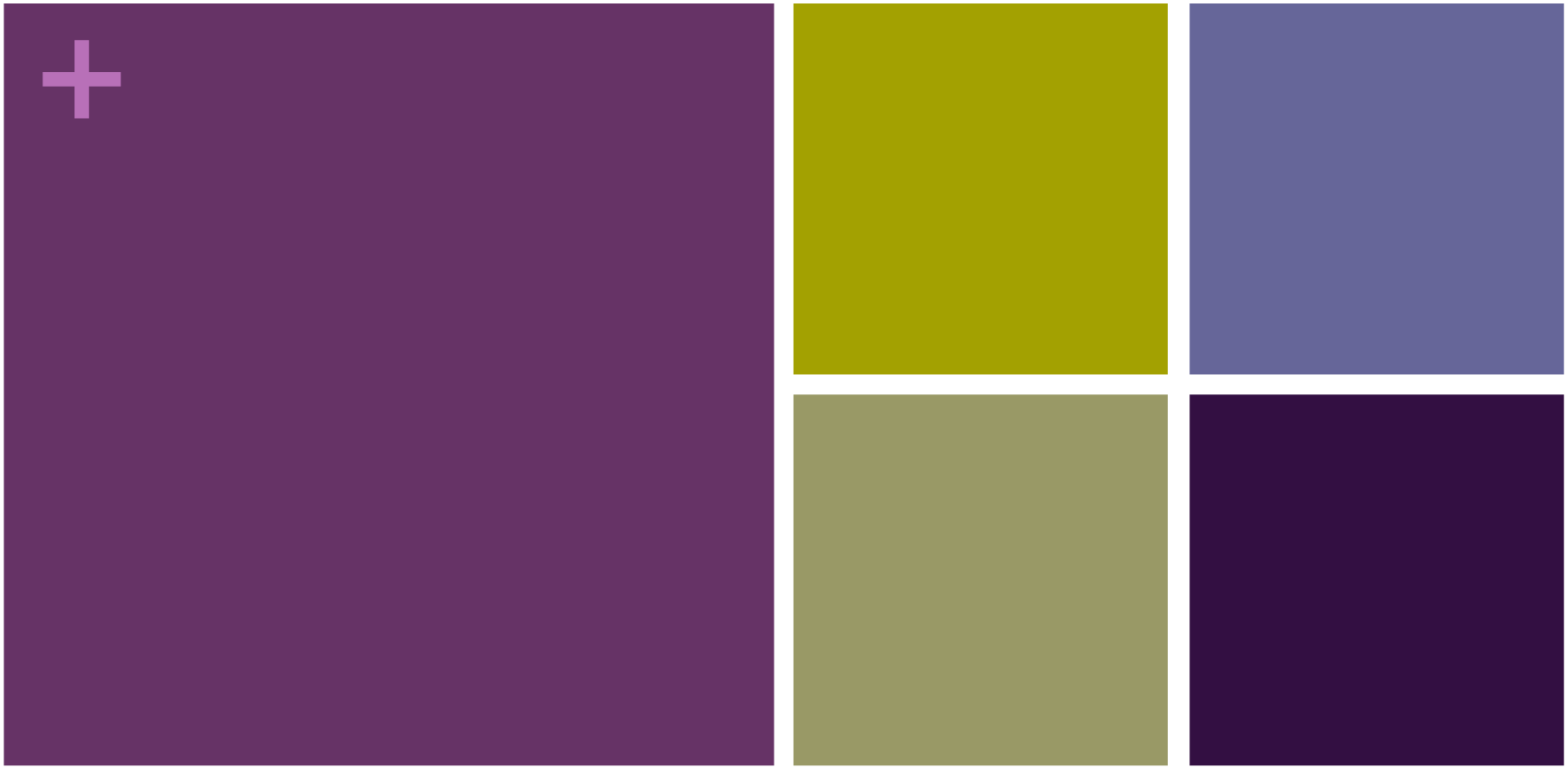


Sistemas Distribuidos
Sistemas de Archivos

Rodrigo Santamaría

Sistemas de archivos

Introducción

- Tipos de almacenamiento
- Requisitos
- Arquitectura

Sun Network File System (NFS)

Andrew Network File System (AFS)

Introducción

- Un sistema de archivos distribuido (SAD) permite almacenar y acceder archivos remotos como si fueran locales
  - Sin pérdida relevante de rendimiento y fiabilidad

- Veremos dos implementaciones que han sido las más utilizadas durante las últimas dos décadas (**NFS** y **AFS**)

- Hay más sistemas de almacenamiento distribuido, como los utilizados por P2P o Google, que se cubren en otros temas

Introducción

Tipos de almacenamiento

Un sistema de archivos distribuido es el único tipo de almacenamiento que provee reparto de archivos garantizando un nivel de consistencia aceptable

