Tema 5: Sistemas de Ficheros Distribuidos.

5.1. Diseño de Sistemas de Ficheros Distribuidos.

- **5.1.1. La interfaz del servidor de ficheros.**
- **5.1.2. La interfaz del servidor de directorios.**
- **5.1.3. Semántica de los ficheros compartidos.**

5.2. Implementación de un Sistema de Ficheros Distribuido.

- **5.2.1. Uso de ficheros.**
- **5.2.2. Estructura del sistema.**
- **5.2.3. Ocultamiento.**
- **5.2.4. Réplica.**
- **5.2.5. Un ejemplo: el sistema de ficheros Andrew.**
- **5.2.6. Lecciones aprendidas.**

5.3. Tendencias en los Sistemas de Ficheros Distribuidos.

- **5.3.1. Hardware reciente.**
- **5.3.2. Escalabilidad.**
- **5.3.3. Redes de área amplia.**
- **5.3.4. Usuarios móviles.**
- **5.3.5. Tolerancia a fallos.**

Bibliografía:

- Andrew S. Tanenbaum: “Sistemas Operativos Distribuidos”; tema 5, Prentice-Hall, 1996.
- Coulouris, George: “Sistemas Distribuidos: conceptos y diseño”; tema , Addison-Wesley, 2001

Tema 5: Sistemas de Ficheros Distribuidos.

- En un sistema distribuidos se debe distinguir entre:
  - **servicio de ficheros**: especificación de los servicios que el Sistema de Ficheros ofrece a sus clientes. Especifica la interfaz del Sistema de Ficheros con los clientes.
  - **servidor o despachador de ficheros**: es un proceso que se ejecuta en alguna máquina y que implementa el servicio de ficheros. Un sistema puede tener varios servidores. El cliente desconoce el nº de servidores y dónde se encuentra ubicados
- un sistema puede contener varios servidores de ficheros con servicios distintos (puede ofrecer ficheros Unix y DOS).

5.1. Diseño de Sistemas de Ficheros Distribuidos.

- Un sistema distribuido de ficheros tiene dos componentes: el servicio de ficheros y el servicio de directorios.

5.1.1. La interfaz del servidor de ficheros.

- ¿Qué es un fichero?.
  - UNIX y DOS, un fichero es una secuencia de bytes sin interpretación. El significado y estructura de la información queda a cargo de los programas de aplicación.
  - los mainframes existen muchos tipos de ficheros: serie de registros, B-árbol, tablas de dispersión para su localización.

Tema 5: Sistemas de Ficheros Distribuidos.

- Los ficheros pueden tener **atributos**: información relativa al fichero (propietario, tamaño, fecha de creación y permiso de acceso). Se disponen de primitivas para cambiar estos atributos y en algunos casos para crear nuevos atributos.
- Los ficheros normalmente se pueden modificar. En algunos sistemas distribuidos las únicas operaciones son CREATE y READ (ficheros inmutables). Evita la actualización de diversas copias de ficheros.
- la protección en sistemas distribuidos emplea las mismas técnicas: **capacidades** y **listas de control**.
- Las capacidades determinan un conjunto de derechos sobre determinados objetos, un sujeto puede tener varias capacidades.
- Las listas de control de acceso asocian a cada fichero una lista explícita o implícita de los usuarios que pueden tener acceso al fichero y los tipos de accesos permitidos.
  - UNIX es una lista de control de acceso simplificada.
- Los servicios de ficheros se pueden dividir en dos tipos:
- modelo carga/descarga: sólo proporciona la lectura y la escritura completa del fichero que se vuelca en la memoria o disco local.
  - es sencillo y eficiente.
  - el cliente debe disponer de espacio suficiente.
  - si el cliente sólo necesita una fracción, el traslado completo es un desperdicio.
- modelo de acceso remoto: proporciona operaciones de abrir y cerrar ficheros, leer y escribir partes de ficheros, moverse a través dentro del fichero, examinar y modificar atributos...

Tema 5: Sistemas de Ficheros Distribuidos.

