UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE ESTUDIOS SUPERIORES
CAMPUS ARAGÓN

Sistema de Archivos Distribuido
Conceptos y Ejemplo

T E S I S

QUE PARA OBTENER EL TÍTULO DE:
INGENIERO EN COMPUTACIÓN

P R E S E N T A:

Cervantes Coronado Iván Christian

ASESOR DE TESIS:
Mat. Luis Ramírez Flores

MÉXICO, 2006.

LizardKing

Esta tesis esta dedicada para Victoria, Lucia y Guadalupe.

Índice

Capitulo 1

Capitulo 2

Capitulo 3

Introducción

El cómputo distribuido es una piedra angular de la era de la información, pienso que estamos llegando al nudo de esta era actual, debido a esto los sistemas distribuidos están llegando a la cumbre de su desarrollo y aplicación, pero como ha pasado en la ciencia, en la mayor parte de las actividades y desarrollos científicos recientes no se le da su lugar, seguimiento e importancia en escuelas e institutos (descartando como es obvio las escuelas de prestigio mundial científico comprobado) esto se debe a lo limitado de la cultura científica de nuestro país lo cual en parte es provocado por la falta de reformas educativas. Esto debería cambiar, ya que la demanda incrementa rápidamente para que los científicos tengan un conocimiento profundo de los sistemas distribuidos. En el entorno básico de cómputo centralizado, un simple conocimiento de conceptos de sistemas operativos es suficiente; en contraste, los desarrolladores de entornos distribuidos no requieren únicamente conocer los conceptos de sistemas distribuidos, paralelos y de tiempo real, también necesitan implementarlos a una escala mucho mayor.

A lo largo de este camino llamado tesis definiremos y explicaremos los conceptos necesarios para comprender el cómputo distribuido y la relevancia de su existencia en esta su época... llamada la era de la información. En especial nos enfocaremos en una de sus partes vitales el “Sistema de Archivos”.

Justificación

Profesionalmente: para obtener el titulo de Ingeniero en Computación.

Académico-Científico: ser participe y cooperar en un de los temas de mayor importancia en la ciencia computacional actual.

Personal y Social: finalizar un proyecto en el cual invertí 5 años de mi vida, en el que fui apoyado, y por lo cual espero este trabajo apoye a las mentes del mañana.

Delimitación del Tema

El tema abarcará el marco teórico, el cual comprende conceptos generales y el ejemplo del sistema de archivos distribuido.

Objetivo General

Generar una referencia de conceptos fundamentales de cómputo distribuido en especial del sistema de archivos y desarrollar un ejemplo con enfoque didáctico.

Objetivo Particular

- Ayudar a futuras generaciones de Ingeniería en Computación en el área de cómputo distribuido.
- Conceptualizar cada uno de los tópicos y organizarlos de forma integral.
- Apoyar la generación de proyectos científicos e investigación en los temas presentados.
- Ejemplificar de forma didáctica algunos conceptos presentados.

1

Introducción a los Sistemas Operativos Distribuidos

1.1 ¿Qué es un sistema operativo?

Un sistema operativo es el administrador de la computadora es decir, es el responsable que controla, regula, coordinada, y calendariza (schedule) todos los procesos y la utilización de sus recursos, algunos de ellos son:

- CPU(s)
- Modulo(s) de memoria
- Medios de almacenamiento (CD-ROMS, DVD, etc.)
- Tarjetas de audio y video
- Redes
- Modem y adaptadores de red
- Monitores, terminales, impresoras, y otros dispositivos de entrada
- Teclados, scanner's, y otros dispositivos de salida

Algunas de las tareas que tiene que administrar el sistema operativo son las siguientes:

- Administración de procesos (u objetos)
- Comunicación
- Administración de acceso de memoria
- Calendarización de recursos
- Seguridad de información y recursos
- Integridad de los datos

Tradicionalmente, todas estas tareas administrativas del sistema operativo fueron preocupación del software. Aunque esto no es necesariamente cierto en un sistema distribuido.

1.2 ¿Qué es un sistema distribuido?

Un **sistema distribuido** es una colección de computadoras heterogéneas y procesadores conectados por medio de una red de computadoras[1]. Esta colección trabaja muy cercana y conjuntamente para realizar un objetivo Figura 1.1.

El objetivo de un sistema operativo distribuido es el de proveer una vista global común y consistente de un sistema de archivos, tiempo, seguridad, y acceso a los recursos distribuidos. Para proveer esta vista común, hay un número grande de problemas y requerimientos en todos los sistemas participantes; los sistemas distribuidos son referidos generalmente como una conexión de sistemas estrechamente acoplados. Si las computadoras heterogéneas son conectadas por una red y no están estrechamente acoplados, ellos son generalmente referidos como un **sistema de red**. Un sistema de red no pretende como objetivo proveer una vista global común, no si esto significa una demanda (mayor de la establecida) de recursos de los sistemas. Los componentes en un sistema

[1] Galli (2000). *Distributed Operating Systems – Concepts and Practice*

distribuido o de red tal vez sean simplemente componentes de sistemas centralizados; hay otro tipo de componentes como los de tiempo real y paralelos.

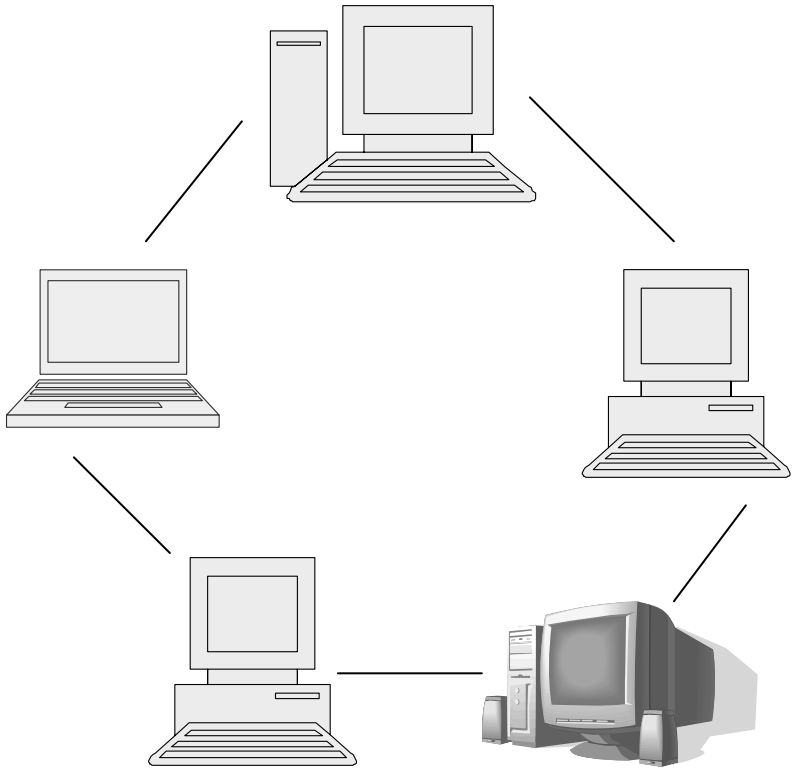


Figura 1.1 Diagrama de un sistema distribuido.

Es común para un sistema distribuido incorporar componentes centralizados, de tiempo real y paralelos al mismo tiempo.

Ambos sistemas distribuidos y de red tienden a crecer en popularidad debido al rápido incremento de poder computacional por dólar.

1.2.1 Topologías de red y sus características[2]

Para clasificar las redes tradicionalmente se consideran dos grandes categorías: las redes de área amplia (WAN, *Wide Area Networks*) y las redes de área local (LAN, *Local Area Networks*). Las diferencias entre estas dos categorías son cada vez más difusas, tanto en términos tecnológicos como de posibles aplicaciones; sin embargo es una forma natural y didáctica de organizar su estudio, por lo que aquí se adoptará dicha clasificación.

[2] Stallings (2004). *Comunicaciones y redes de computadores*

Los sistemas distribuidos utilizan redes de área local (LAN), redes de área amplia (WAN) e interredes para comunicarse. Las características y comportamiento de éstas influyen en el comportamiento de los sistemas distribuidos. Las necesidades de los usuarios han cambiado produciendo una adopción mayor de redes inalámbricas y las redes de altas prestaciones con calidad de servicio garantizada.

LAN

Las redes de área local (*LAN*) son típicamente propiedad de una sola entidad y abarcan únicamente unos pocos kilómetros de rango, llevan mensajes a velocidades relativamente altas entre computadoras conectadas a un único medio de comunicación: un cable de par trenzado, un cable coaxial o uno de fibra óptica. Un *segmento* es una sección de cable que da servicio a un departamento o a una planta de un edificio y que puede tener varias computadoras conectadas. Dentro de un segmento no es necesario *encaminar (routing)* mensajes ya que el medio físico nos proporciona una conexión directa entre las computadoras conectadas a el. En las redes de área local, el ancho de banda total del sistema es grande y la latencia mínima, a excepción de cuando el tráfico de mensajes es demasiado alto; un par de tecnologías para implementar LAN son: *Ethernet* y *token ring*.

- Ethernet es similar a poner un sello a una caja de regalo, mientras que token ring es similar a mandar un paquete con una confirmación de recibido. El Ethernet básico corre a 10 Mbps IEEE 802.3 con populares variaciones como fast Ethernet que corre a 100 Mbps y el Gigabit Ethernet que corre a 1000 Mbps, y no provee algún mecanismo para conocer si el mensaje fue recibido.

- El token ring IEEE 802.5, provee un mecanismo para conocer si todos los mensajes fueron recibidos. También asigna prioridades y garantiza el tiempo de envío lo cual lo hace particularmente atractivo para las aplicaciones en tiempo real. Al contrario de Ethernet no colapsa bajo cargas altas de trabajo.

- Interfaz de datos de fibra distribuida (*FDDI*) es otro protocolo de red común. Es normalmente utilizado como un *backbone* o como una LAN de alta velocidad.

WAN

De forma general se consideran como redes de área amplia a todas aquellas que cubren una extensa área geográfica, requieren atravesar rutas de acceso público y utilizan, por lo menos parcialmente, circuitos proporcionados por una entidad proveedora de servicios de telecomunicación. El medio de comunicación está compuesto por un conjunto de circuitos enlazados mediante dispositivos, llamados *routers* o encaminadores.

Muchos de los sistemas distribuidos no están concentrados en una LAN pero si en una WAN. Para conectar una LAN a una WAN, un *gateway* debe ser empleado. La información es dividida en pequeñas piezas conocidas como frames o paquetes, antes de que sean enviadas.

Protocolos populares utilizados en WAN incluyen a frame relay y al modo de transferencia asíncrona (*ATM*). Frame relay permite que los mensajes de red incluyan información acerca del control de congestión de la red y también si ese mensaje debería ser descartado si la red esta en un estado de transmisión muy lento. Las redes ATM son conocidas por su velocidad y soporte de transmitir archivos multimedia, esta característica no es única a las redes ATM. En ATM los paquetes se denominan *celdas*, una celda ATM tiene una cabecera de 5 bytes y un campo de datos de 48 bytes. La capa de adaptación de ATM (*AAL*) soporta un porcentaje constante de bits y un porcentaje variable de tráfico de bits.

La conexión de los dispositivos portátiles y de mano necesitan redes de comunicación inalámbricas (*Wireless Networks*). Algunas de las tecnologías de comunicación son: IEEE 802.11 (*WaveLan*) ofrecen transmisiones de datos entre 2 y 11 Mbps sobre 150 metros, estas redes son en realidad LAN inalámbricas (*WLAN*). Las redes de área personal inalámbricas (Wireless Personal Area Networks, WPAN) están diseñadas para conectar dispositivos móviles a otros dispositivos móviles o fijos muy próximos, un ejemplo sería comunicarse con impresoras, computadoras de mano o de escritorio, enlaces infrarrojos que incluyen muchas computadoras de mano y portátiles y la tecnología de radio de baja potencia BlueTooth. Varias redes de telefonía móvil están basadas en tecnologías de redes inalámbricas, incluida la red europea mediante el Sistema Global para Comunicaciones Móviles (GSM). La tabla siguiente muestra los rangos, ancho de banda y latencia de las redes vistas anteriormente[3]:

	Rango	***Ancho de Banda (Mbps)***	***Latencia (ms)***
LAN	**1 – 2 Km.**	**10 – 1000**	**1 – 10**
WAN	**mundial**	**0.010 - 600**	**100 – 500**
MAN	**2 – 50 Km.**	**1 – 150**	**10**
LAN inalámbrica	**0, 15 – 1,5 Km.**	**2 – 11**	**5 – 20**
WAN inalámbrica	**mundial**	**0.010 – 2**	**100 – 500**
Internet	**mundial**	**0.010 – 2**	**100 – 500**

Tabla 1.1 Rango, Ancho de Banda y Latencia de varios tipos de Red.

[3] Coulouris, Dollimore y Kindberg (2005). *Distributed Systems Concepts and Design*,

Las redes pueden ser conectadas en una de las siguientes maneras:

1. **Repetidor**: dispositivo que simplemente repite cualquier cosa de un lado de la red al otro lado de la red. Ambas redes deben emplear el mismo protocolo[4].
2. **Puente**: dispositivo que enlaza redes de distintos tipos. Algunos puentes comunican varias redes, y se les denomina puentes/routers por que también efectúan funciones de encaminamiento.
3. **Router** (*encaminador*): un router es aun más avanzado que un puente en cuanto a que también conecta segmentos de LAN que posiblemente emplean diferentes protocolos. Los routers son capaces de conectar más de dos redes al mismo tiempo.
4. **Gateway**: realiza todo lo que hacen los anteriores, puede cambiar de protocolo de comunicación, además dos redes distintas se puedan unir con este dispositivo (por ejemplo una red LAN y una WAN).
5. **Backbone**: LAN que no contiene usuarios en la red pero tiene otras redes, como vemos en la siguiente figura.

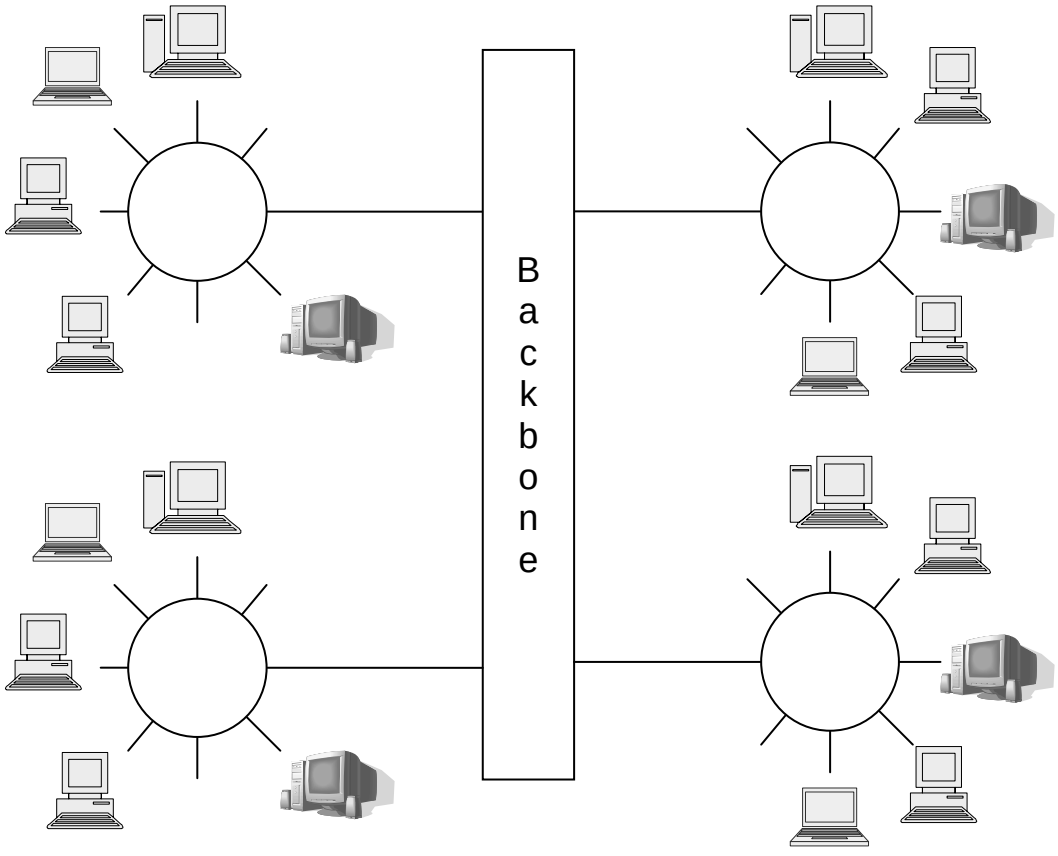


Figura 1.2 Diagrama de una conexión de red tipo Backbone

[4] Protocolo, se utiliza para referirse a un conjunto bien conocido de reglas y formatos que se han de utilizar para la comunicación entre procesos que realizan una tarea determinada.

1.2.2 Modelo de Referencia ISO/OSI[5]

El Modelo de Referencia para la *Interconexión de Sistemas Abiertos (OSI)* de siete capas adoptado por la Organización Internacional de Estándares (*ISO*), es un modelo cuya principal función es favorecer el desarrollo de estándares de protocolos que satisfagan los requisitos de los sistemas abiertos. En la siguiente figura 1.3 se muestran las siete capas del modelo ISO/OSI.

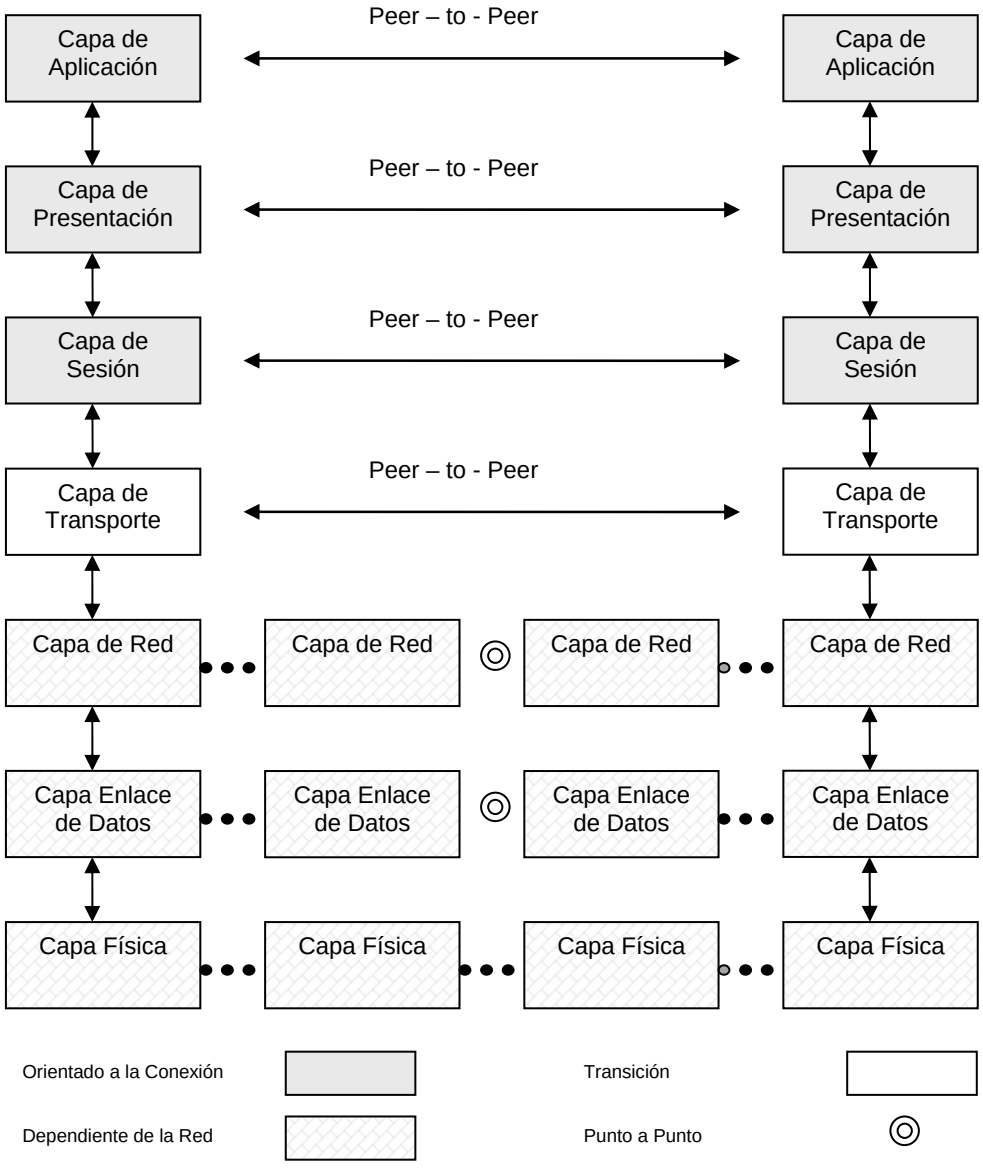


Figura 1.3 Capas del modelo de referencia ISO/OSI

[5] Es una arquitectura de protocolos.

