar esta meta, se ha realizado un gran esfuerzo durante los últimos 30 años en el desarrollo de técnicas y estándares de compresión de video. Algunos de estos estándares de compresión son:

- H.261 [ITU-T, 1993].
- MPEG-1 [ISO / IEC, 1993].
- H.262 (MPEG-2) [ITU-T e ISO / IEC JTC, 1994].
- H.263 [ITU-T, 1995].
- MPEG-4 [ISO / IEC, 1999].
- H.264 / AVC [2009].
- H.265.

La norma H.265 es la más reciente especificación de video, sucesora del estándar H.264/AVC. En julio de 2014 se completó la segunda parte del estándar y se programó su publicación para finales de 2014. Este nuevo estándar incluye diferentes extensiones de mejora de video, entre las que se encuentran video en 3D. En este capítulo nos referimos al estándar H.264/AVC y una de sus extensiones.

11.2.1 Estándar H.264/AVC

H.264/Video Advanced Coding –comúnmente conocido como H.264 / AVC [2009]– es el estándar de codificación de video desarrollado por la ITU-T Video Coding Grupo de Expertos y la ISO/IEC Moving Picture Experts Group [Wiegand, Sullivan, Bjontegaard & Luhtra, 2003]. La estandarización de H.264 / AVC, también llamada MPEG-4/Video Advanced Coding, se completó en mayo de 2003 [ITU-T, 2003]. H.264/AVC ofrece una amplia gama de velocidades de bits, velocidades de cuadro y resoluciones espaciales, por lo que es muy versátil y garantiza su funcionalidad para aplicaciones diversas [Marpe, Wiegand & Sullivan, 2006], [Richardson, 2007], como difusión sobre los distintos tipos de redes, almacenamiento interactivo o de serie, el video bajo demanda o flujos de video.

Conceptualmente, H.264/AVC incluye [Wiegand *et al.*, 2003]:

- *La capa de codificación de video (VCL)*: Crea una representación codificada de los datos de video de la fuente con el fin de proporcionar flexibilidad y personalización para aplicaciones y redes heterogéneas.
- *La capa de abstracción de red (NAL)*: Formatea la VCL para hacerlo compatible con diferentes capas de transporte y dispositivos de almacenamiento, además de proporcionar la información de cabecera necesaria en un formato adecuado.

H.264/AVC organiza los datos de video codificados en unidades NAL, paquetes que contienen, cada uno, un número entero de bytes. Una unidad NAL comienza con una cabecera de un byte, que indica el tipo de los datos contenidos. Los bytes restantes representan datos de carga útil. Un conjunto de unidades NAL consecutivos, con propiedades específicas, se conoce como una unidad de acceso. La decodificación de una unidad de acceso se traduce en exactamente una imagen decodificada. Un conjunto de unidades de acceso consecutivos con ciertas propiedades se refiere como una secuencia de video codificada. Una secuencia de video codificada

representa una parte independientemente decodificable de una corriente de bits unidad NAL. Por otro lado, el diseño básico de VCL en H.264/AVC es muy similar al de las normas de codificación de video anterior, tales como H.261, MPEG-1 video, H.262 MPEG-2 de video, H.263, o MPEG-4 Visual. Sin embargo, H264/AVC incluye nuevas características que permiten una mejora significativa en la eficiencia de compresión con respecto a cualquier norma de codificación de video anterior. La principal diferencia con respecto a las normas anteriores es el gran aumento en la flexibilidad y la adaptabilidad.

11.2.2 Codificación de video escalable (SVC)

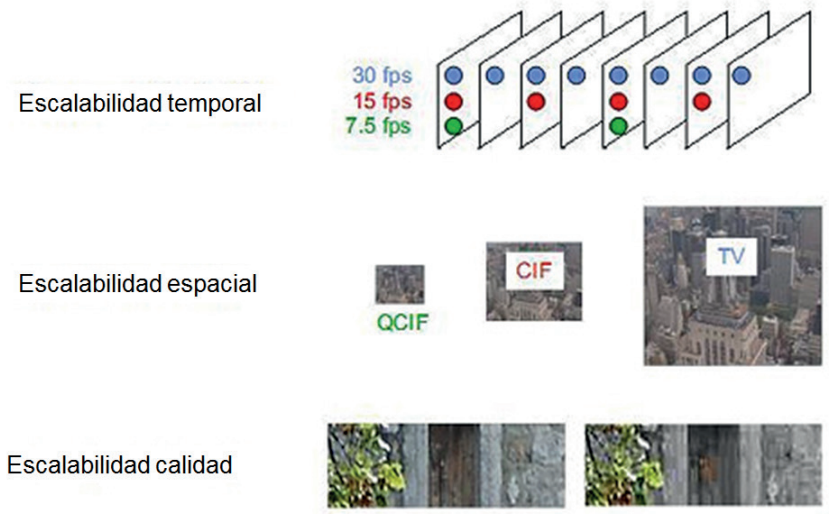
La capacidad de recuperar información importante de una imagen o video mediante la decodificación de solo partes de un flujo de bits comprimido se conoce como escalabilidad [Wang *et al.*, 2002]. Usando codificación de video escalable, partes de una secuencia de video se pueden quitar y la subclasificación resultante constituye otra fuente de video válido para algún decodificador objetivo. Este flujo de video parcial representa el contenido de código con una calidad de reconstrucción, y que es menor que la calidad del flujo de video completa del video original, pero alta si se considera la menor cantidad de datos restantes [Schwarz, Marpe & Wiegand, 2007]. Actualmente las redes heterogéneas están de moda y parecen ser la tendencia hacia el futuro. En estas condiciones, SVC emerge como una solución valiosa por adaptar fácilmente la velocidad de codificación a las distintas tasas de transmisión y diferentes dispositivos físicos.

SVC representa el video en una capa base (BL) y una o más en las capas de mejora (EL), donde cada EL actualiza de manera acumulativa la calidad de video aportado por la BL. En esencia, SVC codifica el video una vez, en un flujo de bits comprimido. Cada uno de los diferentes usuarios puede ahora extraer la cantidad justa de los datos de esta secuencia de video común, según las condiciones de la red o el tipo de dispositivo que esté utilizando. La principal limitación es que para decodificar una capa de mejora particular, se requiere haber obtenido antes la capa base (BL), así como todas las capas de mejora inferiores. Las formas habituales de escalabilidad son (ver figura 11.1):

- Escalabilidad de calidad o escalabilidad SNR (relación señal-ruido).
- Escalabilidad espacial.
- Escalabilidad temporal.

En la escalabilidad SNR, el video es representado por diferentes capas que varían el nivel de calidad de percepción. La decodificación de la capa base proporciona una baja calidad del video reconstruido. Sin embargo, la calidad resultante del video reconstruido se incrementa mediante la decodificación de las capas de mejora. En escalabilidad espacial, la BL y el EL suelen utilizar diferentes resoluciones de imagen espacial (la resolución de la EL es más alta que la BL), por lo que es útil para dispositivos con tamaño limitado de pantalla, como los teléfonos celulares (*smartphones*). Por último, la escalabilidad temporal permite diferentes resoluciones temporales o tasas de cuadros para representar el mismo video. Esta solución permite a los usuarios con conexiones más lentas de datos acceder al mismo contenido de video, pero con velocidades más lentas.

Figura 11.1. Tipos básicos de codificación de video escalable



Cortesía de Fraunhofer Heinrich Hertz-Institute [HHI, 2010].

11.3 INFRAESTRUCTURAS PARA FLUJOS DE VIDEO

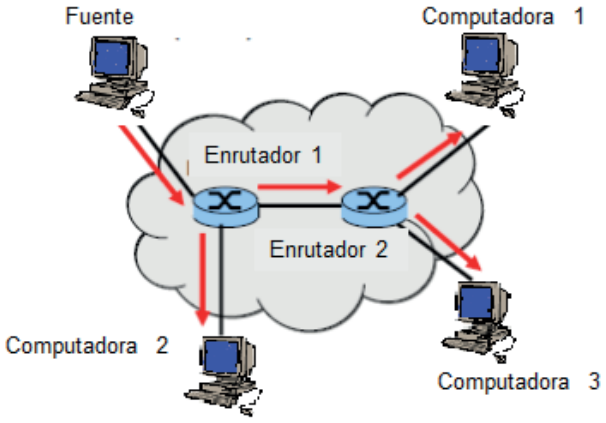
Esta sección describe las diferentes alternativas para distribución de video. Estas infraestructuras son:

- Multicast IP (multidifusión IP).
- Red de distribución de contenidos.
- Multicast de capa de aplicación.