Matriz de Accesos

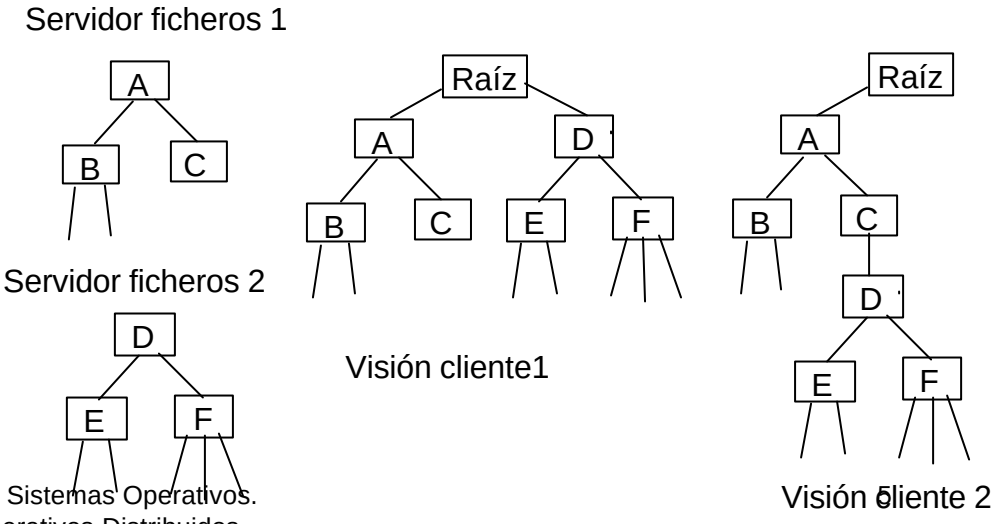
Objeto/ Dominio	Objeto1	Objeto2	Objeto3	Objeto4	Objeto5
Dominio1	Lectura Escritura		Ejecución		Salida
Dominio2		Lectura			
Dominio3				Lectura Escritura Copia	
Dominio4					Salida

- Interpretación por filas: **Capacidades**, acceso por sujeto o dominio
- Interpretación por columnas: **Lista de control**, acceso por objeto

Tema 5: Sistemas de Ficheros Distribuidos.

5.1.2. La interfaz del servidor de directorios.

- El servicio de directorios proporciona: operaciones para crear y eliminar directorios, nombrar o cambiar el nombre de ficheros y moverlos entre ellos.
- Define un alfabeto y sintaxis para formar los nombres de ficheros y directorios.
- Permiten la creación de subdirectorios; sistema jerárquico de ficheros.
- En ciertos sistemas se pueden utilizar enlaces que rompen esta estructura jerárquica. Se necesita un nuevo atributo que cuente el nº de enlaces. En los sistemas distribuidos es muy peligroso perder la pista de parte del sistema de ficheros por encontrarse repartido entre varias máquinas.
- ¿Todas las máquinas y procesos en un Sistema Distribuido deben tener la misma versión de la jerarquía de directorios?
- En los sistemas que manejan varios servidores de ficheros mediante montaje remoto es normal que cada cliente tenga una visión distinta del sistema ficheros
- Un forma de tener un directorio global es que obligar a que esa raíz sólo contenga una entrada por cada servidor (*/servidor/ruta*).



Tema 5: Sistemas de Ficheros Distribuidos.

Transparencia de los nombres

- Debemos distinguir dos formas de transparencia:
  - Transparencia con respecto a la posición; ningún nombre de ruta debe sugerir la posición de un objeto. Ej) */servidor1/dir1/dir2/x* indica que x está en el servidor1, pero no indica la posición del servidor. En principio es transparente.
  - Independencia con respecto a la posición: los ficheros pueden desplazarse sin cambiar sus rutas de acceso. Ej) */servidor1/dir1/dir2/x* . Si el fichero *x* necesita ser desplazado al servidor 2 su ruta de acceso cambiará y los programas que tengan el camino anterior no funcionarán.
  - No es fácil de lograr pero deseable en los Sistemas Distribuidos.
- Existe tres métodos para nombrar los ficheros y directorios en un S.D.:
  - Nombre máquina + ruta acceso, *maquina/ruta* ó *maquina:ruta*
  - Montaje de sistemas de ficheros remotos en la jerarquía local de ficheros.
  - Un único espacio de nombres que tenga la misma apariencia en todas las máquinas.
- Los dos primeros son fáciles de implantar y se puede utilizar los sistemas existentes.
- El tercer método es difícil y requiere un diseño cuidadoso pero es necesario si queremos que el Sistema Distribuido actúe como una única máquina.