	Sharing	*Persistence*	*Distributed cache/replicas*	*Consistency maintenance*	*Example*
Main memory	✗	✗	✗	1	RAM
File system	✗	✓	✗	1	UNIX file system
Distributed file system	✓	✓	✓	✓	Sun NFS
Web	✓	✓	✓	✗	Web server
Peer-to-peer storage system	✓	✓	✓	2	OceanStore

Coulouris et al. 2011

Consistencia:
(1) una sola copia
(1) consistencia un poco más débil
(2) consistencia considerablemente más débil

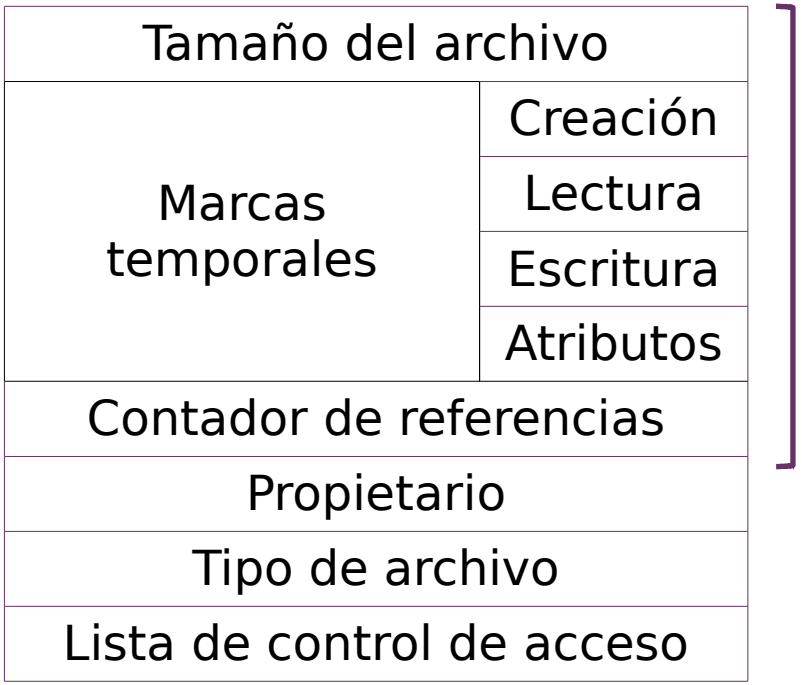
Introducción

Responsabilidades

- Un SAD es responsable de las operaciones sobre archivos:
  - Organización
  - Almacenamiento
  - Recuperación
  - Nombrado
  - Reparto
  - Protección
- Un archivo contiene
  - **Datos**: items (bytes) accesibles mediante lectura y escritura
  - **Atributos**: registro indicando información sobre el archivo

Introducción

Registro de atributos de un archivo



Gestionados por el sistema de archivos
(no actualizables por el usuario)

Introducción

Módulos

	Módulo	Tarea
+D	**Localización**	Resuelve y localiza los nombres de archivos distribuidos
+D	**Comunicación**	Establece la comunicación entre cliente y servidor
SA	Directorio	Relaciona nombres de archivos con ID de archivos
SA	Archivos	Relaciona ID de archivos con archivos concretos
SA	Control de acceso	Comprueba los permisos para una operación solicitada
SA	Acceso a archivos	Lee o escribe datos o atributos de un archivo
SA	Bloques	Accede y asigna bloques de disco
SA	Dispositivo	E/S de disco y búferes

¿Te suena similar a la arquitectura utilizada en el middleware?

Introducción

Interfaz y operaciones

Interfaz	Sistema de Archivos Distribuido	UNIX
`abrir` **`cerrar`**	No (no son idempotentes)	Sí (retornan el ***puntero de r/w***)
`leer` **`escribir`**	Debe especificarse la posición dentro del archivo	Modifican automáticamente la posición del puntero de r/w
`seguir`	No (no tenemos el puntero de r/w)	Permite reposicionar el puntero de r/w
Cualidad	**Sistema de Archivos Distribuido**	**UNIX**
repetición	Semántica “al menos uno”: todas las operaciones son ***idempotentes*** salvo `crear`	Las operaciones ***no*** son idempotentes -> `leer/escribir` **actualizan** el puntero de posición
estado	***sin estado***, el servidor puede reiniciarse tras un fallo sin necesidad de restaurar el estado en el cliente	***con estado***, un reinicio implica la pérdida del puntero de r/w y por tanto restauración en el cliente
control de acceso	La identidad del usuario se envía al servidor con *cada* operación (*sin estado*) Canal no seguro -> seguridad comprometida	A través del `uid` y la lista de permisos