11.3.1 Multicast IP

IP Multicast fue propuesto por Deering [1988] como una solución para realizar difusión de contenidos de uno-a-muchos. Multicast presenta una mejor eficiencia que unicast debido a su reducida sobrecarga de transmisión en el emisor y la red, lo que reduce el tiempo de entrega para la distribución de contenido. Una alternativa de distribución basada en unicast requiere que la fuente envíe un flujo individual a cada usuario final, lo cual es crítico para aplicaciones de alto ancho de banda, como el video, el cual requiere una gran porción de ancho de banda para un solo flujo. IP Multicast reduce el tráfico mediante la distribución simultánea de una sola copia a miles de potenciales usuarios finales, mientras que los paquetes de multidifusión se replican en la red por los enrutadores. Diferentes aplicaciones, como videoconferencias, educación a distancia y noticieros, se basan en la tecnología de multidifusión. La figura 11.2 muestra cómo se distribuyen los datos de una fuente a varios usuarios finales usando multicast IP.

Figura 11.2. Multicast IP



Multicast IP ha demostrado ser una tecnología de rendimiento eficiente para la entrega de datos desde una fuente a un gran número de receptores. Desafortunadamente, IP multicast no se ha desplegado plenamente en la Internet debido principalmente a:

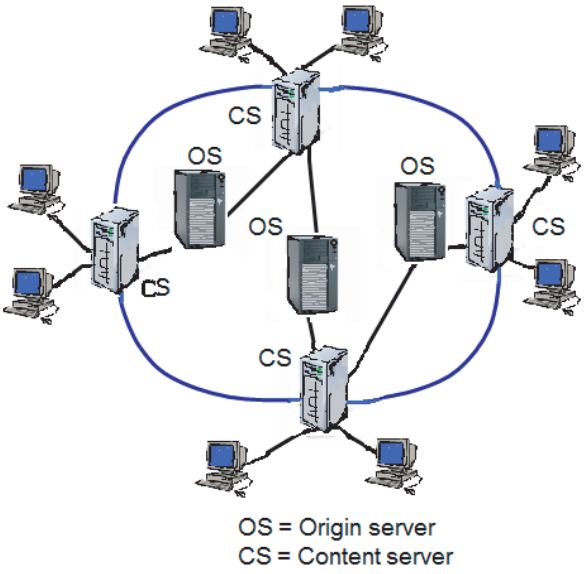
- Que su implementación requiere apoyo de enrutamiento de todos los ISP (proveedores de servicios de Internet), lo cual implica modificaciones sustanciales de la infraestructura de Internet.

- Cuestiones relacionadas con el control y gestión de la red [Pendarakis, Verma & Waldvogel, 2001], como la fiabilidad de extremo a extremo, y el flujo y control de congestión.

11.3.2 Red de distribución de contenidos

En la Internet actual, las aplicaciones de flujos de video de uno-a-muchos se basan en el modelo cliente-servidor tradicional de redes de distribución de contenido (CDN). Soluciones comerciales como Akami [Kontothanassisy *et al.*, 2004], [Akami, 2014], y otras, se ofrecen a través de un sistema de CDN. Un CDN está formado por servidores de contenidos conectados en red a través de Internet, que cooperan entre sí para distribuir contenidos de modo transparente para los usuarios finales. Un ejemplo de una CDN se muestra en la figura 11.3.

Figura 11.3. Red de distribución de contenidos



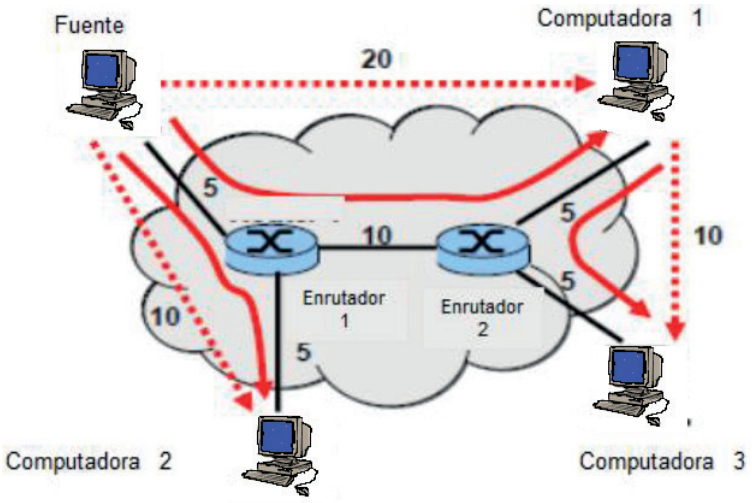
Típicamente, los servidores de contenido (CS) están situados cerca de los usuarios para servir el contenido solicitado rápidamente. Los servidores de contenido de la CDN están conectados a los proveedores de contenido (OS) a través de una red interna, que se utiliza para transferir el contenido de los proveedores a los servidores de contenido (CS).

Sin embargo, el enfoque de CDN se enfrenta a una serie de problemas tales como que el punto único de falla y el acceso es costoso para las redes de alta tasa. Además, incluso los grandes servidores de flujos de video no son capaces de alimentar a más de algunos cientos de sesiones de flujos de video de manera simultánea y la selección del mejor servidor de video en un CDN durante una sesión es difícil. Estas restricciones limitan el desempeño de la CDN.

11.3.3 Multicast de capa de aplicación

Para afrontar las limitantes de los modelos ya explicados, en los últimos años diversos investigadores optaron por utilizar soluciones en el nivel de aplicación del modelo OSI como una alternativa para implementar multidifusión [Pendarakis *et al.*, 2001], [Banerjee, Bhattacharjee & Kommareddy, 2002], [Chu, Rao, Seshan & Zhang, 2002]. En el multicast de capa de aplicación (ALM), todas las tareas de multidifusión se aplican en las computadoras anfitrionas finales exclusivamente, mientras que la infraestructura de la red se mantiene. Un ejemplo de este tipo de sistemas son las redes P2P. La figura 11.4 representa un ejemplo ALM, donde los números indican los retrasos de enlace.

Figura 11.4. Multicast de capa de aplicación



Un inconveniente en el sistema de ALM es la penalización de rendimiento asociada con entornos dinámicos y heterogéneos de Internet, debido a

que el rendimiento se afecta por la ubicación y la estabilidad de los usuarios finales.

11.4 SISTEMAS DE FLUJOS DE VIDEO BASADOS EN REDES P2P

Un sistema P2P de distribución de video implica dos componentes importantes [Magharei & Rejaie, 2006a]:

- Una red superpuesta.
- Un mecanismo de entrega de contenido.

Una red superpuesta se construye sobre la red subyacente física IP [Gao & Huo, 2007] utilizando un mecanismo que determina cómo se conectan los pares. El mecanismo de entrega de contenido es responsable de la transmisión de los contenidos de cada peer a través de la red de superposición. Una multidifusión P2P de superposición tiene las siguientes ventajas sobre multicast IP [Zheng *et al.*, 2005]:

- No se requiere soporte de enrutadores.
- Mayor flexibilidad y adaptabilidad a las diversas necesidades de las aplicaciones.

Zheng *et al.* [2005] señalan que para construir y mantener una red P2P de superposición eficiente, principalmente deben de considerarse tres problemas :

1. La arquitectura de la red P2P, la cual define la topología que se utiliza para construir la red superpuesta.
2. La gestión de la red y cómo se gestionan los peers en el grupo multicast, especialmente cuando los peers presentan capacidades y comportamientos heterogéneos.
3. La adaptabilidad de la red de superposición para encaminar y planificar el video en un ambiente de Internet donde los enlaces tienen un comportamiento impredecible.

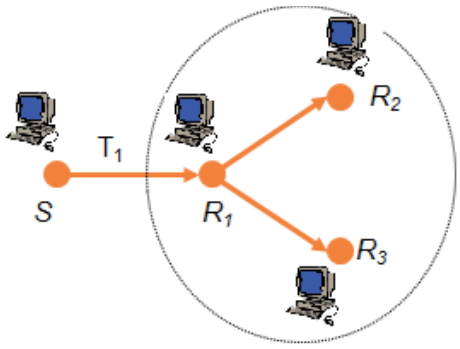
Se han propuesto varias soluciones con el fin de abordar estas cuestiones. Los principales tipos de topologías P2P para ofrecer multidifusión P2P son [Magharei & Rejaie, 2006], [Gao & Huo, 2007], [Venot & Yang, 2007]:

- Árbol.
- Bosque.
- Malla.

11.4.1 Topología de flujo de video P2P basado en árbol

En el enfoque basado en árbol, un mecanismo de construcción de superposición organiza a los peers participantes en un solo árbol cuya raíz se encuentra en el nodo fuente. Los peers que participan están organizados como nodos interiores o nodos de hoja en un solo árbol. En la figura 11.5 se muestra una topología basada en árbol. La fuente S envía los datos al peer R1, que reenvía los datos a los peers R2 y R3. Un flujo de video en esta configuración es básicamente empujado desde un enrutador padre a sus enrutadores hijos a lo largo de una ruta bien definida.