Tema 5: Sistemas de Ficheros Distribuidos.

Nombres de dos niveles

- La mayoría de los Sistemas Distribuidos tienen dos modos de identificar un objeto: nombres simbólicos (utilizados por el usuario) y los nombres binarios (utilizados por el sistema). En UNIX este nombre es el identificador del i-nodo.
- En un esquema más general, el nombre binario indica el servidor y un fichero específico de ese servidor.
- Para referenciar ficheros de otro servidor también podemos emplear un *enlace simbólico*: una entrada en el directorio con una cadena con el camino completo (servidor, ruta).
- Se puede utilizar el nº binario para indicar un nº físico o lógico de una máquina o la dirección de red del servidor.
- Algunos Sistemas Distribuidos obtienen varios nombres binarios, el fichero original y todos sus respaldos. Se logra cierta tolerancia a fallos mediante la redundancia

5.1.3. Semántica de los ficheros compartidos.

- En un sistema de memoria común es fácil mantener la información de ficheros compartidos de manera coherente.
- En Sistemas Distribuidos se puede lograr esta semántica si existe un sólo servidor de ficheros siempre que la red no introduzca retrasos al hacer sus entregas, es decir, mantenga el orden de entrega de las peticiones.
- Sin embargo un sistema con un servidor de ficheros centralizado es pobre. Podemos mantener copias de los ficheros en las *cachés* locales pero entonces es posible obtener un dato obsoleto.
  - Se puede propagar de manera inmediata las modificaciones pero es ineficiente!!.

Tema 5: Sistemas de Ficheros Distribuidos.

- Se puede relajar la semántica de ficheros compartidos. Los cambios serán visibles cuando se cierre el fichero = Semántica de sesión.
  - ¿qué ocurre si dos o más clientes ocultan y modifican el mismo fichero simultáneamente?
  - Se puede no hacer nada.
  - Al cerrar el fichero, se envía su valor al servidor; luego quién antes termina es quién proporciona el valor.
- El uso de cachés y semántica de sesión viola la semántica de UNIX. Ya no es posible compartir por los procesos hijos que se ejecutan en ? máquinas el apuntador de posición actual del fichero.

  Ej) si *script* es un guión del shell que ejecuta *a* y *b*, la orden *script>salida* no funcionará con la semántica de sesión.
- Podemos evitar el problema completamente imponiendo ficheros inmutables. Se pueden reemplazar los ficheros, es decir, crear un fichero con el mismo nombre.
  - Ahora el problema: ¿qué hacer cuando dos procesos intentan a la vez reemplazar el mismo fichero?. Semántica de sesión: que uno de los nuevos ficheros reemplace al anterior.
  - Otro problema: ¿qué hacer si un fichero reemplaza mientras otro proceso está leyéndolo?. Podemos ofrecerle la anterior versión o hacer que falle su intento de lectura.
- Podemos utilizar las transacciones atómicas para resolver el uso de ficheros compartidos.

Tema 5: Sistemas de Ficheros Distribuidos.

Método	**Comentarios**
Semántica de Unix	• Cada operación en un fichero es visible a todos los procesos de manera inmediata
Semántica de sesión	• Ningún cambio es visible a otros procesos hasta que el fichero se cierra
Ficheros inmutables	• No existen actualizadores; es más fácil compartir
Transacciones	• Todos los cambios tienen la propiedad del todo o nada

Tema 5: Sistemas de Ficheros Distribuidos.