Introducción

Requisitos

- **Transparencia**
  - El cliente debe ver y acceder a un SAD de manera uniforme independientemente de la ubicación de los archivos, de si estos cambian de ubicación o incrementa la carga o escala del servicio
- **Heterogeneidad**
  - Respecto al SSOO y HW de cliente y servidor
- **Concurrencia**
- **Eficiencia**
  - Rendimiento parecido al de un sistema de archivos local
- **Seguridad**:
  - Acceso a archivos por listas de permisos -> autenticación de clientes

Introducción

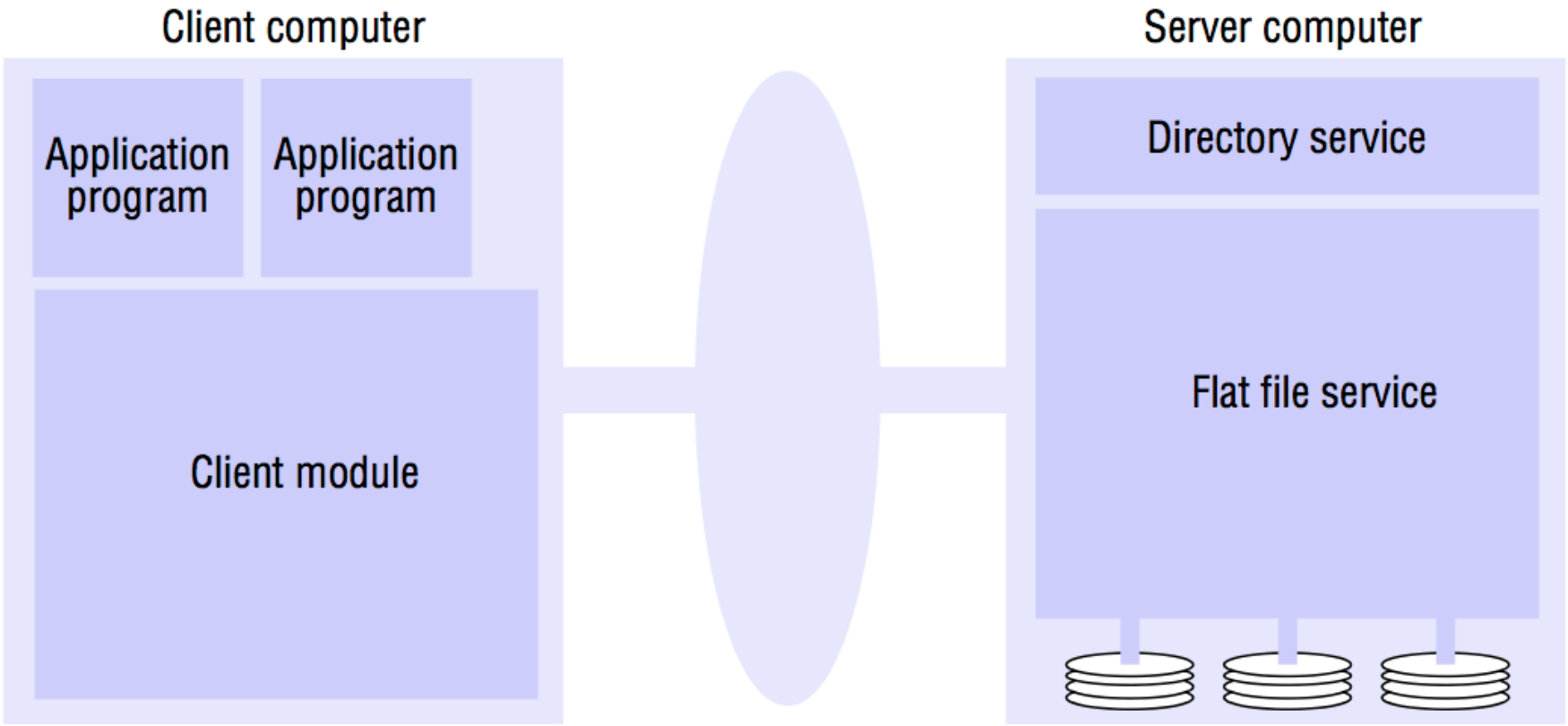
Requisitos (II)

- **Tolerancia a fallos**
  - Al tener un rol central en el SD, este requisito es crítico
  - Fallos de comunicación: fáciles de tolerar, con semánticas *como-mucho-uno,* o *al-menos-uno* y operaciones idempotentes
  - Fallos de proceso: solución más complicada, mediante replicación
- **Replicación**
  - Incrementa la disponibilidad y la tolerancia a fallos, pero debe mantener un nivel de consistencia aceptable*
- **Consistencia**
  - Consistencia de una copia: todos los procesos ven el mismo archivo
  - En caso de replicación, los cambios tardan en propagarse y la consistencia puede ser menor

*Se tratará ampliamente en el tema correspondiente

Introducción

Arquitectura



Coulouris et al. 2011

Introducción

Arquitectura: componentes

- **Servicio de archivos plano**: usa UFIDs (Unique File IDentifiers) para identificar archivos de manera única en el sistema distribuido.
- **Servicio de directorios**: mapea UFIDs a rutas textuales. Cliente del servicio de archivos plano.
- **Módulo cliente**: activo en cada ordenador que acceda al SAD. Emula operaciones locales de acceso a archivos y conoce la localización de los servicios. Puede tener cachés.
- **Interfaz con el servicio de archivos plano**: especificación RPC utilizada por el módulo cliente, no por las aplicaciones.