Figura 11.5. Topología de flujo de video P2P basado en árbol

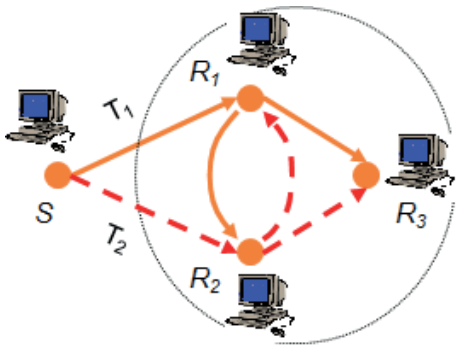


En la figura 11.5 se observa que la capacidad del peer R1 es utilizado por el árbol de multidifusión para la distribución de contenidos, mientras que la capacidad de los peer hoja R2 y R3 no se utiliza. Aunque un enfoque de árbol probablemente representa la estructura de distribución más eficaz en términos de ancho de banda y optimización de retardo [Wang *et al.*, 2007], esta configuración tiene un inconveniente debido a que toda la carga generada por el reenvío de mensajes multicast la realizan un número pequeño de nodos interiores.

11.4.2 Topología de flujo de video P2P basado en bosque

Un enfoque basado en bosque organiza a los peers participantes en una difusión de video en múltiples árboles [Gao & Huo, 2007] y distribuye la carga de reenvío entre estos de una manera eficiente. En una distribución basada en bosque, cada peer determina un número adecuado de árboles para unirse sobre la base de su capacidad de carga. La figura 11.6 muestra un ejemplo de una superposición basado en bosque. En este escenario, los peers participantes se organizan en varios árboles. Para este fin, cada peer se coloca como un nodo interno en al menos un árbol y como un nodo de hoja en otros árboles. La fuente S divide su contenido en dos pedazos y distribuye cada pedazo usando dos árboles de multidifusión separados T1 y T2. Cada nodo interno en cada árbol de distribución reenvía la parte de contenido recibida a todos sus nodos secundarios. Por lo tanto, el algoritmo de construcción del árbol representa un componente importante del enfoque basado en bosque.

Figura 11.6. Topología de flujo de video P2P basado en bosque



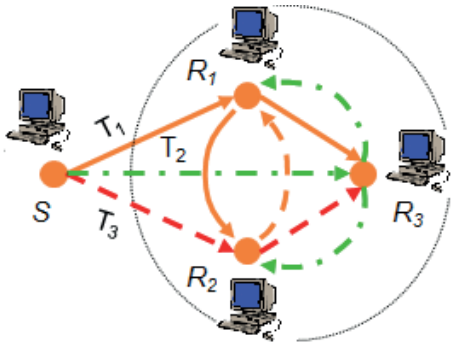
Una estrategia basada en múltiples árboles reduce al mínimo el efecto de rotación [Magharei *et al.*, 2007] y optimiza el uso de los recursos disponibles en el sistema. Sin embargo, la determinación del número de árboles necesarios para maximizar el rendimiento global es un problema abierto.

11.4.3 Topología de flujo de video P2P basado en malla

Las redes superpuestas basadas en malla es un enfoque inspirado en BitTorrent [Cohen, 2003] o Bullet [Kostic, Rodriguez, Albrecht & Vahdat, 2003]. La

figura 11.7 presenta un ejemplo de superposición basado en malla. Dicho esquema está formado por peers conectados al azar, donde cada peer (excepto la fuente) intenta mantener cierto número de peers padres y también sirve a un número determinado de peers hijos utilizando un mecanismo de enjambre para la entrega de contenido [Magharei & Rejaie, 2006b]. En una superposición basada en malla, un peer puede concurrentemente recibir datos desde diferentes emisores, contribuyendo cada uno con una parte de su capacidad de carga. Además, los peers que solicitan también pueden enviar y recibir datos entre sí.

Figura 11.7. Topología de flujo de video P2P basado en malla



Una topología basada en malla es robusta, sin embargo, puede presentar un alto retardo y sobrecarga. También, debido al comportamiento dinámico e impredecible de los peers, es difícil seleccionar los proveedores de video adecuados, así como la manera adecuada de cooperar y programar los datos de peers solicitantes. La tabla 11.1 resume las principales características de estas topologías.

Tabla 11.1. Principales características de las topologías P2P para flujos de video

Basada en árbol		Basado en bosque		Basado en malla	
Ventajas	*Desventajas*	*Ventajas*	*Desventajas*	*Ventajas*	*Desventajas*
Estructura simple	No usa capacidad de los peers hojas	Usa capacidad de peers hojas	La construcción es compleja	Usa capacidad peers hojas	Largas demoras en el arranque
Baja sobrecarga	Vulnerabilidad	Calidad escalable	Vulnerabilidad	Robustez	Sobrecarga
Arranque rápido				Alta disponibilidad	Reproducción de video se congela frecuentemente
Ejemplos de sistemas: ZigZag End-System Multicast		Ejemplos de sistemas: Splitstream CoopNet		Ejemplos de sistemas: BitTorrent Mutualcast	

Los contenidos multimedia se pueden distribuir en Internet tanto como transmisiones en vivo o bajo demanda. Algunos ejemplos de sistemas de difusión de video comercial y experimental son PPLive [2014] y P2P-Next [2014].

P2P-next es un proyecto Paneuropeo lanzado en enero de 2008, el cual es patrocinado por 21 socios industriales, proveedores de contenidos multimedia e instituciones de investigación. P2P-Next es un proyecto integrado a gran escala, cuyo objetivo es construir una plataforma de entrega de contenido de próxima generación basado en el paradigma peer-to-peer (P2P). Los principales objetivos de este proyecto son [Wilson & Miles, 2010]:

- Una plataforma que soporte portales de video de banda ancha y entregue contenidos a través de redes P2P a pantallas de televisión y a computadoras personales.
- Integrar funciones de Gestión de Derechos Digitales (DRM) y funciones para ayudar a los productores de contenidos a convertir contenido de video lineal en contenidos interactivos.
- Crear modelos de negocios sostenibles para el sistema P2P-Next.

EJERCICIOS

1. Actualmente Internet no ofrece reservación de recursos o calidad de servicio (QoS). ¿Cómo logran alcanzar las aplicaciones de flujo de video actual algunos niveles de calidad?

2. Explica la diferencia entre flujo de video y descarga de video

3. ¿Qué ventajas con respecto al almacenamiento y la reproducción introduce el flujo de video?

4. Investiga las principales diferencias entre el estándar H.265 y H.264.

5. ¿Qué ventaja introduce la distribución de video basado en bosque con respecto a la distribución de video basado en árbol?

6. ¿Por qué existe la posibilidad de que la reproducción de un video se congele frecuentemente en una distribución basada en malla?

7. ¿Cuáles son las principales limitantes para desplegar sistemas de video usando IP Multicast?

8. ¿Cuáles son los principales beneficios de usar codificación de video escalable?

9. Cita algunos casos en que sea aplicable y útil usar codificación de video escalable.

10. ¿Por qué es más robusta una red P2P basada en malla que una red P2P basada en único árbol?

ACTIVIDAD INTEGRADORA

1. De la aplicación de los contenidos, así como de las habilidades y actitudes desarrolladas.
   Se recomienda una exposición general por parte del profesor. Además, se pueden dejar temas específicos a investigar al alumno como actividad

extraclase para desarrollar y presentar una exposición la última sesión del curso. Se recomienda realizar algún prototipo de flujo de video que puede servir como proyecto final.

El alumno de debe mostrar un gran interés por el diseño y programación de infraestructuras de redes sobrepuestas basadas en protocolos TCP/IP para flujos de video.

2. Del logro de los objetivos establecidos en el capítulo o apartado del material.

   Se espera alcanzar los objetivos de este capítulo con el apoyo en la solución de los ejercicios, la exposición del profesor, el desarrollo de un prototipo básico como proyecto final y la exposición de los alumnos en la última sesión de clases.

Capítulo 12. Cómputo en la nube

Objetivo: Que el alumno comprenda las diferentes características, ventajas y desventajas del cómputo en la nube y su importancia como un nuevo paradigma del cómputo distribuido.

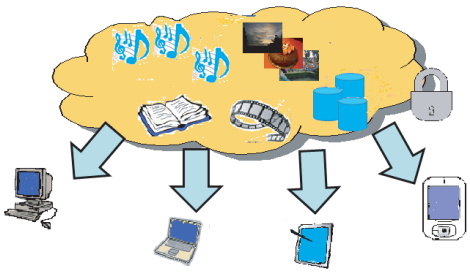
12.1 Introducción

El rol de las tecnologías de la información en nuestra sociedad ha cambiado rápidamente durante los últimos años. Actualmente la gran cantidad de información generada por los usuarios junto con el explosivo crecimiento del número de dispositivos conectados a la Internet ha revolucionado la manera en que almacenamos y accedemos a la información. La cantidad de información generada por los usuarios suele exceder la capacidad de los dispositivos. El cómputo en la nube ha emergido como una solución prometedora para afrontar estos retos. El cómputo en la nube provee servicios de almacenamiento, poder de cómputo y flexibilidad a usuarios finales para acceder a datos desde cualquier lugar en cualquier momento. El cómputo en la nube es cada día más popular. Muchos usuarios y organizaciones usan estos servicios para almacenar sus datos o para obtener más poder de cómputo.