5.2. Implementación de un Sistema de Ficheros Distribuido.

5.2.1. Uso de ficheros.

- Satyanarayanan (81) hizo un estudio de los patrones de uso de los ficheros
- Algunas de las mediciones son estáticas, son una instantánea del sistema. Al examinar el disco determinaremos los tamaños de los ficheros, la distribución de tipo de ficheros y la cantidad de espacio que ocupan los ficheros de varios tamaños y varios tipos.
- Otras mediciones son dinámicas; al modificar el sistema de ficheros se registra en una bitácora: frecuencia relativa de determinadas operaciones, el nº de ficheros abiertos y la cantidad de usos compartidos.

Problemas:

- ¿cómo de representativa es la muestra tomada?.
- se debe tener cuidado con el tipo de características que se miden.
- las mediciones que hizo Satyanarayanan se realizaron sobre UNIX tradicional y NO sabemos si se puede extrapolar a los S.D.

Resultados:

- La mayoría de los ficheros < 10K. Apoya la idea de trasladar ficheros completos entre servidor y cliente.
- La lectura es más común que la escritura.
- Los ficheros suelen tener un tiempo de vida corto. Sería bueno crearlos en el cliente y mantenerlos hasta su terminación. Elimina el tráfico en la red.
- Es poco usual compartir ficheros. Esto está a favor del ocultamiento por parte del cliente, a pesar de complicar la semántica.
- Los procesos en promedio sólo utilizan unos cuantos ficheros.
- Existen distintas clases de ficheros con propiedades diferentes. Deberíamos tratar cada clase por separado.

Tema 5: Sistemas de Ficheros Distribuidos.

5.2.2. Estructura del sistema.

- Organización interna de los servidores de ficheros y directorios.
- ¿Son distintos los clientes y los servidores? No existe un acuerdo.
- NFS no distingue entre cliente y servidor. Todas las máquinas ejecutan el mismo software básico.
- En otros, el servidor de ficheros y directorios son sólo programas del usuario. El sistema se puede configurar para que ejecute o no el software cliente o servidor en la misma máquina.
- Sistemas en los que los servidores y clientes son máquinas distintas en términos hardware o software. Aunque la separación es más transparente no existen razones para preferir un determinado método.
- Organización de servidor de ficheros y directorios
- Combinar los dos en un único servidor.
- Separarlos. Las dos funciones no tienen un relación real. Ej) es posible tener dos servidores de directorios uno para UNIX y otro para DOS y un único servidor de ficheros para gestionar su almacenamiento físico. Contar con dos servidores requiere mayor comunicación.
- ¿qué hacer cuando la jerarquía de directorios se reparta entre varios servidores?
- Podemos notificar al cliente que esa entrada está en otro servidor y que él busque por su cuenta el resto de la ruta. El cliente debe ser consciente de que un servidor contiene un determinado directorio (más mensajes).
- El servidor puede encargarse de completar la búsqueda. Más eficiente pero no se puede manejar con una RPC normal.

Tema 5: Sistemas de Ficheros Distribuidos.

- La búsqueda de rutas con varios servidores puede ser muy cara.
- Se pueden mantener cachés de indicadores, nombres buscados recientemente y los resultados de estas búsquedas.
- Para que funcione el ocultamiento de nombres es necesario actualizar las cachés cuando los ficheros estén obsoletos.
- ¿los servidores de ficheros y directorios deben contener información de estado de los clientes?
- **Sin estado**. Cuando un servidor recibe una solicitud, la sirve y elimina la información relativa a la solicitud.
  - Debemos proporcionar toda la información necesaria. Nombre del fichero y posición dentro del mismo. Aumenta la longitud del mensaje.
  - Son más tolerantes a los fallos.
  - No son necesarias las operaciones OPEN y CLOSE.
  - No se desperdicia el espacio del servidor en tablas.
  - No existe límite en cuanto al nº de ficheros abiertos.
- **Con estado**. Seguir el método tradicional, se mantiene una tabla que vincula los descriptores de fichero con los ficheros reales.
  - Si un servidor falla, se pierde toda la información de estado. Fracasarán todos los intentos posteriores de lectura y escritura.
  - Mensajes de solicitud más cortos.
  - Mejor desempeño.
  - Es posible la lectura adelantada.
  - Es posible la cerradura de ficheros.