La descripción de cada una de las capas es:

1. **Capa física**: los circuitos y el hardware que dirigen la red. Transmite secuencias de datos binarios mediante señales binarias, utilizando modulación en amplitud o en frecuencia de las señales eléctricas (en los circuitos de cables), señales ópticas (en los circuitos de fibra óptica) u otras señales electromagnéticas (microondas); esta capa es de particular interés para los de ingeniería eléctrica.

2. **Capa de enlace de datos**: es responsable de la transmisión de paquetes entre nodos que están conectados directamente por un enlace físico. En una transmisión WAN será entre pares de routers o entre un router y un *host*.

3. **Capa de red**: su función dominante es la de encaminar un mensaje de la fuente hacia el destino. Debido a que encaminar no es necesario en LAN, esta capa es nula en ese entorno y no realiza ninguna funcionalidad.

4. **Capa de transporte**: la función primaria de la capa de transporte es determinar que clase de servicio se necesita para la comunicación sobre la red. Una clase de servicio es considerado *orientado a la conexión* y provee control total de errores y servicios de confirmación. Otra de las clases es *no orientado a conexión* y no provee los servicios del antes mencionado.

5. **Capa de sesión**: su función tiene que ver con la organización y sincronización entre la fuente y el destino hasta el final de su comunicación.

6. **Capa de presentación**: transmitir datos en una representación de datos de red independiente de las utilizadas comúnmente en las computadoras, que pueden ser de distinto tipo. Si se necesitara, el *cifrado* se realizaría en esta capa.

7. **Capa de aplicación**: protocolos diseñados para responder a los requisitos de comunicación de aplicaciones específicas, comúnmente definiendo la interfaz a un usuario.

Las capas de la 1 a la 3 son dependientes de la red, figura 1.3. Estas capas están altamente implicadas con el protocolo exacto de red con el cual el sistema esta directamente comunicándose. Las capas de la 5 a la 7 son orientadas a aplicación, mientras que la capa 4 es una transición entre la red y la aplicación. Los conceptos de sistemas operativos comienzan en la interfaz de la capa 4 e implican de la capa 5 a la 7.

Otra característica importante del modelo de referencia ISO/OSI es el concepto de capas que son *punto a punto (point to point)* contra las capas *de igual a igual* (peer *to peer)*. Las funciones en un punto a punto son realizadas en todas y cada una de las ubicaciones entre la fuente y el destino, como también en la fuente y el destino.

Debido a que tal vez hay miles de *nodos* entre la fuente y el destino, las funciones punto a punto tal vez se realicen miles de veces mientras un mensaje viaja desde su fuente hacia su destino. Todas las funciones definidas de la capa 1 a la 3 son punto a punto. Las funciones en las capas peer to peer que van de la 4 a la 7 son realizadas únicamente dos veces, una en la fuente y otra en el destino.

1.2.3 TCP/IP

Las dos arquitecturas que han sido dominantes y básicas en el desarrollo de los estándares de comunicación son el conjunto de protocolos TCP/IP y el modelo de referencia OSI citado anteriormente. Sin duda TCP/IP es la arquitectura más adoptada para la interconexión de sistemas, mientras que por otro lado OSI se ha convertido en el modelo estándar para clasificar las funciones de comunicación.

TCP/IP es el resultado de la investigación y desarrollo realizado en la red experimental de conmutación de paquetes (ARPANET), siendo financiada esta red por la Agencia de Proyectos de Investigación Avanzada para la Defensa (DARPA), y es conocido globalmente como la familia de protocolos TCP/IP. Esta familia consiste en una extensa colección de protocolos que se han convertido estándares de Internet (cuando el termino internet es escrito con ' I ' mayúscula, este se refiere a la Internet mundial o simplemente Internet).

Todas las tareas involucradas en la comunicación pueden ser organizadas en cinco capas:

1. Capa de aplicación.
2. Capa origen-destino o de transporte.
3. Capa internet.
4. Capa de acceso a la red.
5. Capa física.

La **capa física** define la interfaz física entre el dispositivo de transmisión de datos (por ejemplo: la estación de trabajo o la computadora) y el medio de transmisión o red. Esta capa se encarga de la especificación de las características del medio de transmisión, la naturaleza de las señales, la velocidad de datos, y cuestiones afines.

La **capa de acceso a la red** es responsable del intercambio de datos entre el sistema final y la red a la que se está conectando. El emisor debe proporcionar la dirección del destino a la red, de tal manera que la red pueda encaminar los datos hasta el destino apropiado.

La **capa Internet** utiliza el protocolo internet (IP) para dar el servicio de encaminamiento a través de varias redes, también incluye otros protocolos como son el protocolo internet de mensajes de control (ICMP).

La **capa de transporte** reside al Protocolo de Control de transmisión (TCP) y el Protocolo de Datagrama de Usuario (UDP). TCP provee transmisiones fiables. UDP por otro lado, sólo manda paquetes y deja que capas superiores se preocupen acerca de la fiabilidad. Se sabe que UDP es mucho más eficiente y es algunas veces el método preferido para aplicaciones en tiempo-real.

La **capa de aplicación** incluye el servicio de FTP (Protocolo de Trasferencia de Archivos), correo electrónico a través de SMTP (Protocolo Simple de Transferencia de Correo) y otros servicios.

1.2.4 Modelos de cómputo distribuido

Un modelo de un sistema de cómputo distribuido trata sobre la colocación, administración y las relaciones de sus partes. Algunos ejemplo son: el modelo cliente-servidor y el modelo peer-to-peer. El modelo cliente-servidor es muy popular debido a su simplicidad, y el modelo peer-to-peer es un modelo más avanzado que también es popular.

Modelo cliente-servidor

Por historia es la más importante, y continúa siendo la más ampliamente utilizada. Veamos el modelo con la siguiente sustitución Cliente=Jefe y Servidor=Trabajador en un entorno laboral cotidiano. El cliente necesita que se realice una tarea y le da esa tarea al servidor, quien entonces debe realizar la tarea y notificar a su jefe que es el cliente.

El cliente hace peticiones al servidor y el servidor responde al cliente. Esta operación tiene lugar en la **capa de sesión** ISO/OSI. Es implementada a través de la *comunicación entre procesos (Interprocess Communication)*. En el modelo cliente-servidor, tal vez haya varios clientes haciendo peticiones a varios servidores. También dado un servidor tal vez satisfaga las peticiones de varios clientes. En un entorno local, hay dos métodos básicos para implementar el modelo cliente-servidor los cuales son el modelo *estación de trabajo (Workstation) y el modelo de pool (comunes) de procesadores.*

El modelo de Workstation permite que cada usuario local tenga una pequeña Workstation la cual en si misma no tiene el suficiente poder de cómputo para completar todas las tareas que el usuario pudiera requerir para ésta. Cuando más poder es necesitado, la Workstation del usuario realiza una petición a través de la red en espera de encontrar otra Workstation que le ayude, esto genera otros problemas como es la administración de procesos distribuidos.

En el modelo de pool de procesadores, cada usuario es únicamente provisto con una simple terminal y todo el poder de procesamiento lo tiene una ubicación centralizada, como por ejemplo un mainframe. Cuando se necesitan recursos, el

usuario realiza peticiones al servidor, quien controla todo el poder de procesamiento, su distribución y otros servicios.

El modelo peer-to-peer

El modelo peer-to-peer es una evolución natural del modelo cliente-servidor. Este es similar a un grupo de colegas trabajando juntos. Cada miembro del equipo de cómputo puede ser que realicen peticiones de servicio a otros miembros. Entonces, el modelo cliente-servidor puede ser una simplificación y predecesor del modelo peer-to-peer. Algunas aplicaciones ven la necesidad del modelo peer-to-peer ya que los provee de relaciones muchos-a-muchos (many-to-many relationship).

1.2.5 Diferencia de soluciones entre distribuido y centralizado

Las soluciones de varias tareas en un entorno distribuido ofrecen una opción entre soluciones distribuidas y centralizadas. Las soluciones centralizadas toman toda la decisión y toda la información relacionada para la decisión en una ubicación. Este tipo de solución es la más sencilla y es el método más fácil para adaptar algoritmos planeados para sistemas no distribuidos; de cualquier forma este tiene varias desventajas. La primera desventaja es que la autoridad central viene a ser un elemento crítico. Si este elemento crítico falla, el sistema distribuido entero es sujeto a fallar. El segundo es el tráfico de red alrededor de la autoridad central el cual se incrementa debido a que todos los participantes del sistema tienen que comunicarse con esta unidad central. Por otra parte algo positivo es la sencilla actualización de software de aplicación en una arquitectura centralizada, debido a que sólo a la unidad central se le actualiza el software.

Los sistemas distribuidos no sufren de un elemento crítico pero tienen sus propias debilidades. Soluciones distribuidas frecuentemente incrementan el tráfico a través de redes debido a que varias soluciones distribuidas utilizan broadcast[6] para transmitir información. También es más difícil para varias ubicaciones mantener la información consistente. Algunas ubicaciones tal vez reciban información, si esta se ha actualizado o si han ocurrido actualizaciones de software, esto lo harán con mayor prioridad que aquellas redes con retrasos (lentas). Las soluciones óptimas tal vez no sean posibles debido a la información incompleta. Además, las soluciones distribuidas generalmente requieren cooperación de varios participantes.

1.2.6 Sistemas operativos de red y distribuidos

Los factores primarios que distinguen un sistema operativo de red de uno distribuido son el nivel de transparencia provisto y los requerimientos que se ponen a los sistemas participantes.

[6] Es un mensaje (señal) que se envía a todos los dispositivos conectados a la red.

Requerimientos de sistema

Un sistema distribuido requiere que todos y cada uno de sus participantes corran su propia copia del mismo sistema operativo distribuido, de todas formas estos sistemas distribuidos también corren localmente encima de sistemas operativos base o sistemas operativos centralizados. Los sistemas operativos distribuidos que corren encima de un sistema operativo base son conocidos como midware (INFERNO[7]). Un sistema operativo de red no requiere que los participantes corran el mismo sistema operativo, únicamente deben ponerse de acuerdo en que protocolo de red usar para operaciones como sesión y copiado de archivos. Entonces la gran diferencia entre un sistema operativo de red y uno distribuido es que el de red no trata de esconder el hecho de que tú estas operando en diferentes sistemas. Esto es que no hace transparente operaciones remotas al usuario o no soporta cualquier de las siguientes transparencias.

Transparencia

Examinemos los varios tipos de transparencia que deben de ser soportadas por un sistema operativo distribuido.

Transparencia de nombre: el nombre del recurso no debe de indicar donde esta físicamente el archivo, datos, o procesos dentro del sistema distribuido. Además si los usuarios cambian de ubicación, su vista del sistema no debe de cambiar debido a que el nombre no esta casado con una ubicación especifica. De cualquier forma esto implica algún ordenamiento de un esquema global de nombre.

Transparencia de ubicación: cuando la transparencia de ubicación es provista, los usuarios no pueden decir donde están físicamente los recursos que se están utilizando. Esto implica el soporte de transparencia de nombre y transparencia de acceso. Esto es benéfico cuando hablamos de archivos y procesos, asumiendo que la utilización de éstos no tiene costo adicional. Sin embargo los recursos como impresoras generalmente no desean la transparencia de ubicación. Cuando los usuarios desean imprimir un documento, éstos desean saber exactamente donde será impreso.

Transparencia de acceso: permite acceder a los recursos locales y remotos empleando operaciones idénticas. Uno de los aspectos más difíciles de transparencia de acceso es el tener soporte de acceso seguro, sin interferir con transparencia de acceso. Considere una interfaz gráfica de usuario basada en directorios, en donde los contenidos de los directorios se observan igual no importando si son locales o remotos.

Transparencia de migración: esta implica que los usuarios no pueden ser notificados si un recurso o su trabajo a sido migrado a otra ubicación con el

[7] Ver www.vitanuova.com/inferno/downloads.html

sistema distribuido. La transparencia de nombre es necesaria para que esto ocurra. Considere el caso de los teléfonos móviles, supongamos que tanto el emisor como el receptor viajan en tren por distintas partes del país, moviéndose de un entorno célula a otro. Veamos la terminal del emisor como un cliente y la terminal del receptor como un recurso. Los dos usuarios telefónicos no perciben el desplazamiento de sus terminales (el cliente y el recurso) entre todas las células.

Transparencia de replicación: que permite utilizar múltiples ejemplares de cada recurso para aumentar la fiabilidad y las prestaciones sin que los usuarios y programadores de aplicaciones necesiten saberlo.

Transparencia de concurrencia y paralelismo: múltiples procesos pueden utilizar el mismo recurso o un proceso puede utilizar múltiples recursos al mismo tiempo sin interferencia mutua.

Transparencia frente a fallos: que permite ocultar los fallos, dejando que los usuarios y programas de aplicación completen sus tareas a pesar de fallos de hardware o de los componentes de software. La transparencia a fallos es un beneficio clave para el cómputo distribuido sobre el cómputo centralizado.

1.3 ¿Que es un sistema en tiempo-real?

Un sistema de tiempo-real debe satisfacer las condiciones de tiempo de respuesta o ¡sufrir graves consecuencias! Si las consecuencias consisten en la degradación del desempeño pero no falla, entonces el sistema se conoce como sistema suave de tiempo-real (soft real-time system). Si las consecuencias son el fallo del sistema, el sistema se conoce como sistema riguroso de tiempo-real (hard real-time system). En un sistema riguroso de tiempo-real, una falla en el sistema puede resultar en la muerte del sistema. Si se puede tolerar una probabilidad baja de perdida del tiempo límite, el sistema se conoce como un sistema estable de tiempo-real (firm real-time system). Aplicaciones multimedia a través de la red son ejemplos de sistemas estables de tiempo-real. La mayoría de los sistemas son sistemas suaves de tiempo-real. La herramienta clásica para enseñar cómputo en tiempo-real es el control de trenes por computadora. Si el programador falla, el tren chocara. Tradicionalmente las aplicaciones de tiempo-real son abundantes en el área de las telecomunicaciones, aéreas, espaciales, e industrias de defensa.

Hay dos tipos de sistemas de tiempo-real: reactivos y embebidos. Un sistema de tiempo-real reactivo es aquel que tiene constante interacción con su entorno. Esta reacción tiene que ver con un piloto que oprime constantemente los botones de las funciones de control. Un sistema de tiempo-real embebido es utilizado para controlar hardware especializado que es instalado en un sistema largo. Un ejemplo podría ser el microprocesador que controla la mezcla de combustible para el aire de los automóviles.

1.3.1 Características de los eventos de tiempo-real

Los eventos de tiempo-real caen en una de las siguientes categorías: asíncrono y síncrono. Los eventos asíncronos son totalmente impredecibles como la ubicación exacta en donde caerá un rayo en una tormenta. Generalmente, los eventos asíncronos son causados por fuentes externas. Por ejemplo, las llamadas de teléfono son asíncronas ya que la aplicación de telecomunicaciones no tiene idea de cuando usaran los usuarios su teléfono para conectarse al sistema. Por otra parte los síncronos son predecibles y ocurren con una precisa regularidad. Los eventos síncronos son usualmente generados por fuentes internas como en un componente embebido.

En el sentido de la disciplina de las redes. Asíncrono describe un protocolo de red o una red que transmite cada paquete de datos independientemente en una manera no contigua. En contraste, una red o protocolo de red síncrono manda un mensaje como un flujo contiguo de bits.

1.3.2 Las características de red que afectan aplicaciones distribuidas de tiempo-real

Los siguientes son cuatro hechos en los que las redes pueden afectar las operaciones de aplicaciones de tiempo-real en un entorno distribuido.

1. Latencia de la red
2. Ancho de banda contra costo
3. Optimización de ruteo
4. Características de las micro-redes

El primer hecho es que la latencia de red no pude ser ignorada. La latencia es la experiencia de tener un retrazo debido a los requerimientos de los datos para viajar de la fuente al destino. Las limitaciones físicas básicas pueden causar la latencia en un sistema distribuido, particularmente para aplicaciones distribuidas de tiempo-real como las de telecomunicaciones. Por ejemplo, aún a la velocidad de la luz, la información debería de tomar 20 ms para atravesar los Estados Unidos. Usando una analogía de una carretera, asumimos que no hay semáforos, congestión, o tráfico accidental (errores en los datos causando su retransmisión). Si la aplicación es forzada a realizar una comunicación masiva a componentes distribuidos geográficamente de forma severa, esta tal vez no conozca sus condiciones de tiempo-real. Además, la latencia de red forza a que todos los eventos distribuidos sean asíncronos.

El segundo hecho es el ancho de banda contra costo. Aplicaciones distribuidas de tiempo-real requieren ancho de banda constante. Actualmente, este costo se aproxima al medio millón de dólares por mes para una conexión de 155-Mbps (como una OC3 Sonnet) entre Osaka y Tokio. Entonces, mientras esto puede ser técnicamente posible el mandar video en tiempo-real, este no es costeable. Las

buenas noticias son que el costo del ancho de banda será menor al pasar de los años, pero es difícil determinar cuando será para las masas.

El tercer hecho de los sistemas distribuidos de tiempo-real es que estos residen en una red y entonces son sujetos su topología, su efecto en la distribución y ruteo. En teoría las redes están diseñadas para minimizar el número de saltos (nodos de red) entre la fuente y el destino, pero esta teoría no es cierta en la vida real cotidiana ($ ping www.yahoo.com). Una analogía sería un viaje en automóvil desde los suburbios al área metropolitana, el camino más corto no es siempre el camino más corto en términos de tiempo.

El cuarto hecho es que los sistemas distribuidos de tiempo-real están sujetos a características de micro-redes, como es el tamaño del búfer de los switches en la red, el tamaño de las colas y el tamaño de los paquetes permitidos en la red.

1.4 ¿Qué es un sistema paralelo?

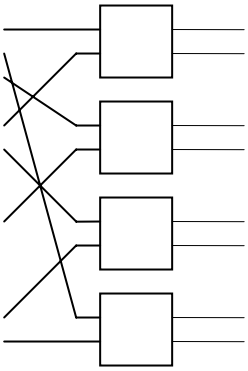
Un sistema paralelo consiste de múltiples procesadores, homogéneos y ubicados en la misma computadora. En adición a los múltiples procesadores, las computadoras paralelas tal vez sean diseñadas para que cada procesador tenga su propia memoria. Estas computadoras paralelas se conocen como **multicomputadoras**. Si el procesador paralelo comparte memoria, estas son conocidas como **multiprocesador**. Estos multiprocesadores también son clasificados en términos de costo para acceder a la memoria compartida. Si el tiempo de acceso es el mismo para todos los procesadores, la memoria del procesador es conocida como Acceso Uniforme a Memoria (UMA), si no tienen el mismo tiempo es conocida como Acceso No-Uniforme a Memoria (NUMA). Multiprocesadores que no comparten (cada procesador tiene su memoria local propia) es conocida como **NORMA** (Acceso No-Remoto a Memoria). Debido a que las computadoras paralelas son utilizadas para resolver los problemas más complejos y son considerablemente más caras que las de un sólo procesador, se conocen como supercomputadoras. Eficientar la utilización de todos los recursos para resolver estos problemas complejos e incrementar la velocidad de cómputo es el trabajo del sistema operativo paralelo.

1.4.1 Arquitecturas paralelas

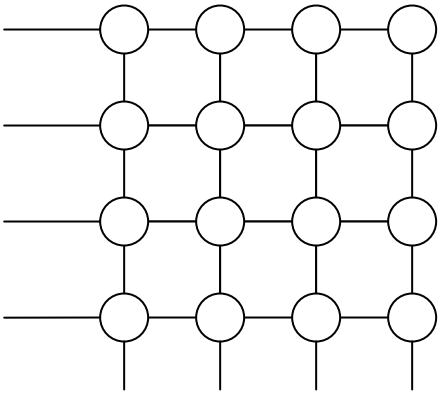
La interconexión de redes que es utilizada para juntar (join) procesadores determina la arquitectura del sistema paralelo. Si el sistema paralelo es multiprocesador, la interconexión de redes tal vez sea diseñada para permitir comunicación síncrona y asíncrona. Los datos y mensajes a través de procesadores y módulos de memoria deberán ser ruteados. Algunas redes de interconexión soportan control de ruteo distribuido, mientras otros emplean control de ruteo centralizado. La topología básica de interconexión de redes tal vez sea estática o dinámica. Las topologías dinámicas permiten la interconexión de redes para ser reconfigurada durante la operación para permitir varias conexiones a

través del uso de switches. Las estáticas son formadas por conexiones directas punto-a-punto que están fijas.