Sistemas de archivos

- Introducción
- Sun Network File System (NFS)
- Andrew Network File System (AFS)

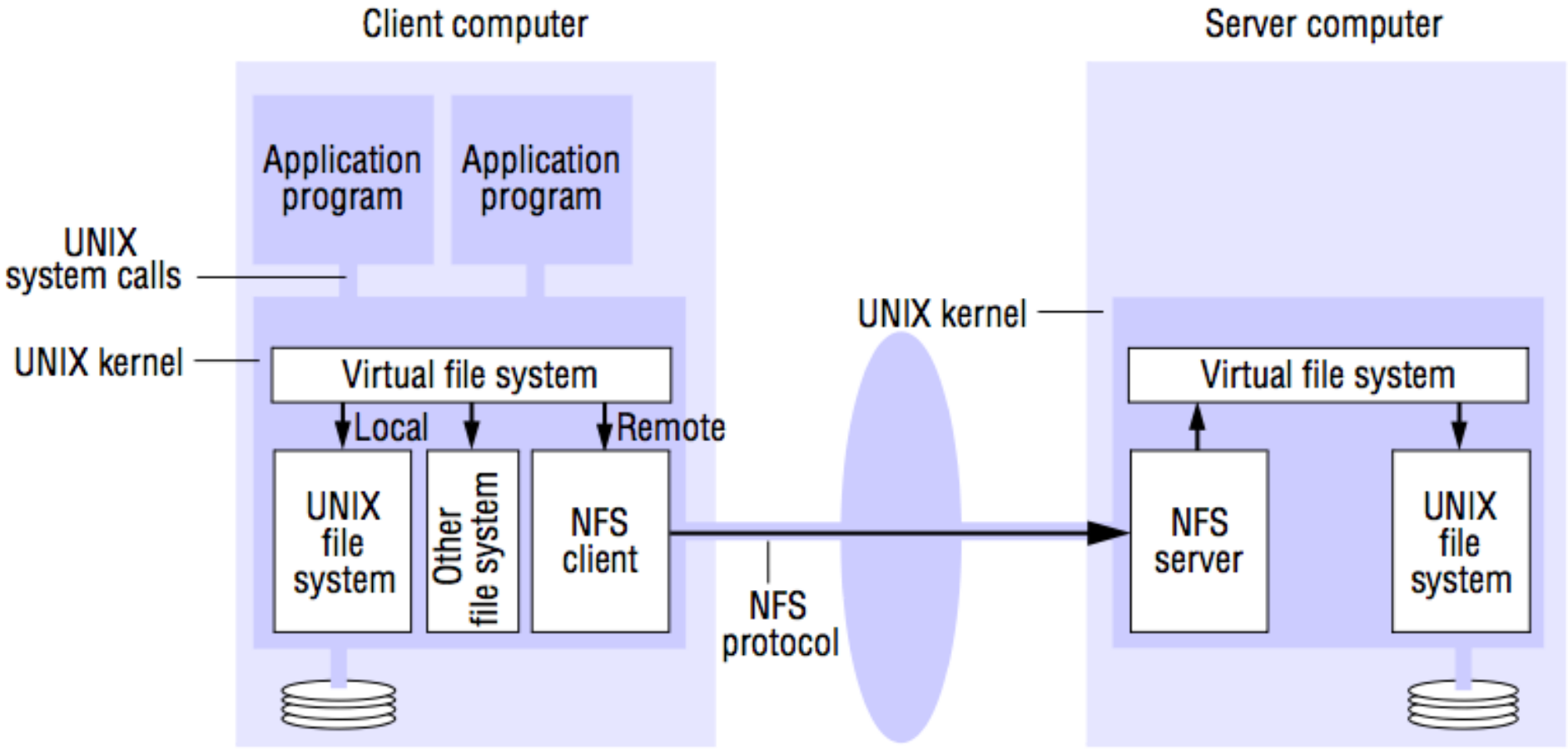
NFS

Sun Network File System

- Protocolo: independiente del sistema operativo
  - Originalmente diseñado para redes con sistemas UNIX
  - Define las operaciones sobre archivos que se pueden realizar vía RPC
- Servidor y cliente tienen módulos específicos que se comunican mediante RPC sobre TCP o UDP
  - La interfaz del servidor es abierta, cualquier cliente que conozca su ubicación puede hacer peticiones
  - La petición debe ser correcta y tener permisos para ejecutarse