Tema 5: Sistemas de Ficheros Distribuidos.

5.2.3. Ocultamiento.

- En un sistema cliente-servidor (con memoria y disco) podemos alojar los ficheros o partes de ficheros en: disco servidor, la M.P. del servidor, el disco del cliente y la M.P. del cliente.
- El lugar más directo: disco servidor. Ficheros accesibles por todos los clientes, una sóla copia del fichero, no existe inconsistencia.

  Problema: rendimiento. El fichero debe ser transferido de disco de servidor a memoria del servidor y de aquí a memoria del cliente (vía red).
- Se puede lograr un mejor rendimiento si se ocultan; si se conservan los ficheros de uso más reciente en M.P. del servidor.
  - Se necesita un algoritmo que determine qué ficheros llevar a memoria.
  - Este algoritmo debe solucionar dos problemas:

1. Unidad que maneja en caché: ficheros completos o bloques disco.
2. ¿Qué hacer cuando la caché está ocupada?. Se puede utilizar cualquier algoritmo de reemplazo (LRU).
   - Es transparente a los clientes porque el servidor resuelve los problemas de inconsistencia sincronizando memoria y disco.

- Seguimos teniendo el acceso a la red. Podemos hacer el ocultamiento en el cliente.
  - ¿Utilizar el disco o la memoria?. Problema espacio vs. desempeño. Lo normal es almacenar los datos en memoria

Tema 5: Sistemas de Ficheros Distribuidos.

- Tres opciones para colocar la caché en Memoria:

1. En el espacio de un proceso de usuario. Tiene un costo mínimo, sólo eficaz si los procesos abren y cierran ficheros varias veces.
2. En el núcleo. Siempre hay que llamar al núcleo pero el fichero sobrevive al proceso. Esto es práctico cuando un fichero es utilizado por más de un proceso. Puede decidir dinámicamente la cantidad de memoria que debe reservar para programas y caché.
3. En un proceso manejador del caché en el espacio del usuario. Libera al núcleo del sistema de ficheros, es más fácil de programar. Si se ejecuta en un sistema con memoria virtual, el núcleo puede decidir si lleva toda la caché o parte a memoria; esto está en contra del ocultamiento!!!. El manejador de caché debería bloquear páginas de Memoria.

- Si las RPC son rápidas (CPU rápidas) y las transferencias en la red lentas, el ocultamiento es un factor importante para el rendimiento.

Consistencia de la caché

- El ocultamiento introduce inconsistencia!!!
- Se puede evitar mediante la semántica de sesión (los efectos de una modificación no son visibles hasta el cierre). Esto es “incorrecto” bajo UNIX
- Existe un problema insoluble: dos ficheros escriben de nuevo en el servidor, el último machaca la información del primero. El ocultamiento del cliente debe pensarse con cuidado!!!.
- Soluciones:

Tema 5: Sistemas de Ficheros Distribuidos.

1. Algoritmo **escritura a través de caché**: cuando se modifica una entrada en el caché el nuevo valor se envía de inmediato al servidor. Esto no evita del todo la inconsistencia: otra máquina cuya caché tuviera el mismo fichero no sabe de la actualización de ese fichero en el servidor.
   - Se puede exigir al manejador de caché que verifique el servidor antes de proporcionar el fichero al cliente. Notifica si tiene la última versión (RPC).
2. Algoritmo **escritura retrasada**: El algoritmo anterior sólo nos reduce el tráfico de la red para las lecturas. Se pueden acumular actualizaciones y hacerlas efectivas cada cierto tiempo. Así evitamos el tráfico para ficheros temporales.
   - El retraso de la escritura obscurece la semántica.
3. Algoritmo **escritura al cierre**: adoptar la semántica de sesión e incluso esperar 30 seg. más por si desaparece en ese lapso. El problema achacable a este algoritmo también puede ocurrir en sistema de una única CPU.
4. Algoritmo **control centralizado**: al abrir un fichero, la máquina envía un mensaje al servidor. El servidor mantiene un registro de los ficheros abiertos, poseedores y el tipo de apertura (lectura, escritura, ambos). El servidor debe hacer incompatible la apertura de un fichero para escritura con otras escrituras y con lecturas.
   - Para hacer posible el acceso simultáneo se puede enviar un mensaje desde el servidor a los clientes que tienen ficheros abiertos para que eliminen estos de sus cachés (sin ocultamiento) y todos acudan a él. No es elegante este proceder porque se invierte el papel del servidor y cliente.
   - Tiene un único punto de fallo y no es escalable.