De acuerdo con Armbrust *et al.* [2009], *cómputo en la nube* se refiere a los servicios y distribución de aplicaciones sobre la Internet, así como al hardware y software de sistema en los centros de datos que proveen estos servicios. Con base en esta definición, se puede deducir que el cómputo en la nube es un modelo que permite acceso a archivos, aplicaciones o servicios de una manera omnipresente y ubicua a través de la red, con el propósito de compartir un conjunto de recursos de cómputo configurable. Un ejemplo de los recursos que pueden ser descargados desde una nube se muestra en la figura 12.1. Estos recursos pueden ser servidores, almacenamiento, aplicaciones y servicios, los cuales pueden ser rápidamente aprovisionados y liberados con

un mínimo esfuerzo en la gestión de servicios o interactuar con el proveedor de manera veloz. Por lo tanto, el cómputo en la nube ofrece la ilusión de una escalabilidad ilimitada y bajo demanda.

Figura 12.1. Ejemplo de cómputo en la nube



12.2 ABSTRACCIÓN Y VIRTUALIZACIÓN

La computación en la nube se refiere a dos conceptos básicos [Sosinsky, 2011]:

- Abstracción.
- Virtualización.

La abstracción significa que los detalles de implementación de los usuarios del sistema y los desarrolladores se abstraen. De esta manera, las aplicaciones se ejecutan en los sistemas físicos que no están especificados, los archivos se almacenan en lugares donde los usuarios no conocen su ubicación real, el sistema se puede gestionar a través de la subcontratación y los clientes pueden acceder al sistema de forma ubicua.

Por otro lado, los recursos de los sistemas se reúnen y se comparten virtualmente. En cuanto a la virtualización, Sosinsky [2011] señala que los sistemas de almacenamiento se pueden aprovisionar cuando sea necesario desde un sistema centralizado, los costos se evalúan de manera dosificada, el multiauto arrendamiento está habilitado y los recursos son escalables con agilidad.

12.3 MODELOS DE CÓMPUTO EN LA NUBE

El cómputo en la nube típicamente se ha usado como un sistema de almacenamiento. Muchos especialistas separan el cómputo en la nube en dos diferentes modelos:

- Servicio.
- Desarrollo.

Un modelo de servicio define el nivel de abstracción en el que un cliente interactúa en un entorno de cómputo en la nube [Babaoglu, Marzolla & Tamburini, 2012], mientras que un modelo de desarrollo se refiere a la ubicación y la gestión de la infraestructura de la nube.

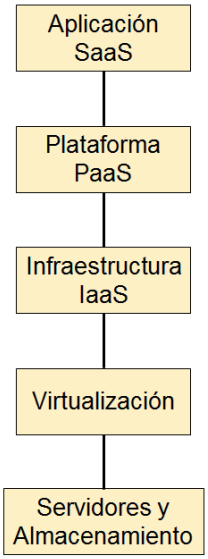
12.4 Tipos de servicios del cómputo en la nube

La computación en nube también tiene tres tipos de servicio aceptado y definido por el NIST (Instituto Nacional de Estándares y Tecnología), que son:

- Modelo de software como servicio (SaaS).
- Modelo de Plataforma como Servicio (PaaS).
- Infraestructura como Servicio (IaaS).

La figura 12.2 muestra una arquitectura de capas para el cómputo en la nube usando una clasificación basada en el tipo de servicio.

Figura 12.2. Arquitectura de capas del cómputo en la nube [Furht, 2010]



SaaS es un entorno operativo completo con aplicaciones, gestión e interfaces de usuario. En este modelo, la aplicación está determinada por el cliente a través de una pequeña interfaz de usuario, el cual podría ser un navegador web y los clientes solamente tener la responsabilidad absoluta de la gestión de sus archivos. Toda la aplicación es responsabilidad del proveedor [Sosinsky, 2011] y el cliente de la nube no tiene control sobre esta infraestructura. Una nube PaaS proporciona máquinas virtuales, sistemas operativos, servicios de aplicaciones, marcos de trabajo para desarrollo, operaciones y estructuras de control para las aplicaciones desarrolladas por el cliente de la nube. En este modelo, los usuarios o los clientes pueden desarrollar sus aplicaciones dentro de la infraestructura de nube o utilizar sus aplicaciones programadas. El proveedor de servicios gestiona la infraestructura de nube, sistema operativo, software o la activación [Sosinsky, 2011]. Sin embargo, los clientes son responsables de la instalación y mantenimiento de las aplicaciones que están desarrollando. Por último, una nube IaaS ofrece capacidades de computación fundamentales tales como máquinas virtuales, almacenamiento virtual, la infraestructura virtual y otras analogías de hardware como una provisión para los clientes. El proveedor de IaaS gestiona toda la infraestructura, mientras que el cliente es responsable de los aspectos de desarrollo [Babaoglu *et al.*, 2012], [Sosinsky, 2011].

12.5. TIPOS DE CÓMPUTO EN LA NUBE

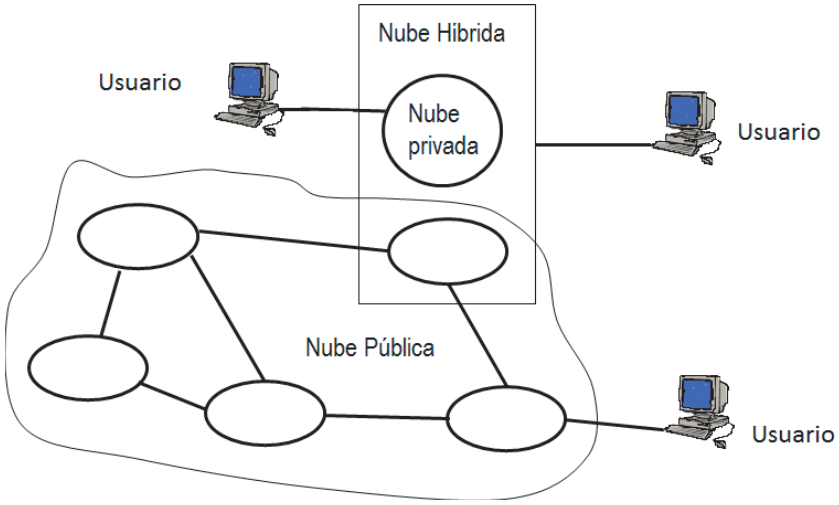
Con respecto al modelo de desarrollo, NIST ha propuesto los siguientes tipos de cómputo en la nube:

- Nube pública.
- Nube privada.
- Nube comunitaria.
- Nube híbrida.

Una nube pública es una infraestructura disponible para uso abierto al público en general. Este tipo de infraestructura en la nube puede ser propiedad de una organización, como universidades, corporaciones empresariales, organizaciones gubernamentales, o alguna combinación de ellos. Una nube privada es una infraestructura para el uso exclusivo de una organización que comprende múltiples consumidores tales como las unidades de negocio. La nube puede ser gestionada por dicha organización, por un tercero o por

alguna combinación de ellos. Una nube privada puede existir dentro o fuera de las instalaciones de una organización. Una nube comunitaria es un modelo en el que la infraestructura en la nube puede ser compartida por múltiples organizaciones que tienen un interés compartido o común, como la política, la misión, consideraciones de cumplimiento o requisitos de seguridad. Este tipo de infraestructura en la nube puede ser concesionado, administrado y operado por una o más de la organización en la comunidad, un tercero o alguna combinación de ellos. Este tipo de nube se puede ubicar dentro o fuera de las instalaciones de la empresa. La figura 12.3 muestra una disposición de tres de estas arquitecturas.

Figura 12.3. Tres tipos de cómputo en la nube [Furht, 2010]



12.6 Características del cómputo en la nube

El cómputo en la nube introduce un número de nuevas características comparadas a otros paradigmas. Algunas de estas características son las siguientes [Furht, 2010]:

- *Escalabilidad y servicios bajo demanda*: La computación en la nube ofrece recursos y servicios para los usuarios que la soliciten. Los recursos son escalables a lo largo de varios centros de datos.

- *Interfaz centrada en el usuario*: Las interfaces de la nube son independientes de la ubicación y pueden ser accedidas por interfaces bien establecidas, como los servicios web y los navegadores de Internet.
- *Garantía de calidad de servicio (QoS)*: El cómputo en la nube puede garantizar calidad de servicio (QoS) para los usuarios en términos de rendimiento del hardware tal como CPU, ancho de banda y capacidad de memoria.
- *Sistema autónomo*: Los sistemas de cómputo en la nube son sistemas autónomos administrados de manera transparente para los usuarios. Sin embargo, el software y los datos dentro de las nubes pueden ser reconfigurados y consolidados en una plataforma sencilla en función de las necesidades del usuario de modo automático.
- *Precios*: La computación en nube no requiere una inversión de los usuarios. No se requiere el gasto de capital. Los usuarios pagan por los servicios y la capacidad a medida que los necesitan.