Hay tres clases de topología en la categoría dinámica[8]: de sección simple (single-stage), múltiple sección (multistage), y de barra (crossbar) figura 1.4. Normalmente, los procesadores son puestos a los dos lados de la red. Una red de sección simple esta compuesta de una sección de elementos de switches. Cada elemento de switcheo es un switch de 2 x 2 con dos entradas y dos salidas, y las conexiones de entrada y salida pueden ser rectas o cruzadas (crossover) controladas por una variable boolean[9]. La figura 1.4 (a) muestra un ejemplo con una conexión perfect-shuffle. Una red múltiple sección consiste de más de una sección de elementos de switcheo y normalmente soportan conexiones de una entrada de procesador arbitraria a una salida de procesador arbitraria. En algunas redes de conexiones simultáneas de entrada y salida de procesadores tal vez resulten conflictos en el uso de enlaces de comunicación. Estas redes son llamadas redes de bloqueo (blocking networks), ver la red de línea de base (baseline network) que se muestra en la figura 1.4 (c). Si no, son llamadas rearrangeable nonblocking networks (ver la red *Benes* figura 1.4 (d)). En un switch crossbar cada entrada puede ser conectada a una salida libre sin bloquear. (Ver figura 1.4 (b)).



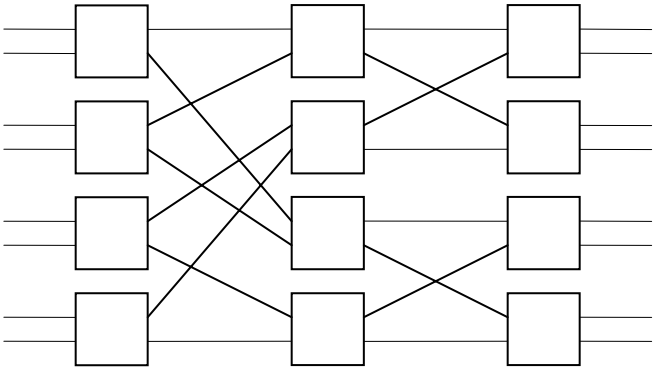
(a) Conexión *perfect-shuffle*



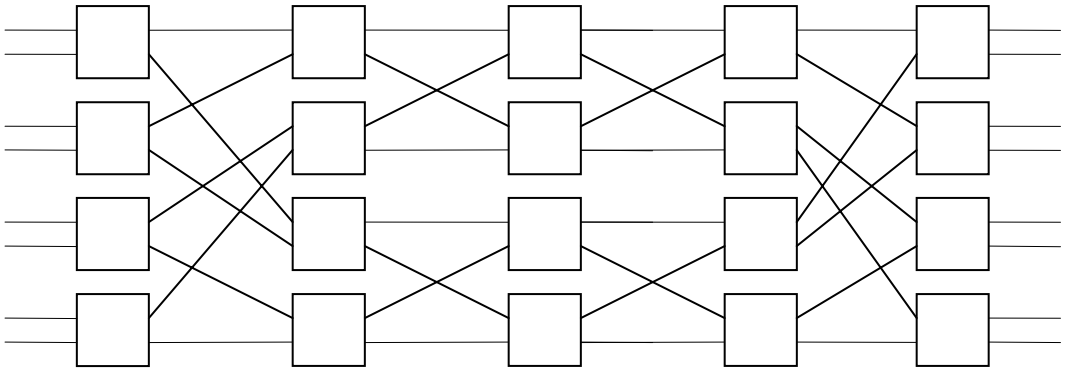
(b) Switch *crossbar*

[8] Wu (1999). *Distributed System Design*

[9] boolean, es un tipo de dato en la mayor parte de los lenguajes de programación, sólo puede tomar dos valores: TRUE o FALSE.



(c) Red *Baseline*



(d) Red *Benes*

Figura 1.4 Topología categoría dinámica

Las topologías en la categoría estática pueden ser clasificadas de acuerdo a las dimensiones requeridas por la composición de la red, por ejemplo de una dimensión, dos dimensiones, y n-dimensión. En general, una conexión estática es representada por una gráfica *G = (V, E)* donde V y E son conjuntos de vértices y nodos respectivamente. Las siguientes son redes estáticas representativas:

1. **Redes completamente conectadas**: cada procesador es directamente conectado a todos los demás procesadores en el sistema.

2. **Arreglo lineal y anillo**: en el arreglo lineal, los procesadores son ordenados en una línea y los nodos adyacentes son conectados. Los procesadores internos tienen dos conexiones y los nodos externos tienen una. Si dos nodos externos son conectados entonces la red forma un anillo.

3. **Malla y redondeada**: una malla de k-elementos y n-dimensiones con $N = k^n$ nodos tiene un grado interior de nodo[10] de 2n y el diámetro[11] de la red es $k(n – 1)$. Cada nodo tiene una dirección ($u_n, u_{n-1}, \ldots , u1$), donde $1 \leq u_i \leq k$. Dos nodos ($u_n, u_{n-1}, \ldots , u1$) y ($v_n, v_{n-1}, \ldots , v1$) son conectados si sus direcciones difieren en uno y sólo un elemento (dimensión), digamos la dimensión *i*; además $| u_i – v_i | = 1$. Básicamente los nodos a lo largo de cada dimensión son conectados como un arreglo lineal. Si los nodos a lo largo de cada dimensión son conectados como anillo, entonces la red toma forma redondeada de n-dimensiones. Mallas de 2-dimensiones y 3-dimensiones son las más populares y varias computadoras paralelas son construidas basadas en éstas.

4. **Árbol y estrella**: una gráfica en la cual hay una única ruta entre cada par de vértices es un árbol. Nodos en un árbol pueden ser ordenadas en niveles. En un árbol binario, los nodos interiores tienen grado 3 con dos hijos y un padre, la raíz tiene grado 2 y las hojas tienen grado 1. La estrella es un árbol de nivel 2 con un alto grado de nodo.

5. **Hipercubo**: un Hipercubo de *n*-dimensión consiste de $N = 2^n$ nodos con un grado del nodo de *n* y un diámetro de *n*. Cada nodo tiene una dirección binaria de *n*-bits. Dos nodos son conectados si y sólo si su dirección difiere en exactamente un bit de posición. El hipercubo binario fue muy popular en la investigación y desarrollo en los 1980s. De hecho, el grado del nodo se incrementa linealmente con respecto a la dimensión, haciendo difícil considerar al hipercubo una arquitectura escalable.

1.5 Paradigmas de software paralelo

Hay tres categorías básicas en las cuales un sistema paralelo divide el trabajo a través de varios procesadores. Esta división tal vez sea completada por el programador, el sistema, o por la arquitectura.

1. **Código replicado**, Partición de datos: en este escenario, a cada procesador se le da la misma tarea pero cada uno trabaja con un conjunto diferente de datos. Por ejemplo, que cada procesador haga una rutina de ordenamiento, pero que un procesador ordene las revistas de la IEEE por título y fecha, y el otro procesador ordene las revistas de la ACM por título y fecha.

2. **Línea de ensamble**: en este escenario, cada servidor tiene una tarea que realiza sobre la tarea completada por el procesador previo, como una línea de ensamble. La tarea no es terminada mientras el procesador final no termine su trabajo. Este se asemeja al modelo que Henry Ford hizo cuando construía automóviles.

[10] Grado de Nodo, es el número de nodos inherentes a un nodo.
[11] Diámetro, es la máxima ruta más corta entre cualquiera dos nodos.

3. **Árbol estructurado**: este escenario es similar al diseño arriba-abajo (top-down) usado para diseñar algoritmos. El primer procesador tiene una tarea que completar y ésta tarea tal vez consista de varias sub-tareas las cuales son pasadas a otro procesador para ser realizadas, quien a su vez pasa parte de la tarea a otro procesador y así sigue.

1.6 Ejemplos de sistemas distribuidos

Un sistema distribuido frecuentemente tiene componentes que no son únicamente las tradicionales máquinas centralizadas si no también componentes de tiempo-real y paralelos. La figura 1.5 muestra un sistema en el dominio de las aplicaciones en telecomunicaciones. Dicha figura no es necesariamente representativa de alguna arquitectura especifica de sistema en producción por alguna compañía, pero si es representativa en lo que uno típicamente encuentra como sistema.

El usuario representa múltiples aplicaciones de usuario deseando acceso al sistema de telecomunicaciones, cuando ellos marcan acceden a la aplicación telefónica a través del switch telefónico que recibe las llamadas basadas en los números que marcaron los usuarios. El usuario espera un servicio rápido y continuo, por eso se hace este sistema distribuido un sistema distribuido de tiempo-real con condiciones de tiempo fuertes, mientras que uno puede pensar que las condiciones de tiempo-real son puestas en el sistema estrictamente para la satisfacción del usuario, uno tiene que tener en mente que las aplicaciones de telefonía conectan clientes al 911 como a las oficinas de sus doctores, centros de control cardíaco, y otras ubicaciones que hacen las condiciones de tiempo-real potencialmente críticas para la vida de los clientes.

El switch telefónico en esta aplicación es capaz de mantener llamadas concurrentes múltiples. Éste es un componente de hardware paralelo manteniendo muchas líneas entrantes y salientes. El switch realiza operaciones concurrentes en las numerosas llamadas concurrentes. La expectativa de los switches es tener menos de un minuto de “apagado” por año y por lo mismo debe ser muy tolerante a fallos. Típicamente un switch es un ejemplo de un componente embebido de tiempo-real.

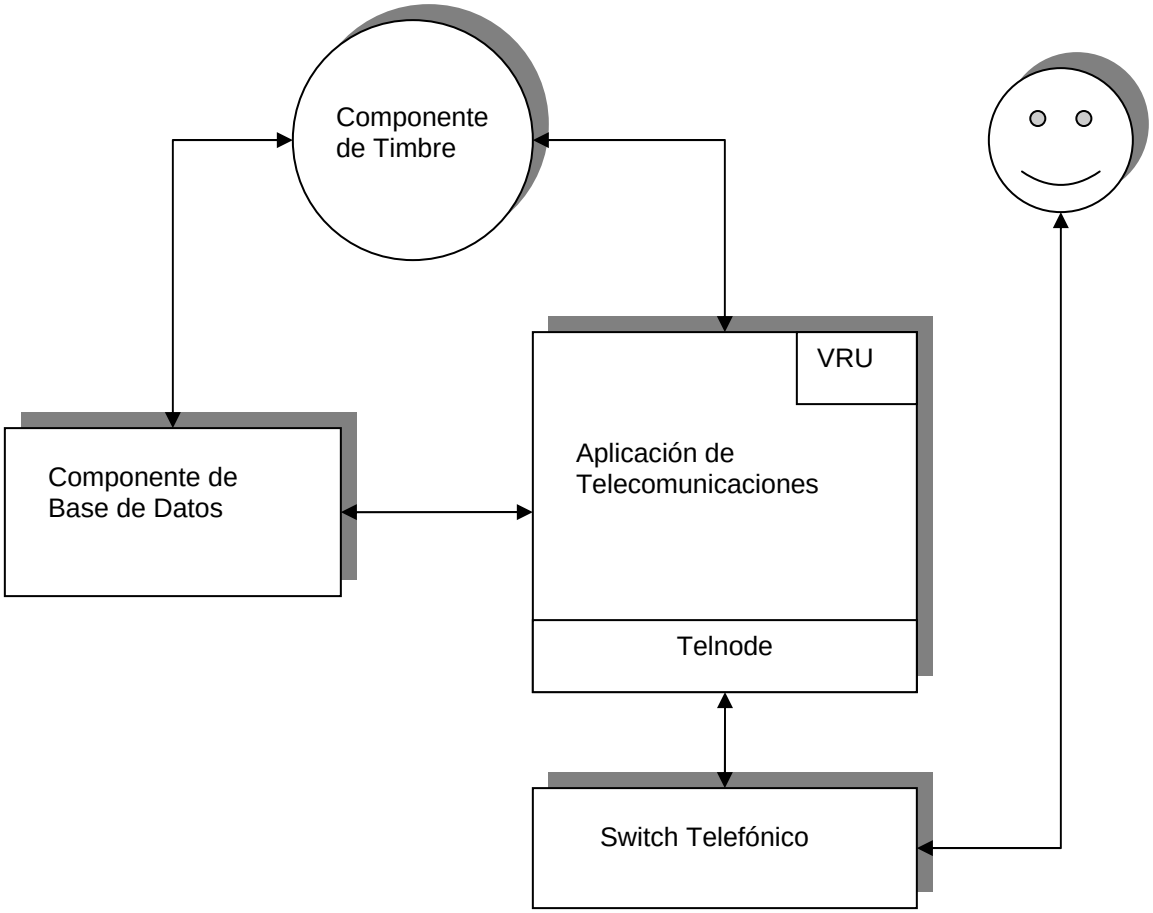


Figura 1.5 Ejemplo de Sistema Distribuido

Un *telnode* es un término genérico para computadoras con tarjetas telefónicas, como son las tarjetas de fax habilitando a la computadora para aceptar faxes o una tarjeta de conferencia habilitando a la computadora para mantener llamadas de conferencia. Un telnode únicamente corre aplicaciones para telecomunicaciones. En un ámbito comercial estos podrían ser un grupo de telnodes que corren el software de aplicación. Estos telnodes tal vez sean organizados de una manera totalmente distribuida como un subsistema distribuido del sistema de telecomunicaciones.

El componente telnode típicamente funciona como o interactúa con una unidad de respuesta a voz (VRU), la cual pone mensajes pregrabados al usuario dependiendo de qué está pasando con la llamada telefónica del usuario. Un ejemplo del mensaje pregrabado sería: “Lo sentimos, el número que usted marco ¡no existe!”. IVR es un ejemplo de un componente reactivo de tiempo-real.

El componente de facturación es una aplicación de tiempo-real que mantiene la pista del costo incurrido por las acciones del usuario. En particular, el componente de facturación en este escenario se ve envuelto en un continuo chequeo de cuanto

dinero está gastando el usuario y lo compara con el que puede gastar según el sistema. Por ejemplo, si el producto es una tarjeta prepagada, la aplicación deberá asegurarse que el usuario no use más de los recursos que es permitido con respecto al monto de su tarjeta prepagada. Esto involucra interacción con un componente de base de datos. Si este componente no es exacto, el usuario perderá su dinero.

El componente de base de datos puede estar involucrado en muchos aspectos de la aplicación, típicamente las compañías que dan el servicio de hospedaje (*hosting)* sirven a un gran número de usuarios, por lo que se requiere que la base de datos sea totalmente distribuida para enfrentar la carga de usuarios y minimizar el tiempo de espera del usuario. Este componente también puede ser implementado en una arquitectura paralela para incrementar la velocidad.

Otro ejemplo es Internet que es una vastísima colección de redes de computadoras heterogéneas interconectadas. Programas ejecutándose en las computadoras conectadas a Internet interactúan mediante paso de mensajes, utilizando un medio común de comunicación. La construcción y el diseño de los mecanismos de comunicación en Internet (los protocolos Internet) es una realización técnica fundamental que permite que un programa que se está ejecutando en cualquier parte dirija mensajes a programas en cualquier otra parte.

Internet también es un vasto sistema distribuido. Les permite a los usuarios, en cualquier lugar que estén, hacer uso de servicios como el World Wide Web, el correo electrónico, y la transferencia de archivos. El conjunto de servicios es abierto, puede ser extendido por la adición de servidores y nuevos tipos de servicios.

1.7 Resumen

Los sistemas distribuidos incrementan su dominio en la industria de la computación y la vida diaria. Los sistemas distribuidos también incluyen componentes de tiempo real y paralelos. Cada componente presenta el sistema operativo con su propio conjunto de retos y problemas. Tanto los sistemas distribuidos como los paralelos emplean múltiples procesadores. Un sistema paralelo involucra componentes homogéneos fuertemente acoplados comparados con el vasto conjunto de componentes diferentes presentes en un sistema distribuido. Un sistema de red no pretende ocultar el hecho de que múltiples recursos remotos se están utilizando. Entonces mientras un sistema distribuido puede emplear un procesador remoto sin que el usuario lo sepa, un usuario debería tener todos los comandos de acciones remotas cuando utilizan un sistema operativo de red. Los sistemas de tiempo real tienen condiciones inherentes severamente estrictas con respecto al tiempo y estas condiciones de tiempo por si mismas son retos y más cuando tienen que lidiar con los retardos de las redes en sistemas distribuidos de tiempo real.

2

Sistema de Archivos Distribuido

2.1 Clasificación de los sistemas de archivos

Un sistema de archivos es uno de los más importantes servicios en un sistema distribuido, soporta procesamiento en muchas computadoras, facilitando el compartir archivos y bases de datos. También provee las bases para construir otro tipo de servicios necesarios para los usuarios como un servidor de impresoras o un servidor de nombres.

La influencia del uso de servidores en la forma general de un sistema distribuido de computadoras es substancial y va más allá de beneficios financieros. Tan pronto como las facilidades que son necesarias para la mayoría de las computadoras son provistas en este camino las computadoras individuales vienen a ser menos autónomas y la red mucho más importante. Además, hay ventajas en tener una cierta uniformidad de servicios (por ejemplo, un camino uniforme de archivar o respaldar archivos).

En suma para dar soporte de compartir (sharing) datos y bases de datos, un servidor de archivos provee los siguientes servicios:

1. **Respaldo (Backup) y recuperación automática**: el backup es realizado como una medida preventiva contra fallas del medio y errores de usuario. Debido a que los procedimientos correspondientes de recuperación requieren algo de entrenamiento y una diaria atención, el usuario debería ser liberado de estas obligaciones.

2. **Movilidad de usuario**: un sistema de archivos hace esto posible al usar computadoras diferentes a tiempos diferentes. Esto es necesario cuando:

    a. Una computadora falla
    b. Algunos usuarios tal vez necesiten computadoras en más de una ubicación
    c. Las computadoras son administradas como un pool común.

    En resumen, un sistema de archivos hace posible el proveer un entorno de trabajo independiente de la computadora utilizada, o de su ubicación, sin la necesidad de dispositivos de almacenamiento secundario.

3. **Estaciones de trabajo sin disco**:
    a. Son deseadas por que los dispositivos de almacenamiento mecánico son ruidosos y generan calor, los cuales deberían ser evitados en un entorno de trabajo.

Hay cuatro clasificaciones principales de sistemas de archivos, Gosinski (1991). De acuerdo con la primera clasificación, hay dos clases principales de servidores de archivos:

1. **Servidor de archivos real**: es un sistema de archivos completo que reconoce nombres de archivos textuales y administra directorios de archivos para sus usuarios.

2. **Servidor de almacenamiento**: almacena datos en objetos no estructurados que son accesibles únicamente por identificadores primitivos.

La segunda clasificación divide a los servidores de archivos en tres categorías (sistema de archivos simple, sistema de archivos universal, y servidor que provee soporte de administración para bases de datos) en términos de tres parámetros de diseño: acceso a unidad de dato, unidad de bloqueo (locking), y ámbito de actualizaciones atómicas. Estos parámetros tienen el impacto más significativo en la complejidad del diseño del servidor de archivos. Las tres categorías son presentadas en la siguiente tabla.

	Sistema de Archivos Simple	Sistema de Archivos Universal	Soporte de Administración para Bases de Datos
Unidad de Acceso a Dato	Archivo	Subconjunto Secuencial de Bytes	Registro
Unidad de Bloqueo	Archivo	Archivo	Registro ó Variable
Ámbito de Actualizaciones Atómicas	Archivo	Archivos Múltiples	Archivos Múltiples

Tabla 2.1 Clasificación de Sistemas de Archivos.

Una tercera clasificación de servicios de archivos distinguen dos tipos de servicios de archivos: tradicional y robusto.

1. **Servicio tradicional de archivos**: ofrecido por casi todos los sistemas operativos centralizados, esto es que los archivos podían ser abiertos, leídos, y vueltos a escribir (rewritten). Las actualizaciones son usualmente implementadas por el servidor de archivos simplemente sobrescribiendo el bloque de disco relevante. El control de concurrencia, si la hay, usualmente involucra bloqueo total de archivos antes de actualizarlos.

2. **Servicio de archivos robusto**: es dirigido a aquellas aplicaciones que requieren una fiabilidad extremadamente alta. Ofrece generalmente actualizaciones atómicas. Un servicio de archivos robusto normalmente incluye un servicio de archivos tradicional como subconjunto.