Resumiendo:

- El ocultamiento por parte del servidor es fácil, no tiene efectos en la semántica del sistema de ficheros.
- El ocultamiento por parte del cliente ofrece mejor rendimiento a costa de mayor complejidad y un semántica más confusa.

Tema 5: Sistemas de Ficheros Distribuidos.

5.2.4. Réplica.

- Razones para mantener varios copias del mismo fichero en ? servidores:

1. Aumentar la confiabilidad.
2. Ser tolerante a fallos. Disponibilidad del servicio.
3. Repartir la carga entre varios servidores. Rendimiento del sistema.

- ¿Se puede lograr la transparencia con respecto a la réplica?. Opciones:

1. El programador controla todo el proceso. En el programa se incluye las órdenes cp para que el fichero se replique en varios servidores. Los programas que utilicen el camino original prueban con cada una de las copias. Aunque funciona es poco elegante.
2. réplica retrasada. El servidor crea copias sin que lo sepa el programador como actividad secundaria.
3. uso de la comunicación en grupo. Se crean copias en todos los servidores al tiempo.

Protocolos de actualización

- No se deben hacer actualizaciones en serie. Métodos:

1. **réplica de la copia primaria**. Un servidor es el primario y el resto secundarios. Los cambios se envían al servidor primario y éste después de modificar su copia envía órdenes para actualizar el resto. Las lecturas se pueden hacer de cualquier copia.

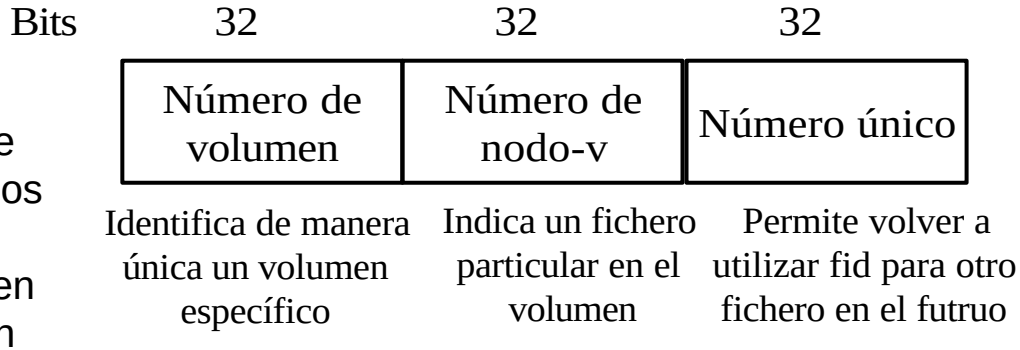
- Para protegerse frente a fallos se debe escribir la modificación en disco.
- Si falla el servidor no se pueden realizar actualizaciones.

Protocolos de actualización

2. **método del voto**: exigir que los clientes soliciten y adquieran el permiso de varios servidores antes de leer o escribir un fichero duplicado. Para modificar un fichero, el cliente debe acordarlo con Nw de los servidores, el fichero se modifica y se le asocia un nº de versión.
   - ? Para leer un fichero, el cliente debe contactar con Nr de lo servidores y les pide el nº de versión; si coinciden esta versión es la más reciente.
3. **método del voto con fantasma**: las lecturas suelen ser más frecuentes que las escrituras (Nr < Nw). Si fallan algunos servidores no se obtiene un quórum de escritura. Con este algoritmo se permiten crear servidores fantasma cuando fallen para la escritura, con la condición de que obtengan la versión actual al incorporarse al sistema antes de dar ningún servicio.