NFS

Arquitectura



Coulouris et al. 2011

NFS

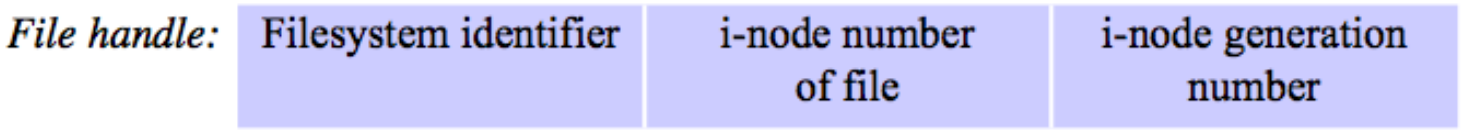
Sistema de archivos virtual (VFS)

- Módulo incrustado en el kernel de UNIX
  - Utiliza **v-nodos** para identificar los archivos
    - Si es local, contiene la referencia al archivo local (i-nodo en UNIX)
    - Si es remoto, contiene el file handle
  - Dirige las peticiones al servicio de archivos correspondiente
    - El sistema de archivos UNIX local
    - El módulo cliente de NFS
    - Otro sistema de archivos implementado
  - Traduce los identificadores de archivo del sistema local a los identificadores de NFS (*file handles*)

NFS

File Handle

- Identificador de archivo en NFS
- Deriva del número de i-nodo de UNIX, añadiendo 2 campos:
  - *Identificador del sistema de archivo*: identifica el sistema de archivos en el que se almacena el archivo
  - *Número de generación del i-nodo*: se incrementa cada vez que el i-nodo es reutilizado, para evitar identificadores iguales
    - En UNIX, un i-nodo se puede reutilizar tras borrar su archivo
    - Si esto ocurre mientras un cliente remoto está haciendo uso de dicho i-nodo, tenemos un problema -> uso del nº de generación



Coulouris et al. 2011

NFS

Cliente NFS

- Emula las primitivas del sistema de archivos de UNIX
  - Integrado directamente en el kernel
- Controla las peticiones del VFS al servidor NFS
  - Transfiere los bloques o archivos hasta/desde el servidor
  - Almacena en caché local los bloques cuando es posible

NFS

Servidor NFS

- Interfaz definida en la RFC 1813 [Callaghan et al. 1995]
  - Peticiones similares a las del modelo de archivos plano (diap 14.)
    - *read, write, getattr, setattr*
  - Peticiones adicionales que emulan las del sistema UNIX
    - *create, remove, mkdir, rmdir, rename, link, symlink*
  - *statfs*: da información del estado de sistemas de archivos remotos
  - *readdir*: lectura del contenido de directorios independiente de su modo de representación
- Además, ofrece servicios de
  - Montaje
  - Control de accesos y autentificación
  - Caché