La cuarta clasificación de servidores de archivos la cual está asociada de forma muy cercana con la primera, toma el acceso remoto de archivos en consideración. Hay dos formas de acceso remoto de archivos.

1. **Transferencia de archivos explícita**: provee el servicio más básico; un cliente debe invocar una utilidad de transferencia de archivo para transferir un archivo remoto. Algunos de estos sistemas realizan conversión limitada de datos para mitigar los efectos de hardware heterogéneo y formato de archivos. Algunos de ellos también utilizan mecanismos de control de acceso y autenticación para proveer protección. Protocolos de transferencia de archivos basados en la conexión como FTP son usados. Estos servidores sufren de lo siguiente: un cliente no puede ejecutar programas almacenados remotamente, la granularidad de acceso es un archivo, y los clientes deben recordar la ubicación del archivo deseado. En resumen, transferencia de archivos explicita es inconveniente y también el tiempo que se consume al acceder al archivo.

2. **Sistema de archivos distribuido**: en él cual la ubicación del archivo es transparente a los usuarios, los cuales hacen referencia al archivo por su nombre. El sistema ubica el servidor que almacena el archivo referenciado, realiza operaciones orientadas a la seguridad (por ejemplo, autenticación), sincroniza el acceso al archivo y finalmente transfiere el archivo. Por razones de desempeño (performance) algunos datos y directorios son almacenados en caché, y por fiabilidad (reliability) algunos archivos son replicados.

2.2 Requisitos del sistema de archivos distribuido

La mayoría de los requisitos y obstáculos potenciales en el diseño de servicios distribuidos fueron observados en los primeros desarrollos de sistemas de archivos distribuidos. Al comienzo ofrecían transparencia de acceso y transparencia de ubicación, requisitos de prestaciones, escalabilidad y control de concurrencia; la tolerancia a fallos y seguridad surgieron después en etapas posteriores del desarrollo.

- **Transparencia**. el servicio de archivos es el servicio más cargado en una intranet, por lo que su funcionalidad y prestaciones son críticas. El diseño de un servicio de archivos debe soportar muchos de los requisitos de transparencia:
    - Transparencia de acceso: los programas del cliente no deben preocuparse de la distribución de los archivos. Se proporcionan un conjunto sencillo de operaciones para el acceso a archivos locales y remotos. Los programas escritos para trabajar sobre archivos locales serán capaces de acceder a los archivos remotos sin modificación.
    - Transparencia de ubicación: los programas del cliente deben ver un espacio de nombres de archivos uniforme. Los archivos o grupos de

archivos pueden ser reubicados sin cambiar sus nombres de ruta, y los programas de usuario verán el mismo espacio de nombres en cualquier parte que sean ejecutados.
    - Transparencia de movilidad: ni los programas del cliente ni las tablas de administración de sistema en los nodos cliente necesitan ser cambiados cuando se mueven los archivos. Esta movilidad de archivos permite que archivos o, más comúnmente, conjuntos o volúmenes de archivos puedan ser movidos, ya sea por los administradores del sistema o automáticamente.
    - Transparencia de prestaciones: los programas cliente deben continuar funcionando satisfactoriamente mientras la carga en el servicio varíe dentro de un rango especificado.
    - Transparencia de escala: el servicio puede ser aumentado por un crecimiento incremental para tratar con un amplio rango de cargas y tamaños de redes.

- **Actualizaciones Concurrentes de Archivos**. los cambios en un archivo por un cliente no deben interferir con la operación de otros clientes que acceden o cambian simultáneamente el mismo archivo. Esto es el tema bien conocido del control de concurrencia, Coulouris, Dollimore y Kindberg (2005). La necesidad de control de concurrencia para el acceso a datos compartidos en muchas aplicaciones está ampliamente aceptada y las técnicas para su implementación son conocidas, aunque muy costosas. La mayoría de los servicios de archivos actuales siguen los estándares de UNIX moderno proporcionando bloqueo consultivo u obligatorio a nivel de archivo o registro.

- **Replicación de archivos**. en un servicio de archivos que soporta replicación, un archivo puede estar representado por varias copias de su contenido en diferentes ubicaciones. Esto tiene dos beneficios, permite que múltiples servidores compartan la carga de proporcionar un servicio a los clientes que acceden al mismo conjunto de archivos, mejorando la escalabilidad del servicio y mejorando la tolerancia a fallos, permitiendo a los clientes localizar otro servidor que mantiene una copia del archivo cuando uno ha fallado. Muy pocos servicios de archivos soportan totalmente la replicación, pero la mayoría soportan la caché local de archivos o porciones de archivos, una forma limitada de replicación.

- **Heterogeneidad del hardware y del sistema operativo**. las interfaces del servicio deben estar definidas de modo que el software del cliente y el servidor puedan estar implementados por diferentes sistemas operativos y computadoras. Este es un requisito importante de la extensibilidad (JVM[1]).

- **Tolerancia a fallos**. el papel central de un servicio de archivos en los sistemas distribuidos hace que sea esencial que el servicio continúe

[1] *Java Virtual Machine. Sun Microsystems.*

funcionando aún en el caso de fallos del cliente y del servidor. Los fallos de comunicación transitorios son manejados basándose en la semántica de invocación de cómo *máximo una vez,* Coulouris, Dollimore y Kindberg (2005). Otra opción es *al menos una vez* la cual es una semántica más sencilla con un protocolo de servidor diseñado en términos de operaciones *idempotens*, asegurando que solicitudes duplicadas no producen actualizaciones inválidas en los archivos. Los servidores pueden ser *sin estado*, por lo que pueden ser rearrancados y el servicio reestablecido después de un fallo sin necesidad de recuperar el estado previo. La tolerancia a la desconexión o fallos del servidor precisa replicación de los archivos, que es más difícil de alcanzar.

- **Consistencia**. los sistemas de archivos convencionales, como los que se proporcionan en UNIX, ofrecen una *semántica de actualización de una copia*. Esto se refiere a un modelo para acceso concurrente a archivos en el que el contenido del archivo visto por todos los procesos que acceden o actualizan a un archivo dado es aquel que ellos verían si existiera únicamente una copia del contenido del archivo. Cuando los archivos están replicados, o en el caché, en diferentes lugares, hay un retardo inevitable en la propagación de las modificaciones hechas en un lugar hacia los otros lugares que mantienen copias, y esto puede producir alguna desviación de la semántica de una copia.

- **Seguridad**. virtualmente todos los sistemas de archivos proporcionan mecanismos de control de acceso basados en el uso de listas de control de acceso. En sistemas de archivos distribuidos, hay una necesidad de autenticar las solicitudes del cliente por lo que el control de acceso en el servidor está basado en identificar al usuario correcto y proteger el contenido de los mensajes de solicitud y respuesta con firmas digitales y opcionalmente con cifrado de datos secretos.

- **Eficiencia**. un servicio de archivos distribuido debe ofrecer posibilidades con la misma potencia y generalidad que las que se encuentran en los sistemas de archivos convencionales y deben proporcionar un nivel de prestaciones comparable.

Las técnicas utilizadas para la implementación de los servicios de archivo son una parte importante del diseño de sistemas distribuidos. Un sistema de archivos distribuidos debe proporcionar un servicio que sea comparable con, o mejor que, los sistemas de archivos locales en prestaciones y fiabilidad.

2.3 Arquitectura de un sistema de archivos centralizado

En un sistema operativo centralizado, los siguientes servicios básicos son provistos: servicio de bloque (disco), servicio de archivo y servicio de directorio. Estas funciones no están usualmente separadas aunque están puestas en una forma jerárquica.

El servicio de disco, de archivo, y de directorio son realizados por los siguientes módulos:

1. **Módulo de dispositivo**: el cual realiza entrada/salida de disco y manejo de buffer.
2. **Módulo de bloque**: accesa y asigna bloques de disco.
3. **Módulo de archivo**: relaciona identificadores de archivo (ID's) a un archivo particular.
4. **Módulo de acceso a archivo**: el cual lee y escribe los datos del archivo o atributos.
5. **Módulo de control de acceso**: checa permisos para realizar las operaciones pedidas.
6. **Módulo de directorio**: provee el mapeo requerido entre nombres de archivo en texto y las referencias a los archivos, (su ID).

El problema es si este modelo jerárquico puede ser usado en un sistema distribuido y si los servicios básicos y sus módulos de funcionalidad cumplen los requerimientos de los usuarios del sistema distribuido, y del sistema mismo.

2.4 Arquitectura de un sistema de archivos distribuido

Un sistema de archivos distribuido podría proveer, en general, los mismos servicios que un sistema de archivos centralizado, estos son: servicio de bloque, servicio de archivo y servicio de disco. De cualquier forma, por razones de desempeño, fiabilidad, y distribución, las siguientes características adicionales y servicios deben ser considerados: replicar y actualizar múltiples copias, transacción y un control de acción atómica, nombramiento de objetos distribuidos, y el colocar y ubicar archivos en el sistema de archivos distribuido.

En un sistema de archivos centralizado normalmente los servicios básicos no son separados, mientras que en un entorno distribuido hay una división de responsabilidades y las funciones de los sistemas de archivos son realizadas por servicios separados.

Los servicios de un sistema de archivos distribuido son realizados por los siguientes módulos:

1. **Módulo de dispositivo**: el cual realiza entrada/salida de disco y manejo de buffer.

2. **Módulo de bloque**: accesa y asigna bloques de disco.
3. **Módulo de caché**: pone en caché resultados obtenidos recientemente de operaciones de disco y operaciones remotas.
4. **Módulo de archivo**: relaciona identificadores de archivo (ID's) a un archivo particular.
5. **Módulo de acceso a archivo**: el cual lee y escribe los datos del archivo o atributos.
6. **Módulo de acceso a atributos**: lee y escribe atributos de archivo.
7. **Módulo de operaciones de transacción**: provee un mecanismo de transacción para permitir agrupar operaciones elementales para que se ejecuten automáticamente.
8. **Módulo de recuperación**: provee mecanismos de recuperación después de una falla.
9. **Módulo de control de concurrencia**: provee mecanismos de control para asegurar que la consistencia del objeto se preserva y que cada acción atómica se realiza en un tiempo finito.
10. **Módulo de consistencia de datos y actualización de copias múltiples**: provee el desempeño de un sistema de archivos cuando los archivos son compartidos, manteniendo la consistencia de los datos y proporcionando mecanismos de actualización de copias múltiples.
11. **Módulo de directorio**: el cual resuelve nombres de usuario proporcionando el mapeo requerido entre nombres textuales de archivo y las referencias del archivo, estos son, nombres de sistema, ID's de archivo.
12. **Módulo de control de acceso**: checa permisos para realizar las operaciones pedidas.
13. **Módulo de colocación**: selecciona el servidor que contendrá un nuevo archivo creado.
14. **Módulo de localización (ubicación)**: encuentra la ubicación del servidor de un archivo referenciado.
15. **Módulo de reubicación**: provee el desempeño del sistema de archivos por medio de la reubicación de archivos.

2.4.1 Servicio de disco

Un servicio de bloque o disco provee una vista lógica del sistema de almacenamiento del disco. Este servicio también habilita un sistema de almacenamiento sencillo para compartir entre diferentes sistemas de archivo. El servicio de disco tiene que ver con la lectura y escritura de bloques de discos físicamente sin considerar como están organizados. Las facilidades provistas deben permitir borrar y truncar archivos. Un servicio de bloques también es responsable de mantener los bloques usados frecuentemente en un buffer de memoria. Normalmente los comandos son para asignar y escribir un bloque de disco, o para regresar una dirección para que el bloque pueda ser leído mas tarde.

En la mayoría de los sistemas de archivos distribuidos el servicio de disco es separado del servicio de archivo. Por lo general, esta separación hace sencillo el combinar diferentes métodos de almacenamiento, y diferentes medios de almacenamiento. Un servicio de disco puede combinar alta velocidad con alto rendimiento usando caché y almacenamiento dual.

Almacenamiento de archivo

Diferentes modelos conceptuales de un archivo son usados por diferentes sistemas de archivos. De acuerdo con el modelo simple, un archivo es una secuencia no estructurada de datos; es decir que no hay una subestructura conocida por el servidor de archivos. Un modelo mas estructurado asume que un archivo es una secuencia ordenada de registros, los cuales pueden ser de diferentes tamaños. Estos registros pueden tener llaves y pueden ser separadamente direccionados. Un modelo mas avanzado de un archivo es un árbol. Cada nodo del árbol puede tener una llave y un registro de datos, tanto ambos como ninguno de ellos.

Manejaremos el modelo de la secuencia de registros debido a que en el lenguaje de programación Java (Deitel y Deitel [2005]) así lo maneja. Cuando un archivo es creado, un espacio debe ser asignado (allocated) para el archivo. Este espacio es recuperado cuando el archivo es borrado o truncado. El tamaño del archivo puede cambiar cuando las operaciones en éste sean realizadas. Por lo que una asignación dinámica de almacenamiento es requerida para permitir a los archivos crecer y decrecer.

Los archivos tienen atributos que los describen. Cada atributo tiene su nombre, un tipo y un valor. Algunos atributos son creados cuando el archivo es creado y no cambian. Otros pueden ser cambiados por los usuarios. Un tercer grupo de atributos son mantenidos y cambiados por el servidor de archivos. Ejemplos de atributos de archivos son: nombre de archivo, operaciones disponibles, control de acceso a los datos, fecha y hora de creación, fecha y hora de la última modificación, dueño, y llave cifrada.

Los archivos están almacenados en bloques de almacenamiento direccionables, su tamaño es fijado por el dispositivo de almacenamiento. Esto explica la existencia del servicio de disco del módulo de bloque, el cual es responsable de la asignación, lectura y escritura de bloques. Las operaciones de bloque en un sistema distribuido pueden ser implementadas en un módulo de software como parte de un software servidor de archivos, o como un servicio separado.

El uso de un servicio de bloques ofrece algunas ventajas en la implementación de un servicio de archivos compartido:

1. Servicios de archivos especificados de forma diferente pueden coexistir y compartir el mismo disco de almacenamiento en un sistema distribuido simple.

2. Una variedad de discos y otros medios pueden ser usados.

3. La implementación de los servicios de archivo pueden estar separados de las especificaciones de optimización del disco.

En un sistema basado en bloques, un archivo es almacenado en un conjunto no contiguo de bloques de almacenamiento. Esto implica la necesidad de registrar una secuencia de apuntadores a los bloques de los cuales el archivo está compuesto. Esta secuencia de apuntadores es almacenada en una estructura de datos (Langsam, Augenstein y Tenenbaum [1997]) separada llamada el índice del archivo. El índice del archivo debería estar organizado en un camino tal que tanto el acceso secuencial como el aleatorio a los componentes del archivo sean soportados ver figura 2.1.

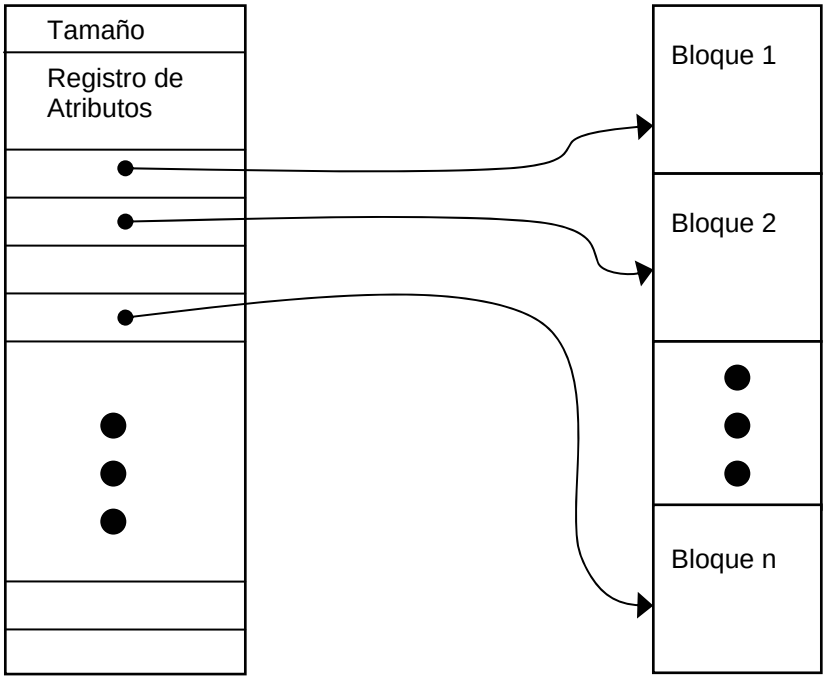


Figura 2.1 Estructura del Índice de Archivo.

Como se muestra, los atributos del archivo aunque son guardados con el archivo, no están incluidos en el contenido del archivo (figura 2.1). Esto es por que son sujetos a diferentes controles de acceso. El registro de atributos es administrado y usado por el servicio de directorio. Hay otro registro almacenado en el índice de archivo el cual es llamado tamaño de archivo.

El servicio de bloques es normalmente implementado con comandos que permiten al usuario asignar, desasignar, leer y escribir bloques de datos. Cada uno de estos bloques esta protegido; entonces si un bloque es asignado a un usuario, otro usuario no puede tener acceso sin permiso. Para implementar acciones atómicas en archivos, la operación de escritura en el bloque debería de ser una operación atómica, con una confirmación después almacenar el bloque en el disco. Para permitir que el servicio de archivos realice políticas de control de concurrencia, una simple función de candado en cada bloque podría ser implementada en el servicio de bloque.

Sistema de caché

El rendimiento de cualquier sistema que soporta memoria externa sufre de operaciones de acceso a disco y operaciones remotas, las cuales consumen mucho tiempo, por lo que es necesario evitarlas lo más posible. Esto motivó el desarrollo de un sistema de software, llamado un sistema de caché, este reduce el costo de acceso al archivo, almacenando bloques recientemente utilizados en la memoria local, reutilizándolos cuando aún son válidos. Hay que tomar en cuenta que no sólo los datos sino también los atributos de los archivos son puestos en el caché siempre y cuando sean válidos.

Un caché es un conjunto de copias locales de bloques o datos remotos. Una posible posición de un caché es en el servicio de disco y la relación entre ellos y el manejador (drive) del disco.

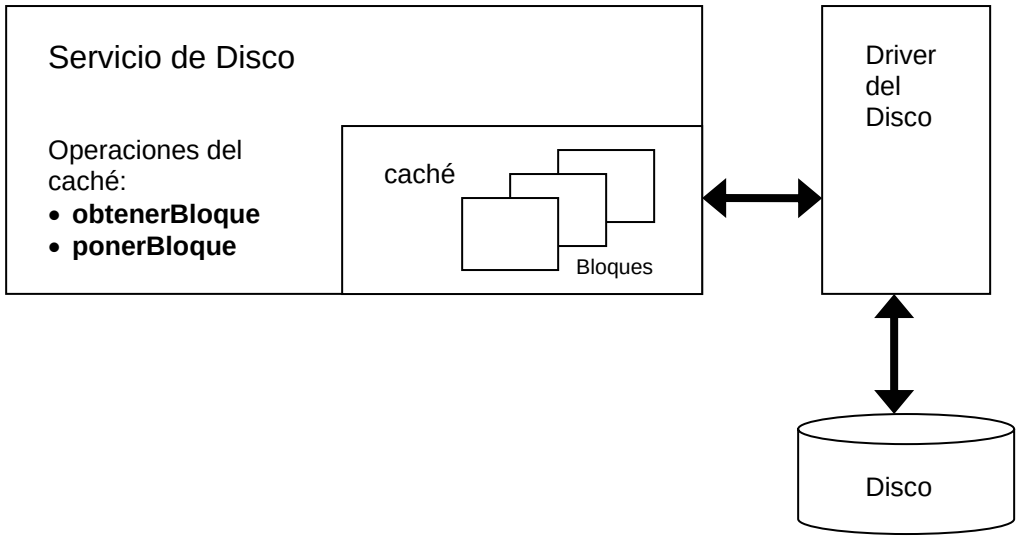


Figura 2.2 Relación entre Servicio de disco, caché y Driver del Disco.

Una memoria caché externa es un área de la memoria principal la cual está organizada como un arreglo de bloques. El tamaño de cada bloque es el mismo que el tamaño del bloque en el disco. Cada bloque de caché tiene asociado un

apuntador. Estos apuntadores son checados en cada operación de lectura. Si el apuntador está presente entonces el bloque de caché es leído. De otro modo el caché es cargado con el contenido del bloque desde el disco y el apuntador es actualizado. Entonces operaciones de lectura subsecuentes que requieran el mismo bloque usan el caché.
Varios problemas surgen al querer introducir un sistema de caché. El más serio es mantener consistencia entre los bloques de caché en más de una ubicación. Se dice que el Caché es consistente si mantiene una copia exacta de datos. El problema de mantener copias de caché locales actualizadas cuando cambios son realizados en ubicaciones remotas es llamado el problema de consistencia del caché.