12.7 VENTAJAS DEL CÓMPUTO EN LA NUBE

Entre las ventajas que introduce el uso del cómputo en la nube se pueden mencionar las siguientes [Sosinsky, 2011]:

- *Reducción de costes*: Dado que las redes de nube operan a mayores eficiencias y con mayor utilización, importantes reducciones de costos se pueden a menudo encontrar.
- *Facilidad de uso*: Dependiendo del tipo de servicio que se ofrece, es posible que un usuario no requiera licencias de hardware o software para implementar su servicio.
- *Calidad de servicio*: La calidad de servicio (QoS) es algo que se puede obtener bajo contrato de su proveedor.
- *Fiabilidad*: La escala de las redes de cómputo en la nube y su capacidad para proporcionar equilibrio de carga y conmutación por error los hace altamente confiables, a menudo mucho más fiable que aquello que puede lograr en una sola organización.
- *Gestión de externalización de las TI (Tecnologías de la Información)*: Un despliegue de cómputo en la nube permite a otras personas manejar la infraestructura de computación, mientras que el usuario maneja su negocio. En la mayoría de los casos, se logran reducciones considerables en los costos de personal de TI.

- *Mantenimiento sencillo y de actualización*: Debido a que el sistema es centralizado, se puede aplicar fácilmente los parches y actualizaciones. Esto significa que sus usuarios siempre tienen acceso a las últimas versiones de software.
- *Baja barrera de acceso*: En particular, los gastos de capital iniciales se reducen drásticamente. En el cómputo de la nube cualquiera puede agigantar sus posibilidades de uso de recursos en cualquier momento.

12.8 Retos del cómputo en la nube

El cómputo en la nube todavía tiene retos los cuales que son atendidos actualmente por investigadores y profesionales en el área. Entre estos retos se pueden mencionar los siguientes:

- *Rendimiento*: El principal problema en el rendimiento puede ser para algunas aplicaciones intensivas orientadas a transacciones y otras para uso intensivo de datos, en el que la computación en nube puede carecer de un desempeño adecuado. También una alta latencia y retardos puede ser experimentado por usuarios que están a grandes distancias de los proveedores de las nubes.
- *Seguridad y privacidad*: Este tema todavía es una tarea pendiente, ya que empresas y usuarios están preocupados por la vulnerabilidad a los ataques que se pueden presentar a los recursos que están en la nube.
- *Control*: Esto se refiere a que los proveedores del cómputo en la nube tienen un control total de la plataforma, lo que ha preocupado a algunos departamentos de Tecnologías de la Información con respecto a la gestión de sus datos.
- *Costos de ancho de banda*: Los gastos en el ancho de banda de red podrían crecer significativamente para aplicaciones intensivas de datos.
- *Confiabilidad*: El cómputo en la nube no siempre ofrece una fiabilidad durante largos periodos, ha habido casos en que el servicio ha sufrido de apagones por algunas horas.

12.9 Cómputo en la nube soportado por redes Peer-to-Peer (P2P)

Aunque la computación en la nube introduce varios beneficios, como poder de cómputo masivo, capacidad de almacenamiento y una gran flexibilidad,

el cómputo en la nube se enfrenta a varios desafíos. La mayoría de los sistemas de computación en nube tradicionales están centralizados y basados en el paradigma cliente-servidor. Una estructura centralizada presenta varios inconvenientes en la computación en nube, como la dependencia de almacenamiento, la escalabilidad, la privacidad a nivel local o la conectividad. En este escenario, las redes peer-to-peer (P2P) se han convertido en una plataforma de gestión de información distribuida prometedora.

En una red P2P cada peer toma el rol tanto de cliente como de servidor al mismo tiempo. Ventajas importantes de las redes P2P para ser usadas en cómputo en la nube son:

- Escalabilidad.
- Incremento en la capacidad de procesamiento.
- Mejor aprovechamiento del ancho de banda disponible.
- Mayor tolerancia a fallas del sistema.
- Mayor capacidad de almacenamiento distribuido.
- Mejor gestión de la privacidad.
- Mejor distribución y balanceo en la carga de trabajo.

Lo anterior se ve más limitado en arquitecturas de cómputo en la nube basados en el modelo cliente-servidor.

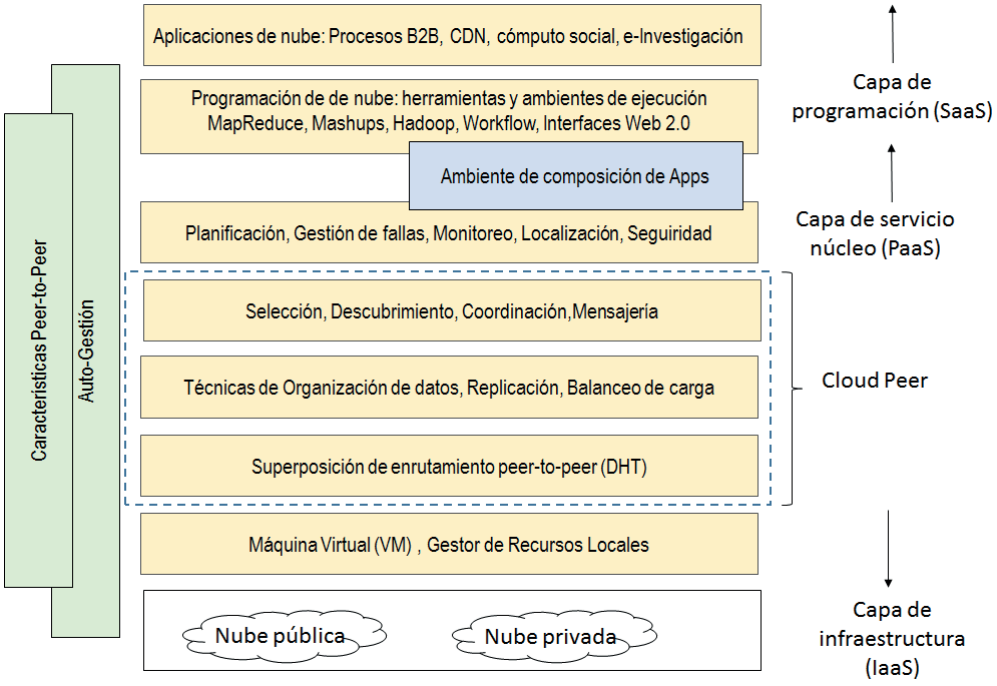
Diferentes sistemas de cómputo basados en redes P2P han sido desarrollados y reportados en la literatura. Entre estos podemos mencionar:

- Cloud peer.
- Cloud@Home.

Cloud peer es propuesto por Ranjan *et al*. [2010] como una novedosa plataforma de computación en la nube que crea una red superpuesta de máquina virtual (VM), así como servicios de aplicaciones para apoyar el descubrimiento de servicios escalables, de autogestión y balanceo de carga. Esta propuesta se basa en un modelo de red P2P estructurado (basado en tablas hash distribuidas) para hacer frente a las limitaciones asociadas al enfoque del cómputo en la nube centralizadas o jerárquicas tradicionales. Las tablas de hash distribuidas (DHT) permiten que el enrutamiento y descubrimiento de consulta/información determinista sea cercana a los límites logarítmicos en cuanto a la complejidad del mensaje de red. Cloud peer combina cómputo en la nube del tipo privado y público, redes superpuestas, así como técnicas de indexación de redes P2P estructuradas para apoyar el descubri-

miento de servicios escalables y de autogestión, además del balanceo de carga en entornos de cómputo en la nube. En concreto, Cloud peer es implementada usando Pastry [Rowstron & Druschel, 2001], red P2P superpuesta y de enrutamiento basado en DHT. Los autores llevan a cabo sus experimentos en la plataforma de Amazon EC2 [Amazon, 2014] y sus resultados confirman que es posible diseñar técnicas y sistemas de aprovisionamiento de cómputo en la nube basado en redes peer-to-peer. La arquitectura de Cloud peer se muestra en la figura 12.4.