NFS

Servicio de montaje

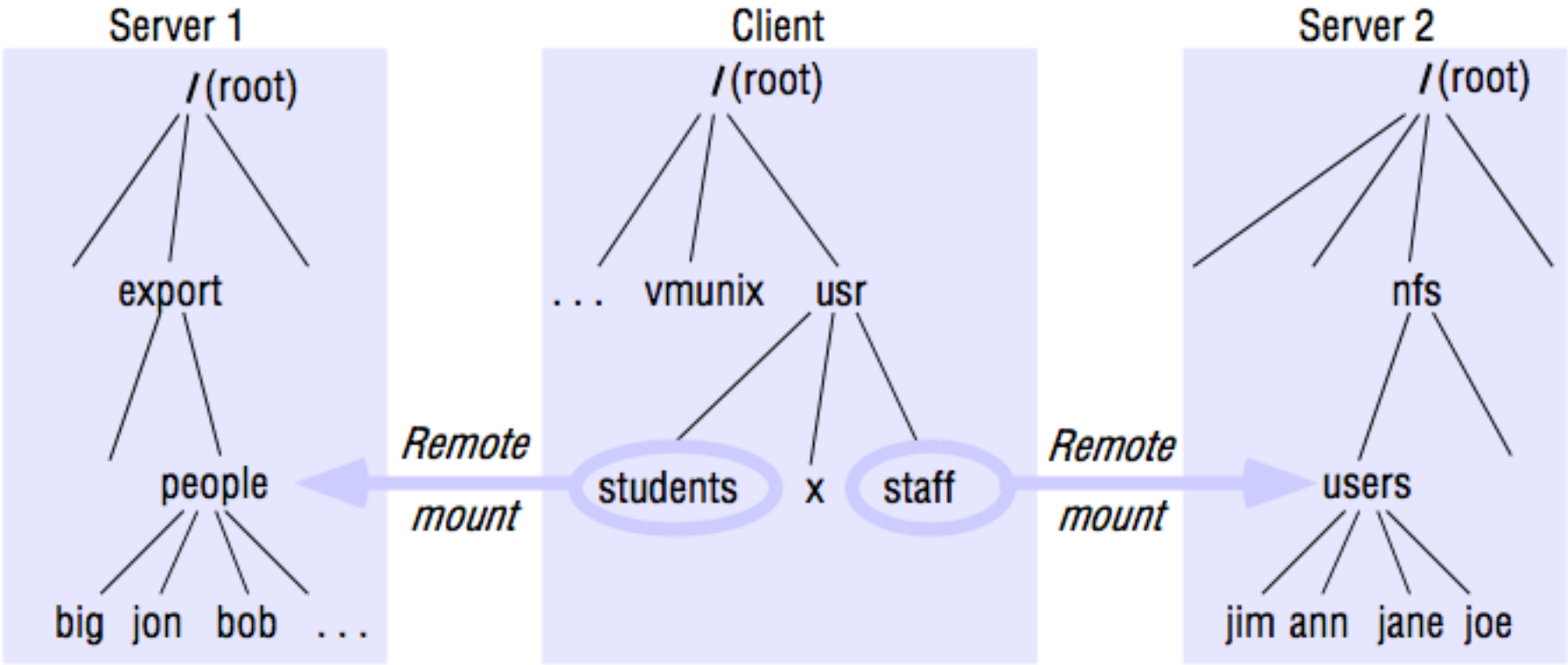
- Archivo `/etc/exports` en el servidor NFS
  - Contiene los nombres de los sistemas de archivos locales disponibles para montaje remoto:

```
/usr/local 192.168.0.1(ro) 192.168.0.2(ro)
/home      192.168.0.1(rw) 192.168.0.2(rw)
```

- Los clientes usan una versión de la orden `mount` para hacer peticiones de montaje de un sistema de archivos remoto
  - Puede ser cualquier subárbol del sistema de archivos remoto
  - Protocolo RPC de montaje remoto
    - argumentos:
      - IP y puerto del servidor NFS
      - file handle del directorio remoto

NFS

Servicio de montaje



El sistema de archivos montado en `/usr/students` es realmente el subárbol ubicado en `/export/people` del servidor 1
El sistema de archivos montado en `/usr/staff` es realmente el subárbol ubicado en `/nfs/users` del servidor 2

NFS

Traducción de pathnames

- No se puede hacer en el servidor, porque las rutas pueden cruzar ‘puntos de montaje’ en el cliente
  - Ejemplo: si el cliente anterior quiere acceder a `/export/people/bob` en el servidor 1 no puede preguntar por `/usr/students/bob`, aunque esa es su ruta de acceso local
- Se realizará de manera iterativa en el cliente NFS
  - Cada parte de la ruta que se refiere a un directorio remoto se traduce a un file handle mediante llamadas ***lookup*** al servidor
  - Los resultados se almacenan en caché, pues es un proceso relativamente ineficiente

NFS

Caché en UNIX

- Lectura:
  - **buffer caché**: mantiene archivos leídos recientemente en memoria principal
  - **read-ahead**: anticipa accesos de lectura a las posiciones de memoria siguientes a las que se están leyendo
- Escritura:
  - **delayed-write**: no se realiza la escritura de modificaciones hasta que el archivo no es requerido por otro proceso
  - **sync**: escritura cada 30s para evitar pérdidas en caso de parada del sistema
- Las opciones de caché en lectura son aplicables a NFS
- Las opciones de caché en escritura necesitan modificaciones para garantizar la consistencia

NFS

Caché en Servidor

- Dos opciones para asegurar la consistencia en escritura
  - **write-through**: los datos de las operaciones de escritura se guardan en caché *y* se escriben en disco antes de responder al cliente
  - **commit**: los datos de las operaciones de escritura se guardan sólo en caché. Sólo se escriben a disco cuando se recibe una operación *commit* (generalmente, cuando se cierra un archivo abierto para escritura)
    - Soluciona problemas de rendimiento por cuello de botella al usar *write-through* en servidores que reciben muchas peticiones de escritura

NFS

Caché en el Cliente (I)

- El módulo cliente de NFS guarda en caché los resultados de las operaciones *read*, *write*, *getattr*, *lookup* y *readdir*
- Puede haber varias versiones de archivos o porciones de archivos en distintos nodos cliente, debido a las escrituras
- Los clientes son responsables de sondear al servidor para comprobar la actualidad de sus datos de caché

NFS

Caché en el Cliente (II)

- Método de marcas temporales para mantener las cachés
  - Cada elemento de datos es etiquetado con dos marcas
    - T_c tiempo en el que se validó el elemento por última vez
    - T_m tiempo de la última modificación del elemento en el servidor
  - Una entrada en caché es válida en un momento T si
    - $(T - T_c < t)$ v $(T_{mcliente} = T_{mservidor})$
    - Siendo t el intervalo de refresco tolerado
      - Compromiso entre consistencia y eficiencia.