Hay tres soluciones básicas al problema de consistencia del caché:

1. **Pasiva**: esta solución asume que el sistema no hace nada, únicamente se les informa a los usuarios que tengan cuidado, ¡ja!

2. **Servidor que mantiene objetos en caché**: el servidor mantiene la pista de que cliente tiene que bloque y archivo en su caché. Cuando un cliente modifica un bloque o archivo, este le informa al servidor de archivos. El servidor de archivos le informa a otros clientes que ese bloque en particular o archivo es inválido y debe ser removido de su caché.

3. **Servidor que monitorea acceso a objetos**: los clientes pueden únicamente almacenar en su caché. Un cliente le informa al servidor de archivos cuando abrir un archivo específico, indicando si está abierto para lectura o escritura. El servidor no se inmuta si el archivo es abierto por todos los clientes como sólo lectura. Si únicamente un cliente abre un archivo para escritura y no hay lectores, el servidor seguirá en su estado vegetativo. Pero cuando un segundo usuario abre el archivo pero antes de esto bloquea al segundo usuario hasta que actualiza el archivo en el servidor. Debido a la gran carga que se genera al compartir archivos esta solución es razonablemente eficiente.

Identificadores de archivo único

Cada archivo administrado por un sistema de archivos tiene por lo menos un nombre de usuario y un nombre único en el sistema. Los servidores de archivos utilizan únicamente nombres de sistema. Un nombre de sistema puede tomar varias formas diferentes, por ejemplo, un identificador de objetos (archivos) único (UFIDs).

UFIDs son usualmente enteros largos de un tamaño fijo. Esto permite al servidor de archivos mapear a los datos en los archivos. Estos identificadores son únicos a lo largo de todos los archivos en el sistema distribuido y deberían ser generados

de tal forma que sea difícil falsificar[2]. Es importante notar que los UFIDs no deben de actuar como una dirección de archivo. Un ejemplo para construir un UFIDs es concatenando los siguientes campos:

1. La dirección del servidor el cual creó el archivo
2. Una *timestamp*[3] del reloj del servidor, o
3. un entero el cual represente el número consecutivo de archivo creado por el servidor, y
4. un número aleatorio.

Estos UFIDs tienen las siguientes ventajas en un sistema de archivos distribuidos:

1. Tienen una completa independencia de ubicación
2. Cada UFID identifica de forma no ambigua a un sólo objeto del sistema de archivos
3. Los UFIDs tal vez pasen entre procesos y de computadora a computadora sin tener que ser cambiados cada vez.
4. La dirección del servidor es necesaria únicamente para garantizar unicidad y no para localizar el objeto en la red.

Entonces el UFID constituye una llave. Este hecho es usado cuando se desarrolla un método para acceder al archivo. Una seguridad más avanzada puede darse añadiendo un campo de acceso al UFID.

Identificador del Servidor	***Número de Archivo***	***Número Pseudo aleatorio***

Figura 2.3 Identificador de Archivo Único

Ubicación de archivo

Los clientes de un servicio de disco no necesitan saber la ubicación de los archivos. Realizar una operación a un archivo requiere que se especifique su UFID

[2] Para generar números primos largos y pseudo aleatorios, primero generamos números pseudo aleatorios largos y después vemos si es primo usando un algoritmo de tiempo polinomial como lo es el algoritmo SOLOVAY-STRASSEN, Stinson (2002). *Cryptography – Theory and Practice*

[3] Es la fecha que el servidor tiene al momento de crear el UFID.

y una página, o un marcador (bloque) de ubicación dentro del archivo (offset[4]). Esta información es enviada en un mensaje de petición al sistema de archivos, el cual realiza las operaciones adecuadas para obtener la ubicación del índice del archivo y de los bloques de archivo.

Entonces, esto es responsabilidad del servicio de disco, el localizar un archivo específico y almacenar los archivos en diferentes computadoras de un sistema distribuido.

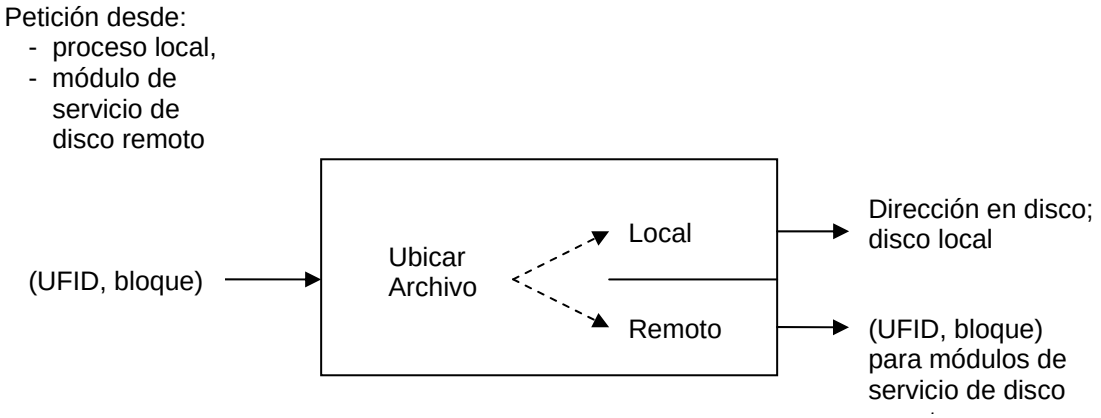


Figura 2.4 Diagrama de la función Ubicar Archivo

La figura anterior muestra que si el archivo especificado es local, el módulo de servicio de disco realiza la operación de entrada/salida; de otro modo una petición es enviada al servicio de módulo de servicio de disco[5].

Este método de referencia de bloques de archivo debería ser rápido porque se realiza sólo una vez por cada petición.

Funciones de un servidor de disco

Un servicio de disco administra y accesa a los contenidos de una colección larga de bloques de disco, esto les permite a los clientes lo siguiente:

1. Obtener nuevos bloques.
2. Liberar bloques que ya no son requeridos.
3. Transferir datos hacia adentro y fuera de los bloques.

[4] En lenguajes de programación como C++ y Java se utiliza esta técnica mediante la función fseek y seek respectivamente.
[5] Este método es utilizado por DOMAIN, Leach (1985). *The File System of an Integrated Local Network*

Estos servicios pueden ser provistos por operaciones de servicio de disco. Servicios de disco de diferentes sistemas de archivos distribuidos otorgan diferentes conjuntos de operaciones. Como un ejemplo, veamos la siguiente lista:

asignarBloque();
Asigna un nuevo bloque y entrega su apuntador.

liberarBloque(*Bloque*);
Libera el *Bloque*.

obtenerBloque(*Bloque*);
El *Bloque* se lee del disco mediante la dirección dada, se guarda en un buffer disponible del caché, y una copia es enviada al proceso del usuario. No hay acceso al *Bloque* si este ya está en el caché.

ponerBloque(*Bloque, Datos*);
Datos almacenados en el caché son copiados en el disco en la dirección de *Bloque* dada.

removerBloque(*Bloque*);
Remueve el *Bloque* de la caché.

Cuando se diseñan las funciones de servicio provistas por el servicio de disco, varios aspectos deberían ser considerados. En primera, el tamaño elegido para el apuntador del bloque[6]. Este valor es importante debido a que determina el máximo número de bloques que pueden ser almacenados por el servidor de bloques. Segundo, el tamaño de los bloques, el uso de bloques grandes reduce el efecto de latencia y la complejidad del índice de archivos para los archivos grandes; por lo tanto el tiempo de acceso se reduce. Tercero, debido a que la eficiencia del sistema de archivos depende del tamaño de bloque y del tamaño de los archivos, la mejor solución puede hacerse dando un número diferente de tamaños de bloque. Cuarto, un servicio de disco debería ser diseñado para asegurar que el archivo será accesible después de todas las fallas de sistema excepto las más catastróficas (un ataque militar por parte del gobierno X… “en nombre de Dios”). Esto implica por lo menos que el mapa de ubicación del archivo y los índices de archivo deben ser recuperables si el sistema falla mientras se actualiza, o mientras ocurre un error de disco. Esto nos lleva al quinto aspecto de diseño: *almacenamiento estable*, es un diseño para asegurarse que cualquier dato esencial permanente será recuperable después de cualquier falla de sistema sencilla.

2.4.2 Servicio de archivo

Un servicio de archivo provee a sus clientes con una abstracción que consiste de archivos, cada uno de cuales es una secuencia lineal de datos (registros). Los datos tal vez sean definidos por el sistema o por el usuario. Las operaciones son de lectura y escritura de registros, comenzando desde un lugar particular en el archivo. Al cliente no le concierne el cómo y donde son almacenados los datos en el archivo o las relaciones entre los archivos.

[6] Bovet y Cesati (2000). *Understanding the Linux Kernel*

Un servicio de archivos distribuido habilita a los usuarios para acceder a los archivos sin necesidad de copiarlos a un disco local, por ejemplo, una Workstation. Si se usan Workstation sin disco, el servicio de archivo provee almacenamiento de datos permanente.

El objetivo principal de un servicio de archivos distribuido es el de ofrecer facilidades de colocación de al menos las mismas que tienen los sistemas de archivos centralizados.

Archivos mutables e inmutables[7]

Dos tipos de archivos pueden ser almacenados por el servidor de archivos: archivos mutables e inmutables. Un archivo es mutable si sólo es una secuencia almacenada que es alterada por cada operación de actualización. Los archivos provistos en sistemas operativos centralizados son archivos mutables.

Un archivo inmutable es aquel que no puede ser modificado un vez que este ha sido creado exceptuando la eliminación. La segunda característica importante de un archivo inmutable es que su nombre no debe ser rehusado. Esto implica que el nombre de un archivo inmutable tiene el contenido fijo en el archivo, el archivo no es un contenedor para información variable. En Gifford (1988) se enfatiza que compartir únicamente archivos inmutables hace sencillo el soporte de compartir los archivos consistentemente. Más aún esto facilita la implementación de un sistema de archivos distribuido. Compartir archivos inmutables significa que el caché local puede ignorar la posibilidad de que archivos remotos cambien.

Los archivos inmutables no son actualizados. Una nueva versión es creada cada vez que un cambio es hecho al contenido del archivo, y la versión anterior se mantiene sin cambios. Algunas veces únicamente un registro de las diferencias entre versiones es almacenado, para minimizar el almacenamiento requerido. Versiones anteriores son removidas cuando hay poco espacio en disco.

Los archivos inmutables son fuente de dos problemas potenciales: incremento en el uso del espacio en disco y el incremento en la asignación de disco[8].

Operaciones del servicio de archivos en los datos

Un archivo contiene datos, y también atributos. Esto implica que hay dos conjuntos separados de operaciones, uno que lidia con datos y otro que lidia con los atributos. Un conjunto de operaciones que trabajan con los datos son puestas a disposición del usuario para manipular los archivos.

[7] Gosinski (1991). *Distributed Operating Systems – The Logical Design*
[8] Los interesados en archivos inmutables pueden referirse al Cedar File System (CFS) en Gifford (1988).

La siguiente es una lista de operaciones comúnmente utilizadas[9] en archivos y en sus datos:

crear(); → *archivo*
Crea un nuevo archivo de tamaño 0 con el nombre especificado por *archivo*, establece sus atributos y entrega un UFID para este.

abrir(*archivo*);
Abre el *archivo* existente que puede ser usado para realizar **lectura, escritura**, y otras operaciones en este archivo.

copiar(*archivoOrigen, archivoDestino*);
Copia el archivo *archivoOrigen* hacia el *archivoDestino*.

obtenerTamaño(*archivo*);
Retorna el tamaño del *archivo,* después pone un candado en las propiedades del *archivo*.

leer(*archivo, offset, cont, buffer*);
Lee el total de bytes de datos del *archivo* descritos por *cont* comenzando desde *offset* y los pone contiguamente en el *buffer*.

escribir(*archivo, offset, cont, buffer*);
Escribe un total de bytes de datos igual a *cont* desde el *buffer* hacia el *archivo* comenzando desde *offset*.

cerrar(*archivo*);
Termina el acceso para realizar acciones al *archivo*.

eliminar(*archivo*);
Remueve el *archivo* desde el almacén de archivos. Un error ocurre si el archivo está actualmente abierto.

Operaciones del servicio de archivos sobre atributos

El servicio de archivos mantiene un conjunto de atributos para cada archivo, como son: la fecha de creación, el tipo de archivo, fecha de último acceso, fecha de última modificación, permisos de archivo y lista de acceso. Estos atributos son interpretados por el servicio de archivos como una secuencia no interpretada de bytes, los cuales tal vez se almacenen y recuperen.

Las siguientes son operaciones realizadas sobre los atributos del archivo:

leerAtributo(*archivo*);
Regresa los valores de los atributos del *archivo*.

escribirAtributo(*archivo, atributo*);
Escribe (modifica) el valor existente del *atributo* (modo, UFID, tamaño, tiempo de acceso, tiempo de modificación, etc.)

[9] Gifford (1988). *The Ceder File System*, y Sandberg (1986). *The Sun Network File System: Design, Implementation and Experience.*

El servicio de archivos, servicio de directorio, y el usuario son responsables de asignar los contenidos de los atributos del archivo. En particular, el servicio de directorio determina la estructura interna y los valores almacenados en la sección de atributos. El tipo de atributos es idéntico para todos los archivos administrados por un servicio de directorio particular.

Propiedades operacionales del sistema de archivos

Dos propiedades de operaciones en los datos de archivos actualmente abiertos son importantes en la construcción de un sistema distribuido: idempotencia, y sin estado (stateless).

1. **Idempotencia**: una operación se dice que es idempotente si su efecto cuando se ejecuta más de una vez es el mismo como si fuera una ejecución sencilla. Ciertos errores causados por fallas de las computadoras y retardos en la comunicación tal vez dejen ejecuciones repetidas o algunas operaciones, por ejemplo, llamadas a procedimientos remotos (RPC). Si la repetición tuviera un efecto inesperado, el servidor debe prevenirlo implementando supresión duplicada. Si la operación es idempotente, entonces las sobrecargas debido a la duplicación tal vez sean anuladas. Como sea, operaciones repetidas pueden degradar la eficiencia del sistema.

2. **Servidor de archivo sin estado**: la discusión sobre idempotencia nos deja con el problema de cual sitio, esto es, el servidor o el cliente, debería mantener el estado de los archivos cuando operaciones son realizadas en ellos, y la información en todas las peticiones pasadas.

   Un servidor el cual no mantiene su estado interno es conocido como un servidor sin estado. En este caso los parámetros necesarios para llevar acabo una operación en un archivo son almacenados por el cliente, y es pasada al servidor en un mensaje de petición. El servidor no mantiene la pista de cualquier petición pasada. Después de abrir un archivo el índice a esa entrada es regresada al cliente para su uso en subsecuentes peticiones.

   Un ejemplo de información de estado es el apuntador de lectura/escritura. En un sistema operativo centralizado, el kernel puede tomar este apuntador por cada archivo en una tabla de procesos. Si un servidor almacena esta información por cada cliente, las dos desafortunadas situaciones tal vez surjan. Si el servidor falla, la información debería perderse; cuando el servidor arranque de nuevo el programa cliente debería continuar su tarea sin percatarse de esta falla y producir resultados inconsistentes. De la otra forma, un programa cliente debería también fallar; el servidor deja de mantener información que es errónea pero no puede ser fácilmente retirada. Entonces cuando el programa se reinicia, este produce resultados

inconsistentes. Claramente operaciones que no dependen del estado almacenado en el servidor de archivos pueden simplificar su diseño.

En resumen el servidor sin estado satisface la recuperación a fallos; cuando un servidor falla, el cliente retransmite únicamente sus mensajes de petición hasta que un mensaje de respuesta es recibido. Cuando un cliente falla, la recuperación no es necesaria ni para el cliente como tampoco para el servidor.

Ahora veamos, si un estado es mantenido por el servidor, llamado un servidor *con estado*, entonces la recuperación es más difícil. Esto es debido a que tanto el cliente y el servidor necesitan detectar de buena forma las fallas: el cliente debe detectar fallos del servidor para reconstruir el estado del servidor; el servidor debe detectar los fallos del cliente para descartar cualquier estado.

Un servidor sin estado tiene también algunas desventajas. Si el cliente mantiene el estado entonces problemas surgen si el servidor mueve el archivo. Más aún, si la información dentro de un archivo puede ser actualizada concurrentemente por varios clientes, los servidores simples sin estado son inadecuados. Así que un servidor de archivos debería soportar transacciones atómicas en los archivos. Esto requiere la retención de cierta información del estado, como son los candados. Esta información es usada únicamente por el servidor de archivos y será removida cuando la transacción se complete o después de que el servidor o el cliente fallen.

En general es difícil implementar semánticas normales en sistema de archivos sin estado. Esto puede observarse en los casos donde por ejemplo un usuario remueve un archivo abierto, o cambia su modo de protección. En ambos casos el cliente no puede hacer nada con el archivo.

2.4.3 Servicio de transacción

Si el servicio de archivos básico es usado concurrentemente por un número de procesos que leen y escriben datos en el mismo archivo, efectos inesperados en el archivo pueden resultar. Si un cliente o servidor de archivos falla, errores indetectables pueden producirse. Para evitar estos problemas, el servicio de archivos debería soportar transacciones y un servicio de transacciones atómicas. Esto implica que el servicio de archivos debería extenderse para proveer un mecanismo que permita operaciones en archivos para ser agrupados (por ejemplo, crear, abrir, escribir, cerrar) dentro de una transacción, usando *inicioTransacción* y *finTransacción*, para ejecutarlas automáticamente. Veamos un ejemplo:

inicioTransacción

crear(*archivo*);
abrir(*archivo*);
leer(*archivo, offset, cont, buffer*);

escribir(*archivo, offset, cont, buffer*);
cerrar(archivo);

finTransacción(*transID*)

Cuando un proceso cliente necesita ser provisto con un servicio, este envía una petición *inicioTransacción* al servidor de archivos. Usualmente, la primitiva *inicioTransacción* permite tanto checar si el cliente tiene permisos para iniciar una transacción y el comienzo de la transacción. Si una transacción puede ser abierta, el servidor regresa un mensaje de resultado con un identificador de transacción único, *transID*; todos los futuros mensajes con la transacción deben traer consigo ese identificador.

Después de abrir una transacción, operaciones orientadas al archivo (por ejemplo, crear, abrir, leer, escribir, cerrar, eliminar) pueden ser realizadas. Después de completar estas operaciones, el cliente enviará la petición *finTransacción* para cerrar la transacción. El sistema completa la transacción y envía una respuesta al cliente.

Dos requerimientos deben satisfacerse para que un sistema soporte transacciones atómicas. La primera es que cada transacción debe ser recuperable: cuando un cliente o servidor se detiene por la razón que sea, entonces los cambios a archivos no deben de ser completadas o los archivos deben de regresar al estado que tenían antes de iniciar la transacción. Entonces todas las operaciones *crear, abrir*, *escribir*, y *eliminar* con una sola transacción deberían de realizarse por completo o ninguna de ellas. Una transacción atómica es llevada acabo por la cooperación entre procesos cliente y el servicio de transacción. Entonces, el proceso cliente especifica la secuencia de operaciones que deben ser realizadas en una sola transacción y el servicio de transacción debe preservar la atomicidad de la secuencia entera.

Las transacciones deben ser exitosas o si no estas deben ser abortadas tanto por el cliente o el servidor. Si la transacción tuvo éxito normalmente el servidor le informa al cliente que la transacción se realizo mediante un *commit*[10]. Si no se tiene éxito la transacción es abortada. Tres posibles escenarios de una transacción se muestran a continuación.

[10] Reese (2000). *Database Programming with JDBC™ and Java™*

inicioTransacción	*inicioTransacción*
crear(archivo *);* *abrir(* archivo *);* *leer(* archivo, offset, cont, buffer *);* *escribir(* archivo, offset, cont, buffer *);* *cerrar(archivo);* *finTransacción(* transID *)*	*crear(* archivo *);* *abrir(* archivo *);* *leer(* archivo, offset, cont, buffer *);* *escribir(* archivo, offset, cont, buffer *);* *abortarTransacción(* transID *)*

inicioTransacción

crear(*archivo*);
abrir(*archivo*);
leer(*archivo, offset, cont, buffer*);
ABORTA EL SERVIDOR ⟶ escribir(*archivo, offset, cont, buffer*);

El hecho de que una transacción puede ser abortada implica que todas las operaciones de actualización de archivos deberán ser realizadas en una manera tentativa. Esto quiere decir que esas operaciones de actualización son realizadas de una manera que ellas deben ser registradas permanentemente o deshechas.