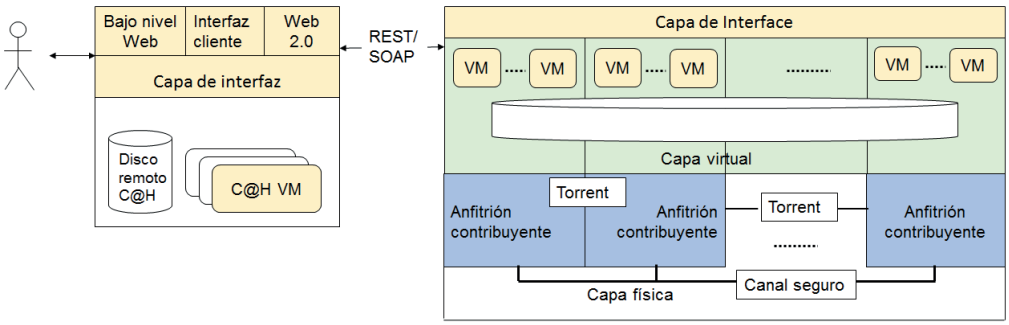
Figura 12.4. Arquitectura de Cloud peer [Ranjan *et al.*, 2010]



Cunsolo, Distefano, Puliafito & Scarpa [2009] proponen Cloud@Home, un sistema híbrido que combina características de los paradigmas de cómputo voluntario y computación en la nube. Se conoce como cómputo voluntario al cómputo que se realiza o que es apoyado por sistemas P2P. Este tipo de cómputo es apoyado por recursos de cómputo voluntariamente compartidos o donados por propietarios de computadoras, las cuales sirven como poder de cómputo y almacenamiento. Este tipo de computación está detrás de la propuesta de Cloud@Home para apoyar a la computación distribuida.

Los autores creen que el paradigma de computación en la nube es también aplicable a escalas inferiores, desde el usuario que contribuye de manera individual hasta los grupos de investigación, comunidades sociales, y pequeñas y medianas empresas.

Figura 12.5. Arquitectura de Cloud@Home [Cunsolo *et al.*, 2009]



La arquitectura Cloud@Home básica está formada por tres capas jerárquicas (ver figura 12.5): interfaz (front-end), capa virtual y capa física. Los usuarios pueden interactuar con la nube a través de una computadora huésped de consumidores después de haber sido autenticados por el sistema. Para implementar las funciones de Cloud@Home, los autores dividen la estructura del núcleo de todo el sistema en subsistemas llamados subsistemas de gestión y de recursos. Cloud@Home es una referencia inicial a la computación en la nube basada en la computación voluntaria y abre la puerta del cómputo en nube a cualquier usuario o comunidad pequeña.

EJERCICIOS

1. ¿Cuál es el beneficio de usar cómputo en la nube?

2. ¿Qué significa el término abstracción en cómputo en la nube?

3. Investiga algunos problemas de seguridad relacionados con el cómputo en la nube.

4. ¿Cuáles son los beneficios que introducen las redes P2P como infraestructuras para desplegar servicios de cómputo en la nube?

5. ¿Por qué la computación voluntaria puede apoyar al desarrollo del cómputo en la nube a pequeñas empresas y comunidades?

6. Explica los principales retos del cómputo en la nube.

7. Explica las diferencias entre IaaS, PaaS y SaaS.

8. Explica el concepto de una nube hibrida y desarrolla un caso posible.

9. En cierta medida el cómputo en la nube retoma la idea del paradigma del cómputo centralizado. Indica las diferencias.

10. ¿Cómo garantiza el cómputo en la nube la calidad de servicio (QoS) a los usuarios?

Actividad integradora

1. De la aplicación de los contenidos, así como de las habilidades y actitudes desarrolladas.
   Se recomienda una exposición general por parte del profesor. Además, se pueden dejar temas específicos a investigar al alumno como actividad extraclase para desarrollar y presentar una exposición la última sesión del curso. Se recomienda realizar algún prototipo básico de cómputo en la nube que puede servir como proyecto final.
   El alumno debe de mostrar un gran interés por el diseño y programación de infraestructuras de redes superpuestas que puedan servir de plataforma para el cómputo en la nube.
2. Del logro de los objetivos establecidos en el capítulo o apartado del material.
   Se espera alcanzar los objetivos de este capítulo con el apoyo en la solución de los ejercicios, la exposición del profesor, el desarrollo de un prototipo básico como proyecto final y la exposición de los alumnos en la última sesión de clases.

Glosario

API (Application Program Interface) - Interfaz de Programación de Aplicaciones: Es una interfaz de programación bien definida y conocida que permite manipular un sistema, aplicación o proceso de manera sencilla.

Applet: Parte de una aplicación que se ejecuta en el contexto de otro programa.

ASCII (American Standard Code for Information Interchange) - Código Estándar Estadounidense para el Intercambio de Información: Código de 128 caracteres basado en el alfabeto latino, incluye símbolos y caracteres de control.

Autenticación: Procedimiento para verificar la identidad de una persona.

ATM (Asynchronos Transfer Mode) - Modo de Transferencia Asíncrona: Tecnología de telecomunicaciones definido para red de banda ancha que organiza la información en celdas de tamaño fijo de 53 bytes.

Banda ancha: Característica usada en telecomunicaciones para indicar que una red tiene alta capacidad para transportar información a alta velocidad. Por ejemplo, soporta multimedia interactiva.

Bps (bit per second): Abreviación usada para bit por segundo, se acostumbra usarla como bit/s, kbit/s o Mbit/s.

Byte: Conjunto de 8 bits que es tratado como una unidad.

Certificado digital: Archivo que contiene información para identificar a su propietario.

Circuito virtual: Llamada reconocida por la red en una red de conmutación de paquetes.

Clave criptográfica: Parámetro usado por un algoritmo para cifrar y descifrar datos o comprobar una firma digital adjunto a un mensaje.

Compresión: Proceso para eliminar la redundancia en cierta información para ocupar menor espacio de almacenamiento o ancho de banda.

Contraseña: Código secreto usado por un usuario al conectarse a un sistema de información.

Criptografía: Disciplina que estudia la transformación de datos para ocultar información contenida en estos, alteración o transmisiones no autorizadas.

Datagrama: Forma de enrutamiento usada en las redes de conmutación de paquetes donde son enviados los datos a su destino sin necesidad de establecer un circuito virtual.

Digital: Capacidad de procesar información de un sistema en función de dos valores: 0 y 1.

Encriptación: Procedimiento para hacer ilegible un mensaje y evitar que sea leído por personas no autorizadas.

Firewall – Cortafuegos: Elemento de seguridad usado para aislar la red de una organización de otras redes externas (por ejemplo, Internet) para hacerla más segura.

Firma electrónica: Datos electrónicos que tienen el propósito de sustituir la firma manuscrita en documentos digitales, lo que garantiza autoría y autenticidad del documento firmado.

Protocolo de Transferencia de Archivo (FTP): Protocolo de red para transferencia de archivos entre diferentes sistemas conectados en una red y basado en el modelo cliente-servidor.

Interfaz: Punto de conexión funcional entre dos dispositivos o sistemas que permite la comunicación entre distintos niveles.

Interfaz de usuario: Medio a través del cual un usuario se puede comunicar o interactuar con una máquina o sistema.

IP (Internet Protocol) - Protocolo de Internet: Protocolo de la capa de red en el modelo OSI que contiene la dirección de los paquetes y los envía a través de la red.

Internet: Red global compuesta por miles de redes de comunicación interconectadas que están distribuidas en todo el mundo y soportada por el protocolo TCP/IP.

LAN (Local Area Network) - Red de Área Local: Red local que permite conectar diferentes dispositivos como computadoras y servidores a una alta velocidad.

Modelo OSI. Modelo que permite la interconexión de redes abiertas, regida por la Organización de Estándares Internacionales (ISO).

Multimedia: Término que agrupa a una variedad de contenidos, como texto, gráficos, imágenes, sonido y video, pero que se maneja como entidad única.

Browser – Navegador: Software usado en Internet para visualizar contenidos y aplicaciones soportadas por tecnología web.

Paquete: Conjunto de bits de datos y control que se transmiten en bloques y que cuentan con la información necesaria para alcanzar a su destino.

Protocolo: Reglas que permiten establecer la manera en que se realizará la comunicación entre dos sistemas en todos sus niveles, desde el inicio hasta la finalización.

Red: Conjunto de recursos (hardware, software, etc.) interconectados entre sí que interactúan para satisfacer los requerimientos de los usuarios.

Retardo: Retraso que incide en la transmisión de la información por la red debido a causas diversas, tales como congestión, propagación, etc.

RTT (Round-Trip Time) - Tiempo de Ida y Vuelta: Tiempo empleado por una señal para viajar del emisor al receptor y retornar.

Síncrono: Modo de transmisión de datos que está sincronizado con una base de tiempo.

Streaming: Flujo de datos del tipo audio o video que no requiere ser almacenado en el receptor y que se transmite en tiempo real con respecto al emisor considerando el retardo de la red.

TCP/IP (Transmision Control Protocol/Internet Protocol) - Protocolo de Control de Transmisión/ Protocolo de Internet: Familia de protocolos de red que se basa en Internet para la comunicación abierta entre diferentes sistemas.

Trama: Conjunto de datos equivalente a un bloque en ciertos protocolos de comunicación.

UDP (User Datagram Protocol) - Protocolo de Datagrama de Usuario: Protocolo orientado a la transmisión de datagrama, está soportado por el protocolo IP y no requiere establecer una conexión virtual para el intercambio de datos.