5.2.5. Un ejemplo: el sistema de ficheros Andrew.

- ? **ASF** se desarrolló en los 80 por Carnegie Mellon University. Está diseñado para manejar entre 5000-10000 estaciones de trabajo interconectadas estando una gran parte de ellas activas.



Tema 5: Sistemas de Ficheros Distribuidos.

Arquitectura de ASF

- Consta de unidades de asignación (clusters), con un servidor de ficheros y varias docenas de estaciones de trabajo clientes.
- Objetivo: lograr que la mayor parte del tráfico sea local en un cluster para reducir la carga de la columna vertebral. Pero un usuario debe de poder utilizar cualquier estación como si fuese local.
- Físicamente no hay distinción entre cliente y servidor, todas ejecutan el mismo núcleo UNIX BSD. Pero clientes y servidores ejecutan software ?.
- Cada servidor ejecuta un único programa **vice** que maneja las solicitudes de ficheros de sus clientes. Cada cliente tiene un parte de código **venus** que controla la interfaz entre cliente y *vice* y funciona como manejador caché.
- El espacio de nombres que ven los programas de usuario es un árbol tradicional UNIX que se la ha añadido */cmu*. AFS soporta el contenido de */cmu* mediante los servidores *vice* y son iguales en todas las estaciones.
- Se pretende que cada usuario interactúe lo mínimo con el resto del sistema Al abrir un fichero, se carga en el disco y se guarda en un caché de forma transparente. Para ello se crea el directorio */caché*.
- El mantenimiento del sistema es sencillo porque los discos de las estaciones sólo utilizan ficheros temporales, la caché y la paginación. El resto de la inf. está en los servidores. Sólo hay que hacer copias de seguridad de estos.

Tema 5: Sistemas de Ficheros Distribuidos.

Semántica de AFS

- AFS soporta el concepto volumen. Es una colección de directorios que se manejan juntos. Ej) los ficheros de un usuario. Un conjunto de volúmenes forman una célula que determina un punto de montaje.
- Al abrir un fichero, se busca el servidor apropiado y se coloca en /cache en el disco local de la estación. Todas las R/W operan en la copia. Cuando el fichero se cierra, se carga de nuevo en el servidor. Semántica de sesión.
- Al cerrar el fichero, se mantiene en caché. Al cargar *venus* el fichero en caché le indica a *vice* si debe preocuparse por posteriores aperturas. Si lo hace, *vice* crea una entrada con la posición del fichero. Si otro proceso abre el fichero, *vice* envía un mensaje a *venus* para que señale la entrada como no válida. Si el fichero sigue en uso, se le deja. Cuando deje de hacerlo deberá acudir a *vice* para obtener la nueva copia.
- Si una estación falla todas las entradas se consideran no válidas.
- El cierre de un fichero dura 30 min., después es liberado.

Implantación de AFS

- Principal aportación: añadir *venus* al núcleo y redireccionar las llamadas OPEN y CLOSE a *venus* para que realicen el trabajo en caché.
- *vice* y *venus* utilizan un esquema de nombres con dos niveles. Se usan fid (file identifier) en vez de i-nodos
- fid es utilizado por *vice* para localizar los volúmenes. Los volúmenes pueden emigrar entre servidores de forma completa y BD debe actualizarse. Pero esto no es muy común.
- La migración es atómica. Primero se copia en el destino y luego se elimina en el original. Los volúmenes de sólo lectura pueden duplicarse.

Tema 5: Sistemas de Ficheros Distribuidos.

Acceso a un fichero en AFS:

- Cuando un programa hace una llamada OPEN *venus* la captura y verifica si es local. Si es así la procesa de la manera usual.
  - Si no es local (su nombre comienza por /cmu), se analiza el nombre y se busca componente a componente hasta encontrar el fid. *Venus* utiliza el fid para verificar el caché, puede ocurrir:
    - El fichero está presente y es válido.
    - El fichero está presente y no es válido.
- *venus* pregunta a *vice* si el fichero ha sido modificado desde que se cargó. Si es así, se carga la nueva copia.
  - El fichero no está presente en la caché.
- Se busca mediante *vice* y se carga en la caché.
- Resultado final: el fichero está en /cache. Las llamadas READ y WRITE se manejan de manera usual. La llamada CLOSE se registran por *venus* que verifica si el fichero se modificó y entonces, lo carga en el servidor que maneja su volumen.
- *venus* tiene la cache para asociar los nombres de ruta con los fids.
- venus elimina ficheros de caché mediante el algoritmo LRU.
- vice se ejecuta como un único programa con varios hilos en cada máquina servidor. Cada hilo maneja una solicitud.
- el protocolo entre venus y vice utiliza RPC y se construye sobre IP.
- vice mantiene sus tablas en memoria virtual.