NFS

Caché en el Cliente (III)

- Validación de la entrada en caché para el archivo *F*

si ($T - T_c < t$)

La entrada es válida

(No requiere acceso al servidor)

si no

Obtener $T_{mservidor}$ mediante *getattr*(F, $T_{mservidor}$)

si ($T_{mcliente} = T_{mservidor}$)

La entrada es válida,

actualizar T_c

si no

La entrada es inválida

solicitar nueva copia al servidor

NFS

Caché en el Cliente (IV)

- Minimización de llamadas a *getattr*
  - Cuando se recibe un valor de $T_{mservidor}$ de un archivo, se aplica a todas las entradas relevantes de dicho archivo
  - *piggyback*: los metadatos del archivo (p. ej. $T_{mservidor}$) se adjuntan al resultado de las operaciones sobre él
  - algoritmo adaptativo para optimizar el valor de *t*
    - P. ej. en Solaris entre 3 y 30s (entre 30 y 60s para directorios)

NFS

Caché en el Cliente (V)

- Todavía habrá problemas de consistencias si tenemos escrituras en dos clientes con una diferencia de tiempo $< t$
- Si queremos una consistencia de grado más fina, tenemos que usar **bloqueo de archivos**
  - Convertimos al archivo en una “sección crítica”
  - Cuando se obtiene el acceso exclusivo al archivo o a una de sus partes, se bloquea el acceso de terceros
    - Tendrán que esperar a que se libere
  - En NFS, este sistema se consigue mediante el protocolo NLM (Network Lock Manager)

NFS

Cachés

	UNIX	Servidor NFS	Cliente NFS
lectura	buffer caché	buffer-caché	cachés (marcas temporales) (*getattr* y *piggyback*) (network lock)
	read-ahead	read-ahead	
escritura	delayed-write	write-through	
	sync	commit	

NFS

Seguridad: Kerberos

- NFS incluye la identidad del usuario en las peticiones al servidor
  - Pero sólo para comparar con los permisos de acceso
  - No valida la identidad del usuario per se
  - No se encripta la identidad para la comunicación
- Kerberización
  - Se realiza autenticación de usuario en el momento del montaje
  - Los resultados de la autenticación se almacenan y se utilizan en cada petición NFS
  - Tráfico adicional mínimo y protección contra la mayoría de los ataques (suponiendo sólo un login por ordenador)

NFS

Rendimiento

- Similar a un sistema de archivos local [Sandberg, 1987]
- Dos cuestiones problemáticas
  - el abuso de *getattr* para comprobar las marcas temporales de los servidores para la validación de las cachés
  - rendimiento bajo de *write* debido a la estrategia write-through del servidor (aún peor con la estrategia de bloqueo de archivos)
- Frecuencia de operaciones*
  - **write**: menos del 5% de las llamadas (penalización de *write-through* poco relevante salvo al escribir ficheros muy grandes)
  - **lookup**: alrededor del 50% (para traducción de rutas)

*Este tipo de aproximaciones para optimizar o diseñar un sistema teniendo en cuenta su realidad práctica es muy importante en ciertas implementaciones, desde Bitcoin hasta los servicios de Google

Sistemas de archivos

- Introducción
- Sun Network File System (NFS)
- Andrew Network File System (AFS)

AFS

- Andrew File System (Carnegie Mellon University)
  - Por Andrew Carnegie y Andrew Mellon
- Sistema de archivos distribuidos para UNIX
  - Similar a NFS en objetivos, pero no en diseño/implementación
  - Centrado en la **escalabilidad** y la seguridad
- Características principales
  - **Servicio de archivo completo**: todo el contenido de archivos o directorios se transmiten al cliente (troceado si es necesario)
  - **Caché de archivo completo**: la caché contiene cientos de archivos y sobrevive a reinicios del cliente

AFS

Observaciones generales

- La mayoría de archivos se actualizan poco (p.ej. bibliotecas de funciones) o sólo los actualiza un usuario (p.ej. ficheros en `/home/usuario`)
  - En ambos casos, las copias de caché son válidas mucho tiempo
- La caché local es suficientemente grande para almacenar un número grande de archivos (p. ej. 100MB)
  - Suficiente para el conjunto de archivos con los que un usuario normal suele trabajar

AFS

Observaciones generales (II)

- Estrategias de diseño basadas en el uso normal de archivos
  - Archivos pequeños (la mayoría <10KB)
  - La lectura es más frecuente que la escritura
  - El acceso secuencial es mucho más frecuente que el aleatorio
  - La mayoría de archivos son accedidos por un único usuario
  - Si se ha accedido a un fichero, es probable que se vuelva a acceder a él en un futuro próximo
- Las bases de datos son el único tipo de archivo que no cumple estas predicciones respecto al uso

¿Y los sistemas tipo wiki o red social, cumplen estas predicciones?