XML (eXtensible Markup Language) - Lenguajes de Marcas Extensible: Permite estructurar e intercambiar información entre sistemas de una manera segura.

REFERENCIAS

Adve, S. & Hill, M. (1990). *Weak Ordering: A New Definition*, 17th Annual International Symposium on Computer Architecture, pp.2-14, Seattle, WA, USA.

Akami (2014). Web: www.akami.com. Accedido: 5 de Octubre de 2014.

Amazon Web Services, Inc (2014) *Amazon Elastic Load Balancer*. Web: http://aws.amazon.com/elasticloadbalancing/. Accedido: 5 de Octubre de 2014.

Apostolopoulos, J. G., Tan, W. T. & Wee, S. J. (2002). *Video Streaming: Concepts, Algorithms, and Systems*, Technical Report HPL-2002-260, HP Laboratories Palo Alto.

Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R. H., Konwinski, A., Lee, G., Patterson, D. A., Rabkin, A., Stoica, I. & Zanaria, I. (2009). *Above the Clouds: A Berkeley View of Cloud,* Technical Report No. UCB/EECS-2009-28, University of California at Berkeley, Berkeley, CA, USA.

Babaoglu, O., Marzolla, M. & Tamburini, M. (2012). *Design and implementation of a P2P cloud system*, 27th Annual ACM Symposium on Applied Computing (SAC'12), pp. 412–417, Trento, Italy.

Banerjee, S., Bhattacharjee, B. & C. Kommareddy, C. (2002). *Scalable Application Layer Multicast, Proceedings of the ACM SIGCOMM'02,* pp. 205-217, Pittsburgh, PA, USA.

Berstis, V. (2005). *Fundamentals of Grid Computing,* IBM Corporation, Redbooks paper.

Black, U. (1999). *Tecnologías Emergentes para Redes de Computadoras,* Prentice Hall.

Black, U. (1993). *Data Communications and Distributed Networks*, Third Edition, Prentice Hall.

Chang, E. & Roberts, R. (1979). An improved algorithm for decentralized extrema-finding in circular configurations of processes, *Communications of the ACM (ACM)*, 22 (5): 281–283.

Chu, Y., Rao, S. G., Seshan, S. & Zhang, H. (2002). A Case for End System Multicast, *IEEE Journal on Selected Areas in Communications*, 20 (8), 1456-1471.

Cohen, B. (2003*). Incentives Build Robustness in BitTorrent*, Proceedings of the 1st Workshop on Economics of Peer-to-Peer Systems, Berkeley, CA, USA.

Comer, E. (1997). *Redes Globales de Información con Internet y TCP/IP, Volumen II*, Prentice Hall México.

Comer, E. (1997). *Redes de Computadoras, Internet e Interredes,* Prentice Hall Hispanoamericana.

Coulouris, G., Dollimore, J. & Kindeberg, T. (2012). *Distributed Systems, Concepts and Design,* Fifth Edition, Addison Wesley.

Coulouris, G., Dollimore, J. & Kindeberg, T. (2001). *Sistemas Distribuidos, Conceptos y Diseño,* Tercera Edición, Pearson, Addison Wesley.

Cunsolo, V., Distefano, S., Puliafito, A. & Scarpa, M. (2009). *Cloud@home: Bridging the gap between volunteer and cloud computing,* 5th International Conference on Emerging Intelligent Computing Technology and Applications (ICIC'09), Ulsan, South Korea.

Deering, S. E. (1988). *Multicast Routing in Internetwork and Extended LANs*. Proceedings of the ACM SIGCOMM'88, pp. 55-64, Stanford, CA, USA.

Flanagan, D. (1998). *Java en pocas palabras, Referencia al instante*, 2ª edición, McGraw-Hill, O'Reilly.

Forouzan, B. A. (2008). *Cryptography and Network Security*, McGraw-Hill International Edition.

Ford, M., Lew, K. H., Spanier, S. & Stevenson, T. (1998). Tecnologías de Interconectividad de Redes, Prentice-Hall Hispanoamericana.

Foster, I. & Kesselman, C. (1999). Computational Grids. Foster, I. & Kesselman, C. Eds. *The Grid: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann, pp. 2-48.

Foster, I. & Kesselman, C. (2013). *The History of the Grid*, in *Cloud Computing and Big Data*, IOS Press, Amsterdam, 2013; 37 pages, 176.

Furth, B. (2010). Chapter 1: Cloud Computing Fundamentals, in Handbook of Cloud Computing; *Springer*, pp. 3-10, London, UK.

Gao, W. & Huo, L. (2007). *Challenges on Peer-to-Peer Live Media Streaming*, International Workshop on Multimedia Content Analysis and Mining, pp. 37-41, Weihai, China, 4577/2007.

H.264/MPEG-4 AVC Reference Software Manual (JVT-AE010). (2009). Web: http://iphome.hhi.de/suehring/tml/. Accedido: 6 de Agosto de 2014,

Hadzilacos, V. & Toueg, S. (1994). *A Modular Approach to Fault-tolerant Broadcasts and Related Problems*. Technical Report, Department of Computer Science, University of Toronto.

HHI. (2010). The Scalable Video Coding Amendment of the H.264/AVC Standard. Web: http://www.hhi.fraunhofer.de/fields-of-competence/image-processing/research-groups/image- Video-coding/svc-extension-of-h264avc.html. Accedido: 7 de Octubre de 2014.

Hu, W. (1995). DCE Security Programming; O'Reilly & Associates, Inc.

Huidobro Moya, J. M. y Roldan Martínez, D. (2006). *Comunicaciones en Redes WLAN*, Limusa- Noriega Editores, México.

Huidobro Moya, J. M. (2006). *Redes y Servicios de Telecomunicaciones*, Thompson-Paraninfo.

ISO/IEC JTC 1. (1993). *Coding of Moving Pictures and Associated Audio for Digital Storage Media at up to about 1.5 Mbit/s Part 2: Video*. ISO/IEC 11172-2 (MPEG-1).

ISO/IEC JTC 1. (1999). *Coding of Audio-Visual Objects - part 2: Visual. ISO/IEC 14496-2 (MPEG-4 Part 2)*, January 1999 (with several subsequent amendments).

ITU-T. (1993). *Video Codec for Audiovisual Services at 64 kbit/s. ITU-T Recommendation H.261, Version 1: November 1990; Version 2: March 1993.*

ITU-T & ISO/IEC JTC 1. (1994). *Generic Coding of Moving Pictures and Associated Audio Information Part 2: Video. ITU-T Recommendation H.262 ISO/IEC 13818-2 (MPEG-2), November 1994 (with several subsequent amendments).*

ITU-T. (1995). *Video coding for low bit rate communication, ITUT Recommendation H.263; version 1, Nov. 1995; version 2, January 1998; version 3, November 2000.*

Jacob, B., Brown, M., Fukui, K. & Trivedi N. (2005). *Introduction to Grid Computing*, IBM Corporation. ReedBooks, 2005.

Jamsa, K. & Cope. K. (1996). *Programación en Internet*, McGraw-Hill, 1996.

Kerberos (2014). *Kerberos: The Network Authentication Protocol.* Massachusetts Institute of Technology. Web: http://web.mit.edu/kerberos/. Accedido: 20 de Octubre de 2014.

Kontothanassisy, L., Sitaramanz, R., Weinz, J., Hongz, D., Kleinberg, R., Mancusoz, B., Shawz, D. & Stodolsky, D. (2004). A Transport Layer for Live Streaming in a Content Delivery Network. *Proceedings of the IEEE*, 92(9), 1408-1419.

Kostic, D., Rodriguez, A., Albrecht, J. & Vahdat, A. (2003). *Bullet: High Bandwidth Data Dissemination Using and Overlay Mesh*, 19th ACM Symposium on Operating Systems Principles, pp. 282-297, Bolton Landing, NY, USA.

Lakshman, A. & Malik, P. (2009). *Cassandra - A Decentralized Structured Storage System*. Workshop on Large-Scale Distribued Systems and Middleware (LADIS'09), Big Sky, MT, USA.

Lamport, L., Shostak, R. & Marshall, P. (1982). The Byzantine Generals Problem, *ACM Transactions on Programming Languages and Systems*, 4 (3), pp. 382-401.

Lann. G. L. (1977). *Distributed Systems-towards a formal approach*, in IFIP Congress, pp. 155-160.

Li, K. [1986]. *Shared virtual memory on loosely coupled multiprocessors*. Ph.D. dissertation, Department of Computer Science, Yale University, New Haven, Conn., USA. (También: Technical Report YALEU-RR- 492).

Liu, M. L. (2004). *Computación Distribuida: Conceptos y Aplicaciones*. Addison Wesley.

Lin, M., Hsieh, J., Du, D. H. C., Thomas, J. P. & MacDonald J. A. (1995). Distributed Network Computing over Local ATM Networks, *IEEE Journal on Selected Areas in Communications*, 13(4), pp. 733-748.