Tema 5: Sistemas de Ficheros Distribuidos.

5.2.6. Lecciones aprendidas.

- Principios generales para el diseño de sistemas de ficheros distribuidos:

1. Se debe utilizar la potencia de CPU de las estaciones de trabajo.
2. Uso de cachés. Ahorran cómputo y ancho de banda de red.
3. Hay que aprovechar las propiedades de uso. Por ej. tratar de forma especial los ficheros temporales mejora el rendimiento.
4. Minimizar el conocimiento para que el sistema sea escalable.
5. Confiar en el menor número posible de entidades.
6. Crear lotes de trabajo mientras sea posible.

Tema 5: Sistemas de Ficheros Distribuidos.

5.3. Tendencias en los Sistemas de Ficheros Distribuidos.

5.3.1. Hardware reciente.

- La memoria tiende a abaratarse:
  - ¿Podremos mantener el sistema ficheros en memoria?.
  - Los ficheros se organizan como una colección de bloques. Esto es así porque su traslado en disco es caro. En memoria esto no es tan engorroso y se podrían mantener de forma adyacente.
  - Un fallo de corriente supone perder todos los ficheros!!!. Se pueden hacer respaldos continuos o por incrementos. Es posible almacenar 5GB en una sola cinta de 8mm.
- Discos ópticos que permiten escritura.
  - Son lentos y gran capacidad de almacenamiento ? cintas, pero
  - Admiten acceso aleatorio.
  - Ahora podemos tener un respaldo en estos dispositivos y mantener el sistema de ficheros en memoria.
- Redes de fibra óptica.
  - Podemos llegar a deshacernos de las cachés del cliente y tratar directamente con la memoria del servidor.
- Llegar a construir redes que soporten señales de modificación de una palabra de memoria. De tal manera que puedan actualizar de forma automática este valor en el resto de las máquinas que tienen la misma copia del fichero.

Tema 5: Sistemas de Ficheros Distribuidos.

5.3.2. Escalabilidad.

- Los algoritmos centralizados NO soportan la escalabilidad. Es mejor separar el sistema en unidades pequeñas que actúen de manera más o menos independiente.
- Las transmisiones son otro gran problema.
- Los datos de los servidores deben mantenerse en tablas que admitan un acceso con tiempo constante.
- Las semánticas estrictas son difíciles de mantener.
- El tener un único árbol de ficheros puede hacer crecer excesivamente la longitud de las rutas de acceso.

5.3.3. Redes de área amplia.

- Los sistemas distribuidos actuales están basados en LAN.
- Las redes de área amplia son heterogéneas!!!.
- Los usuarios demanda otros tipos de aplicaciones de fácil uso y con interés más lúdicos e informativos.
- El ancho de banda es demasiado bajo. La fibra óptica tardará en ser algo general (es carísimo). Puede ser más barato mandarle la BD entera en un CD-ROM o en una cinta.

5.3.4. Usuarios móviles.

- El uso de computadoras portátiles es cada vez más generalizado. Pero su capacidad de conexión es baja e inexistente dependiendo del lugar donde se utilice.

5.3.5. Tolerancia a fallos.

- Los sistemas actuales no son muy tolerantes a fallos. Si se generaliza el uso de sistemas distribuidos a la gente no le gustará.
- Los sistemas necesitarán de una gran redundancia hardware y de comunicaciones, así como de software y a datos. El duplicar ficheros será obligado.
- Se tendrán que diseñar sistemas que puedan trabajar con parte de los datos.