Operaciones del servicio de transacción

Definimos un servicio de transacción como aquella clase de servicio de archivo la cual soporta transacciones en sus archivos. Así que listemos las operaciones de este servicio, las cuales son propuestas en Braban y Schlenk (1989).

iniciarTransacción
Crea una nueva transacción y entrega un *transID* único. Este identificador es usado en las otras operaciones en la transacción.

finTransacción(*transID*)
Realiza el *commit* a la transacción *transID*; el resultado retornado indica si la transacción fue confirmada y exitosa (*commited*), fallo, o si la transacción esta inactiva debido a que esta fue abortada.

abortarTransacción(*transID*)
Aborta la transacción; todos los archivos abiertos para esta transacción son desbloqueados y el estado de la transacción es cambiado a inactivo.

Recuperación después de fallos

La atomicidad de las transacciones es amenazada por dos factores:

1. **Fallas de procesos y computadoras**: tres tipos de fallas son identificadas:

a. fallas en la transacción las cuales resultan de valores de datos totalmente erróneos o errores de procesamiento;
b. fallas del sistema, las cuales son causa de un *bug*[11] en el código del sistema administrador de bases de datos, una falla del sistema operativo, o una falla del hardware;
c. fallas en los medios de almacenamiento, en las cuales una porción de almacenamiento secundario esta dañado.

2. **Concurrencia.**

Hay dos formas de recuperación comúnmente usadas: lista de intenciones y versión de archivo, antes de eso consideremos unos aspectos necesarios de las operaciones de transacción. Una transacción consiste de dos fases con tres estados asociados: *tentativo*, *commited*, y *abortado*.

Operaciones del Cliente	***Fase***	***Acciones del Servidor***

inicioTransacción
 	
crear(*archivo*);
abrir(*archivo*);
leer(*archivo, offset, cont, buffer*);
escribir(*archivo, offset, cont, buffer*);
cerrar(archivo);
 	
finTransacción(*transID*)
	↑
	primera fase
	↓
	COMMIT
	↑
	segunda fase
	↓
		Hace copias tentativas de datos modificados del archivo
		
		Incorpora copias tentativas dentro de los archivos

Tabla 2.2 Dos fases de una Transacción

La primera fase. Esta fase comienza cuando el servidor recibe una petición *inicioTransacción* desde el cliente y finaliza cuando *finTransacción* es recibida. Durante esta fase la transacción esta en el estado tentativo. Entonces, el servidor hace copias tentativas de datos cambiados en los archivos. Únicamente durante la primera fase una transacción puede ser abortada. Los archivos del cliente no son afectados permanentemente por operaciones de la transacción abortada, la única cosa que es hecha por el servidor es realizar copias tentativas de datos cambiados en los archivos.

La segunda fase. Esta fase comienza después de que el servidor recibe una petición *finTransacción*. Si las condiciones descritas anteriormente se llevaron a cabo, se entra en el estado commited, y los valores tentativos de los datos modificados se hacen permanentes insertándolos en lugares apropiados de los archivos. Desde luego que esta fase debe ser atómica. De otro modo el estado abortado entrara en el juego y el servidor informara al cliente que la transacción ha fallado.

[11] Es un error inesperado.

La lista de Intenciones

En el método de la lista de intención (ver Mukherjee [1988]) el servidor hace una lista de todas las acciones (llamada una lista de intenciones) para cada transacción. Esta lista será hecha por el servidor únicamente después de que una transacción es confirmada (commited).

La lista de intenciones puede ser considerada como un log[12] de operaciones de una transacción. La lista contiene un registro por cada operación (por ejemplo, *escribir*) que hará cambios a los archivos. Entonces, cuando por ejemplo una operación *escribir* es realizada, su intención es almacenada en la lista de intenciones en vez de realizar actualizaciones al archivo en cuestión. La construcción de una lista de intenciones se muestra enseguida.

Operaciones del Cliente	***Fase/Estado***	***Registro de Intenciones***

inicioTransacción
 	
crear(*archivo*);
abrir(*archivo*);
leer(*archivo, offset, cont, buffer*);
escribir(*archivo, offset, cont, buffer*);
cerrar(archivo);
 	
finTransacción(*transID*)
	↑
	primera fase/tentativa
	↓
	COMMIT
	↑
	segunda fase/commited
	↓
		escribir(*archivo, offset, cont, buffer*);
		
		intenciones ⟶ archivos

Tabla 2.3 Construcción de una lista de intenciones.

La tabla muestra que la bandera del estado de la transacción esta asociada con la lista de intenciones. Entonces, durante la primera fase se asigna la *bandera commit* a *tentativa*. Este estado es cambiado a *commited* o *abortado* cuando la petición *finTransacción* es recibida. Esta operación de cambio de estado debe realizarse de forma atómica.

En general para tener transacciones en archivos en un sistema basado en paginación de memoria[13], es necesario el mantener diferentes versiones de páginas alrededor. En un sistema basado en el método de *log*, estas páginas son almacenadas en registros de log y son incorporadas dentro de los archivos.

La lista de intenciones y la bandera commit son ubicadas en almacenamiento estable separado del archivo involucrado en la transacción. Listas de intención pueden ser escritas o borradas automáticamente. Este registro tiene que ser hecho de una forma que después que un servidor ha fallado o reiniciado se mantenga intacto. Entonces si una falla ocurre, todas las listas de intención son

[12] Archivo con mensajes de error, confirmación, operaciones, etc. del proceso.
[13] Silberschatz, Peterson y Galvin (1994). *Sistemas operativos - conceptos fundamentales,*

descartadas durante la recuperación de la falla. Enseguida cada lista es borrada. Esto garantiza que la transacción se realice correctamente.

La versión de archivo

En la versión de archivo el servidor hace una nueva versión de los archivos que contienen los cambios. Si la transacción fue exitosa, la nueva versión del archivo es usada; si no es descartada.

Un archivo debe tener una secuencia de versiones las cuales son resultado de modificaciones de archivo. Cada una de estas versiones es escrita sólo una vez y es tentativa. Esta secuencia forma un historial cronológico del archivo. Una transacción puede modificar más de un archivo. En esta situación, hay una versión tentativa por cada archivo. Cuando el servidor confirma la transacción, la versión mas reciente confirmada se añade a la secuencia de versiones viejas.

Cuando dos o más transacciones concurrentes tienen acceso al mismo archivo, dos tipos de conflictos surgen:

1. **Conflicto de versión**: el cual ocurre cuando transacciones concurrentes acceden al mismo archivo, pero ninguno de los datos modificados en cualquiera de las transacciones tienen que ser accedidos o modificados por la otra transacción. Este tipo de conflicto puede ser resuelto por una acción de unión.

2. **Conflicto de serialización**: el cual ocurre cuando dos o más transacciones concurrentes son autorizadas para acceder a los mismos datos dentro de un archivo, y uno o más de estos accesos son una operación *escribir*. Bloquear puede usarse para prevenir este tipo de conflictos.

Cuando se implementa versión de archivo en un sistema de computadora basado en paginación de memoria, la técnica de páginas sombra[14] (shadow pages) puede ser usada para hacer versiones nuevas de los archivos.

Para ejemplificar el método de páginas sombra, recordemos que una página es una secuencia de bytes de tamaño fijo. Una página (un bloque) es identificado con un archivo usando un número de página. Una página es normalmente almacenada en un sector de disco. Para asignar mapeos del par (UFID, número de página) en un sector de disco, un mapa de archivos es utilizado. El mapa de archivos es usado por el sistema de archivos para encontrar el sector correcto para realizar las operaciones en la página relevante. La asignación es realizada basándose en el mapa de asignación que contiene el estatus de *libre* o *asignado* de cada sector.

[14] Gosinski (1991)

Control de concurrencia

Si un sistema no falla, todos los mecanismos de control de concurrencia deben asegurarse que la consistencia de los objetos se preserva, y cada acción atómica es completada en un tiempo finito.

Esto puede llevarse acabo si las transacciones se ejecutan de una forma que su efecto sobre datos compartidos es equivalencia en serie. La equivalencia en serie de transacciones puede ser llevada a acabo por un servidor de archivos serializando el acceso a datos. La parte que se comparte para la cual el acceso debe ser serializado debería ser la parte más pequeña de un archivo. Esto es muy importante por que si partes largas de un archivo o un archivo entero es accesado en serie por procesos, la concurrencia es reducida.

Los problemas de control de concurrencia pueden ser resueltos por los tres métodos comúnmente utilizados que se muestran a continuación:

1. **Bloqueo (locking)**: usando este método, un servidor asigna un bloqueo a cada dato (ítem[15]) de un archivo antes de que este sea accesado en nombre de la primera transacción la cual quiere acceder. Cada bloqueo es etiquetado con el identificador de la transacción. Entonces únicamente la transacción que bloqueo los ítems puede acceder. Otra transacción debería esperar hasta que los ítems de datos sean desbloqueados. Los ítems de datos son desbloqueados, cuando su bloqueo es removido debido a que la transacción fue completada (commited o abortada).
2. **Control de concurrencia optimista**: de acuerdo con este método el cual esta basado en la esperanza de que no haya conflictos de acceso, el servidor le permite a una transacción proceder con el fin de la primer fase. De cualquier forma, antes de que se haga un commit el servidor se fija para ver si la transacción ha usado los mismos ítems de datos que transacciones anteriores. Si un conflicto ocurre, la transacción debe ser abortada y reiniciada.
3. **Timestamps**: un servidor basado en este método registra el tiempo mas reciente de lectura y escritura de cada ítem de dato. Cada transacción compara su propia timestamp con el del ítem. Esta comparación es usada para determinar si la operación actual puede ser realizada o no. Las transacciones son abortadas y reiniciadas cuando se tardan en hacer una operación en un ítem particular dentro del archivo.

Es importante saber que cada transacción tiene su propio registro de cambios (valores/copias tentativas). Más aún, cada transacción no puede observar valores tentativos de otras transacciones.

[15] Tipos de datos mínimos a los que se les asigna un bloqueo.

Implementación de un servicio de transacción

Antes que nada, dimos una introducción a algunas operaciones para trabajar en una lista de intenciones. Enseguida, presente la implementación de bloqueos para control de concurrencia en combinación con las listas de intención. En estas implementaciones la técnica de *shadow pages* fue utilizada. Es importante saber que las páginas sombra son accesibles para la transacción que la creó.

Implementación de listas de intención

Las operaciones que pueden cambiar el contenido de un archivo son *crear*, *escribir* y *eliminar*. Entonces una entrada debe ser hecha en la lista de intenciones por cada una de estas operaciones. La entrada consiste de un registro de la información necesaria para llevar acabo las operaciones *crear*, *escribir* y *eliminar*.

Cada transacción tiene una lista de intención, la cual es administrada por el servicio de transacciones en la forma de una lista de *registros de intención* en almacenamiento estable. Los registros de intención grabados son los siguientes:

1. El tipo de operación, *crear*, *escribir* y *eliminar*.
2. El identificador de transacción
3. El archivo y bloque (página) a la que la intención hace referencia
4. Apuntador al bloque que mantiene la página sombra
5. Tamaño sombra de un archivo.

Algunas operaciones de almacenamiento y registros de intención de acceso deben ser usados únicamente con el servidor, como se muestra en la siguiente lista:

obtenerIntención(*operación, transID, archivo, página*)
: Si la lista de intención para la transacción identificada por *transID* contiene una intención para realizar la *operación* en la *página* del *archivo*, regresa un apuntador al registro de intención.

asignarIntención(*operación, transID, archivo, página*)
: Registra una entrada en la lista de intención que corresponde a *transID*.

Bloqueos (Locks)

Los bloqueos son usados para proveer control de concurrencia en operaciones del servicio de transacción. Asumimos por simplicidad que los bloqueos se aplican a nivel de página. Entonces, el servidor mantiene un conjunto de bloqueos por cada página de un archivo. Hay dos tipos de bloqueos: lectura y escritura. El conjunto debe contener varios locks de lectura, un lock de escritura, o no tener locks asociados con cada página. Enseguida hay una lista de operaciones requeridas en los bloqueos:

bloqueo(*transID, archivo, página, tipoBloqueo*)
Espera en la variable de condición de bloqueo, bloqueoLibre, si hay un bloqueo asignado en la *página* del *archivo* que tiene conflictos con *tipoBloqueo*. Se regresa el valor boolean *true* si la página esta bloqueada.

rebloqueo(transID*, archivo, página, tipoBloqueo*)
Asigna un nuevo bloqueo del tipo *tipoBloqueo* en *página* del *archivo*.

desbloqueo(*transID, archivo, página, tipoBloqueo*)
Remueve el bloqueo y señales si ahí hay un bloqueo del tipo *tipoBloqueo* en la *página* del *archivo* en nombre de la transacción *transID*. Esto permite a cualquier transacción esperando en un *bloqueo* para *página* proceder.

Estas operaciones de bloqueo son usadas en la implementación de operaciones de servicio de transacción, como son *escribir* y *leer*. Cuando más de una transacción trata de modificar un bloqueo concurrentemente, pueden surgir conflictos entre ellas. Para evitar estos conflictos las operaciones deberían ser protegidas por un monitor. Como resultado únicamente una transacción a la vez puede ejecutar *escribir*, *leer* o *desbloquear*, y el rango de concurrencia con el servidor es restringido severamente.

2.4.4 Servicio de nombres/directorio

Una facilidad de nombres administra nombres de usuario y su mapeo dentro del sistema de nombres que es significativo a nivel máquina. Esto le permite a los archivos ser referenciados por nombres de cadenas de texto. Esto es muy importante para separar lógicamente las políticas de almacenamiento y las reglas de mapeo de la facilidad fundamental que administra los archivos (y transacciones) que los nombres representan.

Un sistema de nombres debería de ser único a través de un sistema distribuido. Los nombres de usuario deberían también ser uniformes tal que el nombre del archivo no cambie de una computadora a otra. Cada usuario debería poder nombrar archivos personales sin ningún conocimiento de los nombres usados por los otros usuarios. Esto nos deja con diferentes nombres de usuario para los mismos archivos o los mismos nombres de usuario para diferentes archivos. Un servicio de nombres debería ser desarrollado de cierta forma que habilite estos requerimientos contradictorios. Para llevarlo a acabo utilizamos un árbol jerárquico de espacio de nombres.

Los archivos usualmente le pertenecen a sus creadores. El sistema de archivos debería proveer un servicio de protección para detener accesos no autorizados a archivos. De cualquier forma muchos archivos se comparten. Esto implica que un sistema de protección debería permitir compartir y esto debería ser garantizado en diferentes niveles de granularidad (por ejemplo, propietario, grupo, usuario).

En un sistema distribuido podría haber más de un sistema de archivos. De cualquier forma un cliente no debería tener que preocuparse por este hecho, ni por que estos servidores estén distribuidos geográficamente, ni que los archivos estén ubicados en algún servidor en particular. Esto resulta en sistemas de archivo colaborando para proveer un servicio de archivos integrado transparente en su ubicación, distribución, y replicación de sus archivos.

Esquemas de nombramiento

En un sistema distribuido, hay tres métodos principales a esquemas de nombramiento (Levy y Silberschatz [1989]):

1. **Nombramiento simple**: los archivos son nombrados por alguna combinación de su nombre local y el nombre del servidor. Como ya se menciono esto garantiza unicidad de un amplio espacio de nombres.

2. **Montar directorios remotos en directorios locales**: este nos da la apariencia de un directorio de árbol coherente. Este es usado en NFS de Sun.

3. **Espacio global de nombres**: provee la total integración entre los componentes del sistema de archivos, una sola estructura global de nombres crea todos los archivos en el sistema.

Servicio de directorio

El servicio de directorio implica nombramiento y protección de archivos. Este servicio típicamente provee objetos llamados *directorios*, que mapean nombres de archivos en ASCII a identificadores de archivos (UFIDs) usados por el servicio de archivos. Un directorio no es más que un archivo y un conjunto de funciones que lo distinguen de un archivo común, permitiéndole a este archivo verse como un registro de archivo de tamaño fijo, cada registro corresponde a una entrada en el directorio. Debido a que el servicio de directorio es un cliente del servicio de archivos, sus mapeos son almacenados por el servicio de archivos. Entonces cada directorio tiene un UFID. El servicio de directorio protege archivos, utilizando una lista de acceso para cada archivo para asegurarse de que los UFIDs no son dados a clientes erróneos. Debido a que el servicio de directorio esta separado del servicio de archivos una variedad de servicios de directorio pueden desarrollarse y usarse en un sólo servicio de archivos.

La división de responsabilidades entre servicio de archivo y servicio de directorio está basada sobre el uso de UFID para acceder a los contenidos del archivo. Entonces, cuándo un archivo es creado por la petición del cliente, el servicio de archivos le asigna un nuevo UFID. Este cliente puede hacer peticiones al servicio de directorio para guardar el UFID y su nombre textual. Como resultado, el cliente

puede hacer peticiones de acceso al archivo dando su nombre de usuario al servicio de directorio.

Un árbol jerárquico de espacio de nombres es comúnmente utilizado en muchos sistemas centralizados (ejemplo, UNIX). Este árbol garantiza unicidad de nombres, y resolución de nombres eficiente. Un árbol jerárquico de espacio de nombres fue desarrollado también para LOCUS.

Veamos la lista de operaciones del servicio de directorio:

crearDir(*dir, dirPadre*)
Crea un nuevo directorio *dir* en le directorio *dirPadre*.

abrirDir(*dir*)
Abre el directorio *dir*.

leerDir(*dir, nombre, modoAcceso*)
Localiza el nombre textual en el directorio y regresa el UFID correspondiente; reporta un error si este no puede ser encontrado o si el cliente que hace la petición no esta autorizado para acceder al archivo en la manera especificada por *modoAcceso*.

insertar(*dir, nombre, archivo)*
Añade el par (*nombre, archivo*) al directorio *dir*.

desnombrar(*dir, nombre*)
Remueve la entrada que contiene *nombre* del directorio. Si *nombre* no esta en el directorio se lanza un error.

cambio(*dirViejo, nombreViejo, dirNuevo, nombreNuevo*)
Cambia la entrada que contiene *nombreViejo* a *nombreNuevo*.

obtenerNombres(*dir, patrón*) ⟶ secuenciaNombres
Retorna el conjunto de todos los nombres dentro del directorio *dir* que coinciden (match) con la expresión regular dada por *patrón*.

cerrarDir(*dir*)
Cierra el directorio *dir*.

eliminarDir(*dir, dirPadre*)
Remueve el directorio vacío *dir* del directorio *dirPadre*.

cambAtribDir(*dir, atributos*)
Cambia los atributos del directorio *dir*; permite la modificación de la lista de control de acceso asociada con este directorio.

Control de acceso

El control de acceso requiere del servicio de directorio para acceder y actualizar los atributos de los archivos. Por otro lado, el servicio de directorio debería proveer una operación la cual habilite al cliente para inspeccionar los atributos del archivo. Debido a que cada archivo tiene su propio dueño y el *usuarioID* del propietario es registrado en los atributos, el servicio de directorio debería también proveer

operaciones que permitan a los propietarios de los archivos otorgar y revocar permisos de acceso a otros usuarios.

El problema es quien puede acceder a los directorios y puede realizar operaciones como *leerDir*, *insertar*, *cambio* y *desnombrar*. Debido a que el servicio de directorio es el propietario de todos los archivos los cuales contienen sus directorios, es permitido realizar todas las operaciones del servicio de archivos en ellos. Entonces, el servicio de directorio puede crear, escribir, leer y eliminar el mapeo entre los nombres de archivo y sus UFIDs.

Es obvio que cuándo los usuarios tienen directorios privados separados, el UFID del directorio debería de tomarse en consideración como una capacidad. Entonces no se necesita control de acceso. Únicamente cuando los directorios se comparten se necesita algo de control de acceso. Un esquema de permisos de acceso simple usado para archivos puede bastar en esta situación.

Expongamos el control de acceso y autenticación que lleva acabo el servidor NFS. A diferencia del sistema de archivos convencional UNIX, el servidor NFS es sin estado y no mantiene archivos abiertos en nombre de sus clientes. Por lo tanto el servidor debe comprobar la identidad del usuario frente a los atributos de acceso del archivo en cada solicitud, para poder ver si el usuario tiene permiso de acceso al archivo de la forma solicitada.[16] El protocolo Sun RPC requiere que los clientes envíen la información de autenticación del usuario (por ejemplo, los convencionales 16 bits del ID de usuario y del grupo de UNIX) con cada solicitud y está se comprueba frente a los permisos de acceso en los atributos del archivo.