López-Fuentes, F. A. (2009). *Video Multicast in Peer-to-Peer Networks,* Verlag Dr. Hunt, Munich, Germany.

Magharei, N. & Rejaie, R. (2006). Adaptive Receiver-Driven Streaming from Multiple Senders, *ACM/Springer Multimedia Systems Journal* 11(6), 1-18.

Magharei N. & Rejaie, R. (2006b). *Understanding Mesh based Peer to Peer Streaming,* 16th International Workshop on Network and Operating Systems Support for Digital Audio and Video, Newport, RI, USA.

Magharei, N., Rejaie, R., & Guo, Y. (2007). Mesh or Multiple-Tree: A Comparative Study of Live P2P Streaming Approaches, *The IEEE INFOCOM 2007*, pp. 1424-1432, Anchorage, Alaska, USA.

Marozo, F., Talia, D. & Trunfio, P. (2010). Chapter 7: A Peer-to-Peer Framework for Supporting MapReduce Applications in Dynamic Cloud Environments, Cloud Computing Principles, Systems and Applications; *Springer*, pp. 113-125, London, UK.

Marpe, D., Wiegand, T. & Sullivan, G. J. (2006). The H.264/MPEG4 Advanced Video Coding Standard and its Applications, *IEEE Communications Magazine,* pp.134-142.

Martínez Gomaríz E. (2014). *Diseño de Sistemas Distribuidos,* Universidad Politécnica de Cataluña. Web: http://www.essi.upc.edu/~gomariz/index_archivos/IntroduccionSD-EnricMartinez.pdf. Accedido: 15 Octubre 2014.

Mell, P. & Grance, T. (2011). *The NIST Definition of Cloud Computing*, Special publication 800-145 (draft), Gaithersburg, MD, USA.

Menezes, A. Van Oorschot, P. & Vanstone. S. (1996). *Handbook of Applied Cryptography*, CRC Press.

OASIS [2014]. Organization for the Advancement of Structured Information Standards, OASIS Committee Categories: Web Services, Web: https://www.oasis-open.org/committees/tc_cat.php?cat=ws. Accedido: 18 de Octubre de 2014.

Özsu, M. T. & Valduriez, P. (1999). *Principles of Distributed Database Systems*. Prentice Hall.

Patterson, D. A. & Hennessy, J. L. (1994). *Computer Organization & Design, The Hardware/Software Interface*, Morgan Kaufmann Publisher, Inc., San Mateo, CA, USA.

Pendarakis, D., Shi, S., Verma, D. & Waldvogel, M. (2001). *ALMI: An Application Level Multicast Infrastructure.* Third USENIX Symposium on Internet Technologies and Systems, pp. 49-60, San Francisco, CA, USA.

PPlive [2014]. Web: http://www.pptv.com. Accedido: 20 de Octubre de 2014.

P2P-Next [2014]. Web: http://www.p2p-next.org. Accedido: 18 Septiembre de 2014.

Ranjan, R., Zhao, L., Wu, X., Liu, A., Quiroz, A. & Parashar, M. (2010). Chapter 12; Peer-to-Peer Cloud Provisioning: Service Discovery and Load-Balancing. Cloud Computing Principles, Systems and Applications; *Springer*, pp. 195-217, London, UK.

Rajkumar, B. & Bubendorfer, K. (2009). *Market Oriented Grid and Utility Computing*. Wiley.

Richardson, I. (2007). *An Overview of H.264 Advanced Video Coding*. Vcodex white paper.

Ripeanu, M., Foster, I., Iamnitchi, A. & Rogers, A. (2007). *In Search for Simplicity: A Self-Organizing Multi-Source Multicast Overlay*, 1st IEEE International Conference (SASO'07), pp. 371-374, Boston, MA, USA.

Rowstron & P. Druschel (2001). *Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems.* IFIP/ACM International Conference on Distributed Systems Platforms (Middleware), pp. 329-350, Heidelberg, Germany.

Robbins, K. A. & Robbins, S. (1997). *Unix Programación Práctica, guía para la concurrencia, la comunicación y los multihilos.* Prentice-Hall Hispanoamericana.

Saltzer, J. H. & Schroeder, D. M (1975). The Protection of Information in Computer Systems, *Proceedings of the IEEE*, Vol. 63, No.9, pp. 1278-1308.

Sánchez, J.; Sistemas Distribuidos, Material de Apoyo por Sesiones, Dirección de Educación a Distancia, ITESM-CEM, 1995.

Schneier, B. (1996). *Applied Cryptography: Protocols, Algorithms, and Source Code in C,* second edition, John Wiley & Sons, Inc. NY, USA.

Schwarz, H., Marpe, D. & Wiegand, T. (2007). Overview of the Scalable Video Coding Extension of the H.264/AVC Standard. *IEEE Transactions on Circuits and Systems for Video Technology*, Special Issue on Scalable Video Coding, 17(9), pp. 1103-1120.

Sosinsky, B. (2011). *Cloud Computing Bible,* Wiley Publishing Inc., Indianapolis, IN, USA, 2011.

Stalling, W. (2011). *Comunicaciones y Redes de Computadores,* Pearson.

Stalling, W. (2004). *Fundamentos de Seguridad en Redes,* 4ª. Edición, Pearson-Prentice-Hall.

Tanenbaum, A. S (1996). *Sistemas Operativos Distribuidos,* Prentice Hall Hispanoamericana.

Tanenbaum, A. S (1997). *Redes de Computadoras,* 3ª. Edición, Pearson.

Tanenbaum, A.S., Van Renesse, R., van Staveren, H., Sharp, G., Mullender, S., Jansen, J. & van Rossum, G. (1990). Experiences with the Amoeba Distributed Operating System. *Communications ACM*, Vol. 33, No. 12, pp. 46–63.

Tanenbaum, A. & Van Steen, M. (2008). *Sistemas Distribuidos: Principios y Paradigmas,* 2ª. Edición, Prentice Hall.

Tel, G. (2001). *Introduction to Distributed Algorithms,* 2a. Edition, Cambridge University Press.

Todino, G., Strang, J. & Peek, J. (1994). *Learning the UNIX Operating System.* O'Reilly & Associates, Inc.

Venot, S. & Yang, L. (2007). *Peer-to-Peer Media Streaming Application Survey,* International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies, Papeete, pp. 139-148, French Polynesia.

W3C (2014). World Wide Web Consortium, Web Services Architecture. W3C Working Group Note 11 February 2004. Web: http://www.w3.org/TR/ws-arch/. Accedido: 24 de Septiembre de 2014.

Wang, F., Xiong, Y. & Liu, J. (2007). *mTreebone: A Hybrid Tree/Mesh Overlay for Application-Layer Live Video Multicast.* 27th International Conference on Distributed Computing System, pp. 49-56, Toronto, Ontario, Canada.

Wang, Y., Ostermann, J., & Zhang, Y-Q. (2002). *Video Processing and Communications,* Prentice Hall PTR, Upper Saddle River, NJ, USA.

Wiegand, T., Sullivan, G. J., Bjontegaard, G. & Luthra, A. (2003). Overview of the H.264/AVC Video Coding Standard, *IEEE Transaction on Circuit and Systems for Video Technology*, 13(7), pp. 560-576.

Wiesmann, M., Pedone, F., Schiper, A., Kemme, B. & Alonso, G. (2000). *Understanding replication in databases and distributed systems*. 20th International Conference on Distributed Computing Systems (ICDCS 2000), p. 464, Taipei, Republic of China.

Wilson E. & Miles R. (Ed.). (2010). *Peer-to-Peer (P2P) Technologies and Services,* Technical Report 009, European Broadcasting Union (EBU), Geneva, Switzerland.

Wu, D., Hou, Y. T., Zhu, W., Zhang, Y.-Q. & Peha, J. M. (2001). Streaming Video over the Internet: Approaches and Directions. *IEEE Transactions on Circuits and Systems for Video Technology*, 11(3), pp. 282-300.

Xu, K., Song, M., Zhang, X. and Song, J. (2009). A Cloud Computing Platform based on P2P. *IEEE International Symposium on IT in Medicine Education (ITME '09)*, pp. 427-432, Shandong, China.

Zheng, W., Liu, X., Shi, S., Hu, J. & Dong, H. (2005). Peer-to-Peer: A Technique Perspective, J. Wu (Ed.), *Handbook of Theoretical and Algorithmic Aspects of Ad Hoc, Sensor, and Peer-to-Peer Networks*, Auerbach Publications, pp. 591-616, Boca Raton, FL.

Sistemas distribuidos se terminó de imprimir en agosto de 2015
de forma digital en los talleres de Imprenta 1200+
Andorra 29. Colonia Del Carmen Zacahuitzco, México D.F.
Tel. (52)55218493.

El tiraje consta de 100 ejemplares de 17x24 cm, 200 páginas
cada uno, a cuatro tintas, encuadernación pegado cubierta
flexible. En su composición se utilizó la familia Avenir. Se
empleó papel reciclado de 90g para páginas interiores.