En la forma más simple hay un boquete de seguridad en este mecanismo de control de acceso. Un servidor NFS proporciona una interfaz RPC convencional en un puerto bien conocido en cada máquina y cualquier proceso puede comportarse como un cliente, enviando solicitudes al servidor para poder acceder o actualizar un archivo. El cliente puede modificar las llamadas RPC para incluir la ID de cualquier usuario, haciéndose pasar por el usuario sin su consentimiento o permiso. Este boquete de seguridad ha sido cerrado con el uso de una opción en el protocolo RPC para el cifrado de la información de autenticación del usuario. Recientemente se ha integrado Kerberos en Sun NFS para proporcionar una solución más fuerte y completa a los problemas de autenticación y seguridad del usuario.

Colocación y ubicación de archivos en un servicio de archivos distribuido

En un sistema distribuido podría haber más de un servidor de archivos. Un cliente no debería de preocuparse por el número de servidores, ni por que estén distribuidos geográficamente, y ni por que estén en un servidor en particular los

[16] Coulouris, Dollimore y Kindberg (2005). *Distributed Systems Concepts and Design*

archivos. Todos estos servidores deberían colaborar para proveer un servicio de archivos integro y transparente a su ubicación, distribución y replicación de archivos.

Cuando un servidor de archivos mantiene particiones del conjunto total de archivos, usualmente cada partición es asociada con una aplicación en particular y los servidores son colocados en ubicaciones separadas. Un servidor de archivos dado administra su partición independientemente de otros servidores. El cliente no esta alerta de la ubicación del servidor que tiene sus archivos. Hay tres problemas básicos. El *problema de colocación* surge cuando un archivo es creado y un servidor debería escogerlo para mantenerlo. El *problema de ubicación* surge cuando un cliente desea referirse a un archivo dado (el servidor que mantenga ese archivo debería ser encontrado). El *problema de reubicación* surge cuando los archivos necesitan ser reubicados por razones de eficiencia.

Para resolver estos problemas es necesario proveer servicios adicionales con un responsable servicio de nombres para la colocación de archivos. Un esquema sistemático que provee estos servicios se muestra en la siguiente figura.

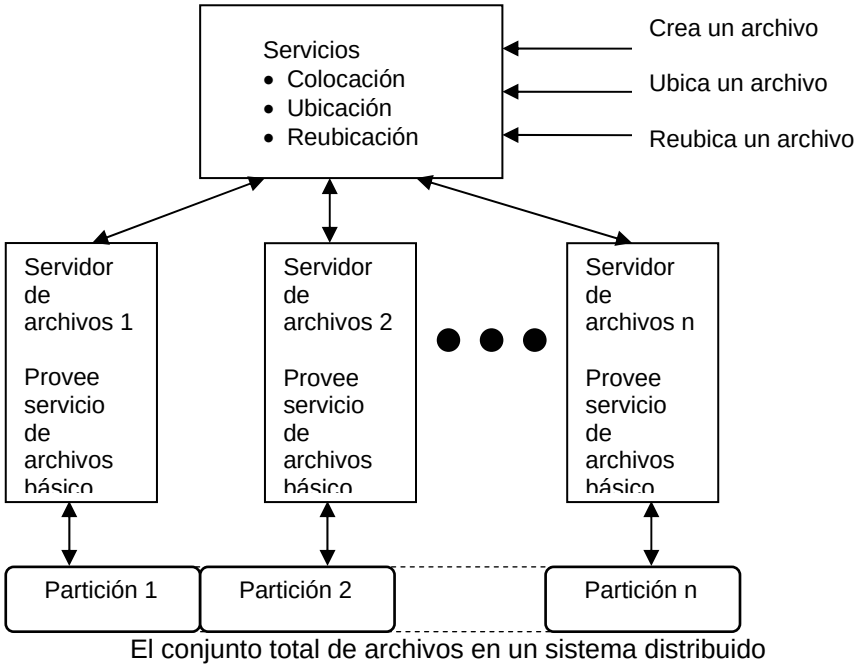


Figura 2.5 Esquema sistemático para resolver problemas de Colocación, Ubicación y Reubicación.

La figura muestra que la tarea básica de un servicio de archivos en el sistema distribuido con un número de servidores administrando las particiones del conjunto total de archivos es la de ubicar un archivo. Cada servidor sufre de los mismos problemas como si fueran un servidor de archivos centralizados: eficiencia precaria y bajo rendimiento.

En un servicio de archivos distribuido básico, cuándo un cliente crea un archivo, un servidor tiene que ser seleccionado para mantener el archivo. La selección puede ser realizada automáticamente o puede tomar en consideración las preferencias del usuario. En el caso del algoritmo de colocación automática, la ubicación se escoge para minimizar retardos de comunicación entre el cliente y servidor, o para optimizar la utilización de almacenamiento a través de todos los servidores. El servidor seleccionado genera un UFID nuevo el cual contiene su propio ID de servidor, y administra el archivo.

Para hacer referencia a un archivo, un cliente tiene que conocer la ubicación del servidor de archivos que administra el archivo requerido. No hay problema si el archivo no se mueve. Si un archivo puede moverse entre servidores, un algoritmo de ubicación debe usarse para determinar su ubicación actual. Esta información es retenida por el cliente para referencias posteriores hasta que el archivo es movido de nuevo. Acceso exitoso por parte del cliente nunca debe depender en mantener una variable de ubicación actual.

Para soportar colocación y ubicación las operaciones primitivas de la siguiente lista pueden ser utilizadas:

colocar(*archivo*)
: Encuentra el mejor servidor de archivos para el *archivo*.

ubicar(*archivo*)
: Encuentra la dirección del servidor que mantiene a *archivo*.

2.5 Resumen

Hay un número de servicios provistos por un sistema distribuido, pero uno de los más importantes es el servicio de archivos. En la práctica, los usuarios tienen contacto con el sistema de cómputo mediante el servicio de archivos. Este servicio es el principal servicio utilizado por los usuarios para acceder al total del sistema distribuido. Hay que tomar en cuenta que un servicio de archivos es necesario también para soportar la automatización de backup y recuperación.

Hay tres clasificaciones principales de servidores de archivos. El primero es el servidor de archivos verdadero, el cual es un sistema completamente de colocación, para servidores de almacenamiento. Otra clasificación nos habla de un servidor de archivos tradicionales, el cual es ofrecido por todos los sistemas operativos centralizados, y un servicio robusto, el cual es objetivo de aquellas aplicaciones que requieren una alta fiabilidad. El último servicio generalmente ofrece actualizaciones atómicas y características similares de las cuales carecen los sistemas de archivos tradicionales.

La tercer clasificación esta basada en acceso remoto a los archivos. Hay dos formas de acceso remoto a archivos: transferencia de archivos explicita en la cual

un cliente debe invocar una utilidad de transferencia de archivos para transferir un archivo remotamente antes o después de ser usado; y sistemas de archivos distribuidos en el cuales la ubicación de archivos es transparente a los clientes, a los archivos se les hace referencia por su nombre, la información de archivos y directorios es puesta en caché, y algunos archivos son replicados.

Los archivos representan un camino conveniente de ver los datos y programas desde la vista lógica de usuario y la vista física de almacenamiento. Esto implica que los archivos necesitan un método para el control a nivel de sistema.

Las siguientes funciones deberían ser realizadas en sistemas centralizados como también en un sistema distribuido: servicio de disco, servicio de archivo, y servicio de directorio. Estas tres funciones son usualmente realizadas por los siguientes módulos: dispositivo, bloque, archivo, control de acceso y directorio. Mientras que en un sistema centralizado estas tres funciones no son usualmente separadas, en un sistema distribuido hay una división de responsabilidades de estas tres funciones, tres servicios realizan estas funciones.

Cuando se trata con el servicio de disco, los siguientes tópicos deben ser considerados: almacenamiento de archivos, el sistema de caché, identificadores de archivos únicos, ubicación de archivos y operaciones del servicio de bloque. El diseño de un servicio de archivos plano lidia con los archivos mutables e inmutables y operaciones del servicio de archivos en datos y atributos. El servicio de directorio esta principalmente enfocado en realizar operaciones en el servicio de directorio, y mecanismos de control de acceso.

Debido a que muchos procesos cliente deberían permitir leer y escribir datos en el mismo archivo concurrentemente, el servicio de archivos debería soportar transacciones, y acciones atómicas deberían ser provistas. Esto requiere operaciones especiales del servicio de transacciones. Más aun, debido a la probabilidad de fallas, algunos procedimientos de recuperación deben ser implementados. Hay dos formas para realizar la recuperación: la aproximación por medio de listas de intención y la aproximación por medio de la versión de archivo.

Por razones de la conexión de varias LAN en un sistema distribuido, rendimiento y disponibilidad, es que existen los servidores de archivos de colaboración. Hay dos tipos de servidores de archivos de colaboración: aquellos que administran particiones del conjunto total de archivos en un sistema distribuido, y aquellos que soportan archivos replicados.

Los clientes de la primera clase no están alerta de la ubicación de los archivos. Esto genera tres problemas básicos: colocación, ubicación y reubicación. La consistencia de datos y el problema de actualizaciones múltiples de copias son los problemas con los que lidia el servicio de archivos replicados.

3

Código Fuente: Serpent

3.1 Diagrama Serpent

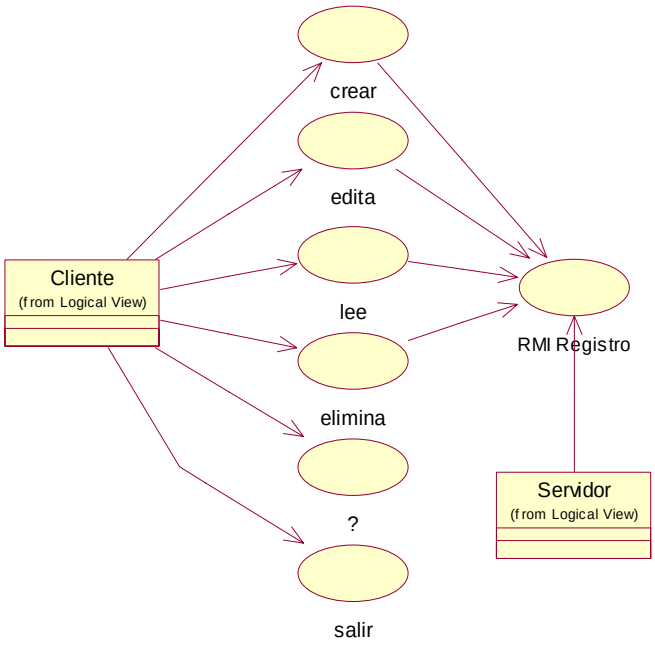


Figura 3.1 Diagrama de Casos de Uso

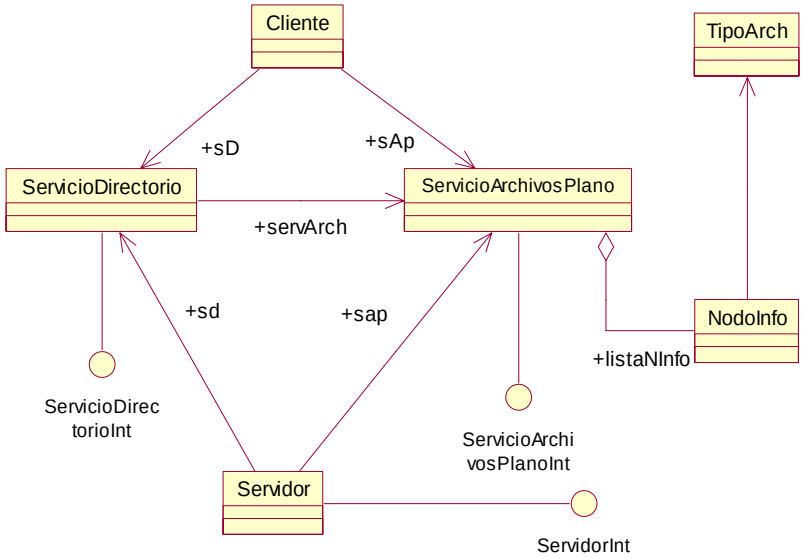


Figura 3.2 Diagrama de Clases

3.2 Cliente.java

```
//Programa: Cliente.java
//Autor: Cervantes Coronado Iván

import java.net.InetAddress;
import java.rmi.RemoteException;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import java.io.*;
import java.nio.*;

public class Cliente
{
	private static final int PORT = 2019;

	public static void main(String args[])
	{
		try
		{
			// Hacer la referencia al registro SSL-based
			Registry registry = LocateRegistry.getRegistry(
				InetAddress.getLocalHost().getHostName(), PORT,
					new RMISSLClientSocketFactory());

			// Objetos remotos
			ServicioDirectorioInt sD =
				(ServicioDirectorioInt) registry.lookup("SD");
			ServicioArchivosPlanoInt sAp =
				 (ServicioArchivosPlanoInt)registry.lookup("SAP");

			//iniciar directorio
			sD.SDIniciar( "lagarto" );

			BufferedReader in = new BufferedReader(
					new InputStreamReader(System.in));

			String[] s;
			String s2, s3;

			int opcion;
			int i = 0;

			//Cargar la lista del dir default
			s2 = new String();
			s2 = sD.busca( ".", "lagarto" );

			System.out.println("Funciona" + s2 );

			String linea[][] = sD.busca( s2 );
			for( i = 1; i < linea.length ; i++ )
			{
				if( linea[i][0] != null )
					sAp.crea( linea[i][0], linea[i][1] );
				else
					break;
			}
```

```
            //Llenar una lista con todos los archivos del dir actual .


                  while( true )
                  {
                        System.out.print( "root@localhost$ " );

                        s = in.readLine().split( "\\s" );

                        if( s[0].contentEquals( "crea" ) )
                              opcion = 1;
                        else if( s[0].contentEquals( "edita" ) )
                              opcion = 2;
                        else if( s[0].contentEquals( "lee" ) )
                              opcion = 3;
                        else if( s[0].contentEquals( "elimina" ) )
                              opcion = 4;
                        else if( s[ 0 ].contentEquals( "?" ) )
                              opcion = 5;
                        else if( s[ 0 ].contentEquals( "salir" ) )
                              opcion = 6;
                        else
                              opcion = 9;

                        switch( opcion )
                        {
                              case 1:

                              System.out.println( s[ 1 ] );
                                    // Una linea vacía termina el comando
                              sD.agregaNombre( ".", s[ 1 ], sAp.crea( s[ 1
] ) );


                              s = null;

                              break;

                              case 2:

                                    s3 = new String();
                                    while((s2 = in.readLine())!= null &&
s2.length() != 0 )
                                    s3 += s2 + "\n";

                                    // buscar el UFID
                                    s2 = sD.busca( ".", s[ 1 ] );

                                    if( s2 == null )
                                    {
                                          System.err.println( "El archivo
no existe!!" );


                                    }
                                    else
                                          sAp.escribe( s2, s3 );

                                    s2 = null;
```

```
                                    s3 = null;

                              break;

                        case 3:

                              s2 = new String();
                              ByteBuffer buff;

                              s2 = sD.busca( ".", s[ 1 ] );

                              byte[] bA = sAp.lee( s2 );

                              if( bA != null )
                              {
                                    buff = ByteBuffer.wrap( bA );
                                    buff.rewind();

                                    while(buff.hasRemaining())
                                           System.out.print(
(char)buff.get() );

                              }
                              else
                                    System.out.println( "El archivo no
existe!!" );

                              s = null;
                              s2 = null;

                              break;

                              case 4:
                                    s2 = new String();

                                    s2 = sD.desNombra( ".", s[ 1 ] );
                                    if( s2 == null )
                                          System.out.println( s[ 1 ] + "
No existe o alguna excepción" );
                                    else
                                    {
                                          sAp.elimina( s2 );
                                          System.out.println( s[ 1 ] + "
Eliminado!" );
                                    }

                                    s = null;
                                    s2 = null;

                                    break;

                              case 5:

                                    System.out.println( "Nombre" );
                                    System.out.println( "\tcrea" );
                                    System.out.println( "\nSinopsis" );
```

```
                                    System.out.println( "\tcrea archivo"
);
                                    System.out.println( "Descripción" );
                                    System.out.println( "\tEste comando
crea un archivo," +
                                    " el argumento archivo denota el
nombre del mismo. " +
                                    "No puede crearse un archivo sin
nombre." );
                                    System.out.println( "Nombre" );
                                    System.out.println( "\tedita" );
                                    System.out.println( "\nSinopsis" );
                                    System.out.println( "\tedita archivo"
);
                                    System.out.println( "Descripción" );
                                    System.out.println( "\tEste comando
edita el contenido del archivo," +
                                    " el argumento archivo denota el
nombre del mismo." );
                                    System.out.println( "Nombre" );
                                    System.out.println( "\tlee" );
                                    System.out.println( "\nSinopsis" );
                                    System.out.println( "\tlee archivo" );
                                    System.out.println( "Descripción" );
                                    System.out.println( "\tEste comando
lee el contenido de un archivo," +
                                    " el argumento archivo denota el
nombre del mismo. " +
                                    "No puede leerse un archivo sin nombre
o que no ha sido creado." );
                                    System.out.println( "Nombre" );
                                    System.out.println( "\telimina" );
                                    System.out.println( "\nSinopsis" );
                                    System.out.println( "\telimina
archivo" );
                                    System.out.println( "Descripción" );
                                    System.out.println( "\tEste comando
elimina un archivo," +
                                    " el argumento archivo denota el
nombre del mismo. " +
                                    "No puede eliminarse un archivo sin
nombre." );

                                    break;

                              case 6:
                                    System.exit(1);

                              default:

                                    System.out.println( "Ayuda ?" );
                              }
                        }

                  }
                  catch (Exception e)
                  {
```

```
                System.out.println("Servidor exception: " +
e.getMessage());
                e.printStackTrace();
            }
      }
}
```

3.3 Servidor.java

```
//Programa: Servidor.java
//Autor: Cervantes Coronado Iván

import java.io.*;
import java.net.InetAddress;
import java.rmi.RemoteException;
import java.rmi.RMISecurityManager;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import java.rmi.server.UnicastRemoteObject;

public class Servidor extends UnicastRemoteObject implements ServidorInt
{

      private static final int PORT = 2019;

      public Servidor() throws Exception
      {
            super(PORT,
            new RMISSLClientSocketFactory(),
            new RMISSLServerSocketFactory());

      }

      public ServicioArchivosPlano obtenerSAP() throws Exception,
RemoteException
      {
            return new ServicioArchivosPlano();
      }

      public ServicioDirectorio obtenerSD() throws Exception,
RemoteException
      {
            return new ServicioDirectorio();
      }

      public static void main(String args[]) throws Exception,
ClassNotFoundException
    {
            // Crear e instalar security manager
            if (System.getSecurityManager() == null)
            {
                  System.setSecurityManager(new RMISecurityManager());
            }

            try
            {
```

```
                  // Crear registro SSL-based
                Registry registry = LocateRegistry.createRegistry(PORT,
                      new RMISSLClientSocketFactory(),
                            new RMISSLServerSocketFactory());

                ServicioArchivosPlano sap =
                      new ServicioArchivosPlano();
                ServicioDirectorio sd = new ServicioDirectorio();

                registry.bind("SD", sd );
                registry.bind("SAP", sap );

            System.out.println("FileServer trabajando");
            }
            catch (Exception e)
            {
                  System.out.println("ServidorImpl err: " +
e.getMessage());
                  e.printStackTrace();
            }
      }
}
```

3.4 ServidorInt.java

```
//Programa: ServidorInt.java
//Autor: Cervantes Coronado Iván

import java.rmi.Remote;
import java.rmi.RemoteException;

public interface ServidorInt extends Remote
{
      ServicioArchivosPlano obtenerSAP() throws Exception,
RemoteException;
      ServicioDirectorio obtenerSD() throws Exception, RemoteException;
}
```

3.5 ServicioArchivosPlano.java

```
//Programa: ServicioArchivosPlano.java
//Autor: Cervantes Coronado Iván

import java.util.*;
import java.io.*;
import java.nio.*;
import java.nio.channels.*;
import java.rmi.server.UnicastRemoteObject;
import java.rmi.RemoteException;
import java.net.InetAddress;
import java.rmi.RMISecurityManager;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
```

```
public class ServicioArchivosPlano extends UnicastRemoteObject implements
ServicioArchivosPlanoInt, Serializable
{
	private final int BSIZE = 1024;
	private static final int PORT = 2019;

	NodoInfo[] listaNInfo = new NodoInfo[ 29999 ];

	public ServicioArchivosPlano() throws Exception, RemoteException
	{
		super(PORT,
			new RMISSLClientSocketFactory(),
			new RMISSLServerSocketFactory());

	}

	public String crea() throws Exception, IOException, RemoteException
	{
		listaNInfo[ NodoInfo.numNodoInfo - 1 ] = new NodoInfo();

		return listaNInfo[ NodoInfo.numNodoInfo - 1 ].UFID;
	}

	public String crea( String nombre ) throws Exception, IOException,
RemoteException
	{
		listaNInfo[ NodoInfo.numNodoInfo ] = new NodoInfo( nombre );

		return listaNInfo[ NodoInfo.numNodoInfo - 1 ].UFID;
	}

	public String crea( String UFID, String nombre ) throws Exception,
IOException, RemoteException
	{
		listaNInfo[ NodoInfo.numNodoInfo ] = new NodoInfo( UFID,
nombre );

		return listaNInfo[ NodoInfo.numNodoInfo - 1 ].UFID;
	}

	public byte[] lee( String idArch, int i, int n ) throws Exception,
IOException, RemoteException
	{
		// Hay que buscar el UFID del idArch
		RandomAccessFile fc;
		int numNodoInfo = buscar( idArch );

		fc = listaNInfo[ NodoInfo.numNodoInfo - 1 ].getArchivo();
		fc = new RandomAccessFile(
			listaNInfo[ NodoInfo.numNodoInfo - 1 ].getNombre(), "r"
);

		byte[] buff = new byte[ (int)fc.length() ];
		fc.read( buff, i, (int)fc.length() );
		fc.close();

		return buff;
```

```
	}

	public byte[] lee( String idArch ) throws Exception, IOException,
RemoteException
	{
		// Hay que buscar el UFID del idArch
		RandomAccessFile fc;
		int numNodoInfo = buscar( idArch );

		if( numNodoInfo != -1 )
		{
			fc = listaNInfo[ numNodoInfo ].getArchivo();
			fc = new RandomAccessFile(
				listaNInfo[ numNodoInfo ].getNombre(), "r" );

			byte[] buff = new byte[ (int)fc.length() ];
			fc.read( buff, 0, (int)fc.length() );
			fc.close();

			return buff;
		}
		else
			return null;
	}

	public String lee( String idArch, int linea ) throws Exception,
IOException, RemoteException
	{
		// Hay que buscar el UFID del idArch
		RandomAccessFile fc;
		int numNodoInfo = buscar( idArch );

		fc = listaNInfo[ numNodoInfo ].getArchivo();
		fc = new RandomAccessFile(
			listaNInfo[ numNodoInfo ].getNombre(), "r" );

		String s = new String();

		fc.seek( 0 );
		for( int i = 0; i < linea; i++ )
			s = fc.readLine();

		fc.close();

		return s;
	}

	public void escribe( String idArch, int i, byte[] datos ) throws
Exception, IOException, RemoteException
	{
		// Hay que buscar el UFID del idArch
		RandomAccessFile fc;
		int numNodoInfo = buscar( idArch );

		fc = listaNInfo[ NodoInfo.numNodoInfo -1 ].getArchivo();
		fc = new RandomAccessFile(
```

```
                  listaNInfo[ NodoInfo.numNodoInfo - 1 ].getNombre(),
"rw");
            fc.seek(i);
            fc.write( datos );
            fc.close();
      }

      public void escribe( String idArch, String datos ) throws
Exception, IOException, RemoteException
      {
            // Hay que buscar el UFID del idArch
            RandomAccessFile fc;
            int numNodoInfo = buscar( idArch );

            fc = listaNInfo[ numNodoInfo ].getArchivo();
            fc = new RandomAccessFile(
                  listaNInfo[ numNodoInfo ].getNombre(), "rw");
            fc.seek( fc.length() );
            fc.write( (datos + "\n").getBytes() );
            fc.close();
      }

      public void escribe( String idArch, String datos, int i ) throws
Exception, IOException, RemoteException
      {
            // Hay que buscar el UFID del idArch
            RandomAccessFile fc;
            int numNodoInfo = buscar( idArch );

            fc = listaNInfo[ numNodoInfo ].getArchivo();
            fc = new RandomAccessFile(
                  listaNInfo[ numNodoInfo ].getNombre(), "rw");
            fc.seek( 0 );

            long l = 0;
            for( int j = 1; j <= i; j++ )
            {

                  if( i == j )
                        fc.write( (datos).getBytes() );

                  fc.readLine();
            }

            fc.close();
      }

      public long obtenerTam( String idArch, String datos ) throws
Exception, IOException, RemoteException
      {
            long tam = 0;

            // Hay que buscar el UFID del idArch
            RandomAccessFile fc;

            int numNodoInfo = buscar( idArch );
```

```
            fc = listaNInfo[ numNodoInfo ].getArchivo();
            fc = new RandomAccessFile(
                  listaNInfo[ numNodoInfo ].getNombre(), "rw");
            tam = fc.length();
            fc.close();

            return tam;
      }

      public void elimina( String idArch ) throws Exception, IOException,
RemoteException
      {
            // elimina el archivo del almacen de archivos
            // Hay que buscar el UFID del idArch
            int numNodoInfo = buscar( idArch );

            File fc = new File(
                  listaNInfo[ numNodoInfo ].getNombre());

            // tambien eliminar de la lista de inodos fisica en
archivo!!!
            System.out.println( "Blur " + numNodoInfo + " " + listaNInfo[
numNodoInfo ].getNombre() );
            listaNInfo[ numNodoInfo ] = null;
            fc.delete();
      }

      public int buscar( String UFID ) throws RemoteException
      {
            for( int i = 0; i < listaNInfo.length; i++ )
            {
                  if( listaNInfo[ i ] == null )
                  {
                        continue;
                  }
                  else
                  {
                        if( listaNInfo[ i ].UFID.equals( UFID ) )
                        {
                              System.out.println( i + " " +listaNInfo[ i
].UFID);
                              return i;
                        }
                  }
            }

            return -1;
      }
}
```

3.6 ServicioArchivosPlanoInt.java

```
//Programa: ServicioArchivosPlanoInt.java
//Autor: Cervantes Coronado Iván
```

```
import java.util.*;
import java.io.*;
import java.nio.*;
import java.nio.channels.*;
import java.rmi.Remote;
import java.rmi.RemoteException;

public interface ServicioArchivosPlanoInt extends Remote
{
      public String crea() throws Exception, IOException,
RemoteException;
      public String crea( String nombre ) throws Exception, IOException,
RemoteException;
      public String crea( String UFID, String nombre )
            throws Exception, IOException, RemoteException;
      public byte[] lee( String idArch, int i, int n )
            throws Exception, IOException, RemoteException;
      public byte[] lee( String idArch ) throws Exception, IOException,
RemoteException;
      public String lee( String idArch, int linea )
            throws Exception, IOException, RemoteException;
      public void escribe( String idArch, int i, byte[] datos )
            throws Exception, IOException, RemoteException;
      public void escribe( String idArch, String datos )
            throws Exception, IOException, RemoteException;
      public void escribe( String idArch, String datos, int i )
            throws Exception, IOException, RemoteException;
      public long obtenerTam( String idArch, String datos )
            throws Exception, IOException, RemoteException;
      public void elimina( String idArch ) throws Exception, IOException,
RemoteException;
      public int buscar( String UFID ) throws RemoteException;

}
```

3.7 ServicioDirectorio.java

```
//Programa: ServicioDirectorio.java
//Autor: Cervantes Coronado Iván

import java.io.*;
import java.rmi.server.UnicastRemoteObject;
import java.rmi.RemoteException;
import java.rmi.RMISecurityManager;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;

// Relaciona los nombres en texto con los identificadores internos de los
archivos
public class ServicioDirectorio extends UnicastRemoteObject implements
ServicioDirectorioInt, Serializable
{
      private static final int PORT = 2019;

      ServicioArchivosPlano servArch = new ServicioArchivosPlano();
      String raiz = new String();
```

```
	public ServicioDirectorio()
		throws Exception, IOException, RemoteException
	{
		super(PORT,
			new RMISSLClientSocketFactory(),
			new RMISSLServerSocketFactory());
	}

	public void SDIniciar( String dir )
		throws Exception, IOException, RemoteException
	{
		raiz = servArch.crea( dir );

		if( servArch.obtenerTam( raiz, raiz + " " + "lagarto" ) != 0
);
		else
		{
			servArch.escribe( raiz, raiz + " " + "lagarto" );
		}
	}

	public String[][] busca( String dir )
		throws Exception, IOException, RemoteException
	{
		String s;
		int i = 1;
		String pup[][] = new String[1024][1];

		while( ( s = servArch.lee( raiz, i ) ) != null )
		{
			if( s.equals( null ) )
				continue;
			else
			{
				pup[i++] = s.split("\\s");
			}
		}

		return pup;
	}

	public String busca( String dir, String nombreArch )
		throws Exception, IOException, RemoteException
	{
		String s;
		int i = 1;

		while( ( s = servArch.lee( raiz, i++ ) ) != null )
		{
			if( s.equals( null ) )
				continue;
			else
			{
				String[] pup = s.split("\\s");
				System.out.println( "pup" + pup[0] + pup[1] +
nombreArch );
```

```
                        if( pup[1].contentEquals( nombreArch ) )
                        {
                              System.out.println( "*K" + pup[0] + "*K" );
                              return pup[0];
                        }
                  }
            }

            return null;
      }

      public String buscaToEli( String dir, String nombreArch )
            throws Exception, IOException, RemoteException
      {
            String s;
            int i = 1;

            while( ( s = servArch.lee( raiz, i++ ) ) != null )
            {
                  if( s.equals( null ) )
                        continue;
                  else
                  {
                        String[] pup = s.split("\\s");
                        System.out.println( "pup" + pup[0] + pup[1] +
nombreArch );

                        if( pup[1].contentEquals( nombreArch ) )
                        {
                              // Eliminar registro del directorio
                              servArch.escribe( raiz, "-1" + " " + "", --i
);

                              return pup[ 0 ];
                        }
                  }
            }

            return null;
      }

      public boolean agregaNombre( String dir, String nombreArch, String
idArch )
            throws Exception, IOException, RemoteException
      {
            String s = new String();
            s = busca( dir, nombreArch );

            if( s != null )
                  return false;
            else
            {
                  servArch.escribe( raiz , idArch + " " + nombreArch );
                  return true;
            }
      }
```

```
	public byte[] lista( String UFID )throws Exception, IOException,
RemoteException
	{
		return servArch.lee( UFID );
	}

	public String desNombra( String dir, String nombreArch ) throws
Exception, RemoteException
	{
		return buscaToEli( dir, nombreArch );
	}

	public String[][] dameNombres( String dir )throws Exception,
IOException, RemoteException
	{
		return busca( dir );
	}
}
```

3.8 ServicioDirectorioInt.java

```
//Programa: ServicioDirectorioInt.java
//Autor: Cervantes Coronado Iván

import java.io.*;
import java.rmi.Remote;
import java.rmi.RemoteException;

public interface ServicioDirectorioInt extends Remote
{
	public void SDIniciar( String dir )
		throws Exception, IOException, RemoteException;
	public String[][] busca( String dir )
		throws Exception, IOException, RemoteException;
	public String busca( String dir, String nombreArch )
		throws Exception, IOException, RemoteException;
	public String buscaToEli( String dir, String nombreArch )
		throws Exception, IOException, RemoteException;
	public boolean agregaNombre( String dir, String nombreArch, String
idArch )
		throws Exception, IOException, RemoteException;
	public byte[] lista( String UFID )throws Exception, IOException,
RemoteException;
	public String desNombra( String dir, String nombreArch ) throws
Exception, RemoteException;
	public String[][] dameNombres( String dir )
		throws Exception, IOException, RemoteException;
}
```

3.9 NodoInfo.java

```
//Programa: NodoInfo.java
//Autor: Cervantes Coronado Iván
```

```
import java.util.*;
import java.io.*;
import java.nio.*;
import java.nio.channels.*;

class NodoInfo
{
      String UFID;

      static int numNodoInfo = 0;
      private int contRef;    // contador de referencias

      private Date tCreacion;
      private Date tUltAcceso;
      private Date tUltEscritura;
      private Date tUltLectura;
      private Date tUltCambAtrib;

      String nombreArch;
      private String propietario;

      private TipoArch tipo;

      // Lista de control de acceso
      // El objeto que le da nombre a este trabajo
      private RandomAccessFile fc;

      NodoInfo() throws Exception
      {
            // Generar semilla
            Random rand = new Random();

            nombreArch = Integer.toString( ++numNodoInfo ) +
rand.nextInt( 299999 );

            UFID = Integer.toString( rand.nextInt( 23999999 ) );
            fc = new RandomAccessFile( nombreArch, "rw" );

            setTCreacion(); setTUltAcceso(); setTUltEscritura();
            setTUltLectura(); setTUltCambAtrib();

            //numNodoInfo++;
            contRef++;

            fc.close();
      }

      NodoInfo( String nombre ) throws Exception
      {
            // Generar semilla
            Random rand = new Random();

            nombreArch = nombre;

            UFID = Integer.toString( rand.nextInt( 23999999 ) );
            fc = new RandomAccessFile( nombreArch, "rw" );
```

```
        setTCreacion(); setTUltAcceso(); setTUltEscritura();
        setTUltLectura(); setTUltCambAtrib();

        numNodoInfo++;
        contRef++;

        fc.close();
    }

    NodoInfo( String UFID, String nombre ) throws Exception
    {
        // Generar semilla
        Random rand = new Random();

        nombreArch = nombre;

        this.UFID = UFID;
        fc = new RandomAccessFile( nombreArch, "rw" );

        setTCreacion(); setTUltAcceso(); setTUltEscritura();
        setTUltLectura(); setTUltCambAtrib();

        numNodoInfo++;
        contRef++;

        fc.close();
    }

    // Metodos de asignación!!!!!!!!
    private void setTCreacion()
    {
        tCreacion = new Date();
    }

    private void setTUltAcceso()
    {
        tUltAcceso = new Date();
    }

    private void setTUltEscritura()
    {
        tUltEscritura = new Date();
    }

    private void setTUltLectura()
    {
        tUltLectura = new Date();
    }

    private void setTUltCambAtrib()
    {
        tUltCambAtrib = new Date();
    }

    void setPropietario( String propi )
    {
        propietario = propi;
```

```
            // setTUltCambAtrib();
      }

      void setTipoArchivo( int num )
      {
            tipo = TipoArch.numero( num );
            // setTUltCambAtrib();
      }

      // Métodos de obtención
      RandomAccessFile getArchivo()
      {
            return fc;
      }

      String getNombre()
      {
            return nombreArch;
      }
}
```

3.10 TipoArch.java

```
//Programa: TipoArch.java
//Autor: Cervantes Coronado Iván

public final class TipoArch
{
      private String nombre;

      private TipoArch( String nom )
      {
            nombre = nom;
      }

      public String toString()
      {
            return nombre;
      }

      public static final TipoArch
            TEX = new TipoArch( "Texto" ),
            DIR = new TipoArch( "Directorio" ),
            EXE = new TipoArch( "Ejecutable" );

      public static final TipoArch[] tipoArch = {
            TEX, DIR, EXE
      };

      public static final TipoArch numero( int ord )
      {
            return tipoArch[ ord - 1 ];
      }
}
```

3.11 policy

```
/* AUTOMATICALLY GENERATED ON Wed Feb 08 18:07:17 CST 2006*/
/* DO NOT EDIT */

grant {
  permission java.net.SocketPermission "*", "accept, connect, listen,
resolve";
  permission java.io.FilePermission "<<ALL FILES>>", "read, write,
delete, execute";
};
```

3.12 Instalar

- Todos los archivos con terminación ".java" deben ser puestos en un mismo directorio (los aquí citados mas los siguientes: RMISSLServerSocketFactory.java y RMISSLClientSocketFactory.java 29).
- Los archivos policy, samplecacerts y testkeys[28], deben estar en el mismo directorio que los archivos con terminación ".class".
- Para compilar ejecutamos la siguiente instrucción (doy por hecho la propia configuración del CLASSPATH):
  - javac *.java
- Si se tiene un jdk < jdk1.5 ejecutamos lo siguiente:
  - rmic Servidor ServicioArchivosPlano ServicioDirectorio
- Para ejecutar el *Servidor* damos el siguiente comando al shell:
  - java -Djava.security.policy=policy Djavax.net.ssl.trustStore=samplecacerts[29] Servidor &
- Ya que tenemos corriendo el *Servidor* como demonio ejecutamos la siguiente instrucción para darle vida al *Cliente*:
  - java -Djavax.net.ssl.trustStore=samplecacerts Cliente
- Existe una opción de ejecución con la que podemos ver el flujo cifrado de SSL tanto para el cliente como para el Servidor.
  - java **-Djavax.net.debug=ssl** -Djava.security.policy=policy -Djavax.net.ssl.trustStore=samplecacerts Cliente

[28] Estos tres archivos pueden encontrarse en la documentación de RMI del JDK1.5
[29] Ver documentación del JDK1.5 (www.java.sun.com)

Conclusión

Saber qué es un sistema distribuido es un excelente comienzo para involucrarse con el tema de ésta tesis, claro que se necesita de un respaldo de conocimientos en cómputo, pero una vez tomada la foto global podemos involucrarnos en lo profundo de cada pieza que compone la obra. Una de esas piezas fue la expuesta en este trabajo profesional con la profundidad deseada. Me refiero al sistema de archivos el cual fue desmenuzado como un proceso el cual es diseminado en varios hilos de proceso (threads). Como los jóvenes y también veteranos científicos se preguntaran: “¿pero si un thread también puede ser diseminado?”, a lo que respondo en efecto los subtemas del capítulo 1 y 2 pueden fragmentarse y verse de forma más profunda, pero el enfoque de este trabajo no era ese. Por otra parte es importante saber que la necesidad de manipular la información se ve reflejada en el sistema de archivos distribuido.

La implementación que se desarrollo en esta tesis esta enfocada en el paradigma orientado a objetos y en el lenguaje de programación Java (el cual se toma mejor leyendo un buen libro).

Los conceptos aquí plasmados y su ejemplo generan una documentación básica en el cómputo distribuido, por lo que espero esta documentación realice su tarea por la cual fue creada.

No quiero terminar sin sugerir que se reflexione y resuelva la falta de generación de ciencia y tecnología en cómputo de este plantel.

Cervantes Coronado Iván Christian

Bibliografía

GALLI, L. Doreen.
Distributed Operating Systems – Concepts and Practice,
Ed. Prentice Hall, 2000.

STALLINGS, William.
Comunicaciones y redes de computadores,
España. Ed. Pearson - Prentice Hall, 2004.

WU, Jie.
Distributed System Design,
E.U.A. Ed. CRC Press. 1999.

GOSINSKI A.
Distributed Operating Systems – The Logical Design,
Ed. Addison-Wesley. 1991.

COULOURIS George, DOLLIMORE Jean y KINDBERG Tim.
Distributed Systems Concepts and Design,
Ed. Addison-Wesley. 2005.

Harvey M. Deitel and Paul J. Deitel.
Java How to Program,
Ed. Prentice Hall. 2005.

STINSON R. Douglas.
Cryptography – Theory and Practice,
E.U.A. Ed. Chapman & Hall/CRC. 2002.

LANGSAM Yedidyah, AUGENSTEIN J. Moshe y TENENBAUM M. Aaron.
Estructuras de Datos con C y C++,
México. Ed. Prentice Hall. 1997.

LEACH P. J., LEVINE P. H., HAMILTON J. A., y STUMPF B. L.
The File System of an Integrated Local Network,
ACM Computer Science Conference, New Orleans. 1985.

BOVET P. Daniel y CESATI Marco.
Understanding the Linux Kernel,
Ed. O'Reilly. 2000.

GIFFORD D. K., NEEDHAM R. M. y SCHREADER M. D.
The Ceder File System,
Communications of the ACM. 1988.

SANDBERG R.
The Sun Network File System: Design, Implementation and Experience,
Ed. Sun Microsystems, Inc. 1986.

BRABAN B. y SCHLENK P.
A *Well Structured Parallel File System for PM*,
Operating Systems Review. 1989.

MUKHERJEE A., KRAMER J. y MAGGE J. (1988).
A Distributed File Server for Embedded Applications,
Softwrae Engineering Journal. 1988.

LEVY E. y SILBERSCHATZ A.
Distributed File Systems: Concepts and Examples,
TR-89-04 Department of Computer Sciences, The University of Texas at Austin. 1989.

REESE George.
Database Programming with JDBC™ and Java™,
Ed. O'Reilly. 2000.

SILBERSCHATZ Abraham, JAMES L. Peterson y PETER B. Galvin.
Sistemas operativos - conceptos fundamentales,
Ed. Addison-Wesley Longman. 1994.

SILBERSCHATZ Abraham y PETER B. Galvin.
Sistemas Operativos,
México. Ed. Pearson - Addison-Wesley Longman. 1999.

Sitio Oficial de la tecnología Java www.java.sun.com

Sitio Oficial de MindView www.mindview.net/Books