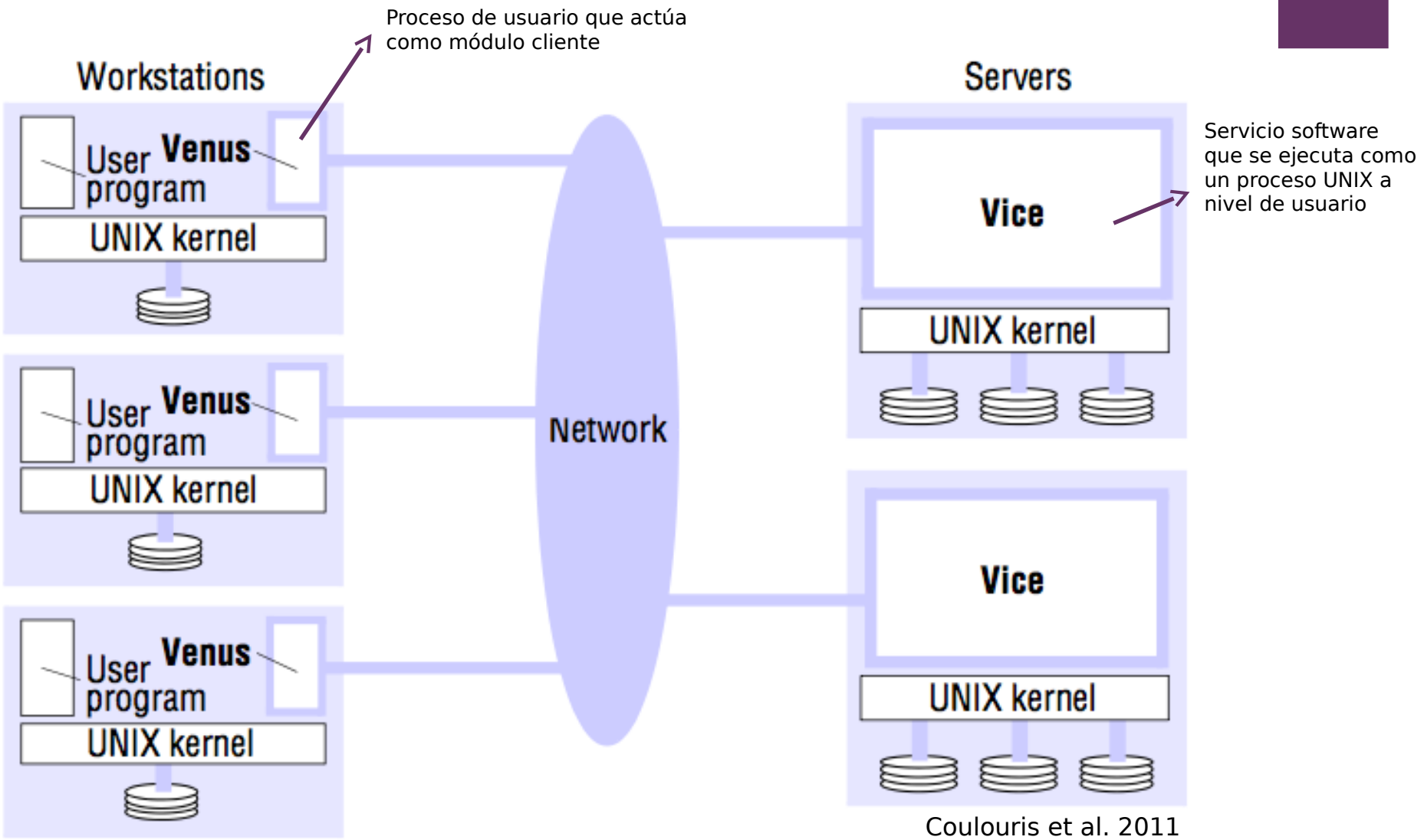
AFS

Escenario

- Un proceso cliente ejecuta **open** sobre un archivo remoto
  - del que no tiene copia en caché
- Se localiza el servidor del archivo y se obtiene una copia
  - se almacena en caché y se realiza open de UNIX
- Operaciones **read/write** sobre la copia local
- Cuando el cliente ejecuta **close**, si la copia ha sido actualizada se envía al servidor
  - el servidor actualiza el contenido del archivo y sus marcas temporales

AFS

Arquitectura

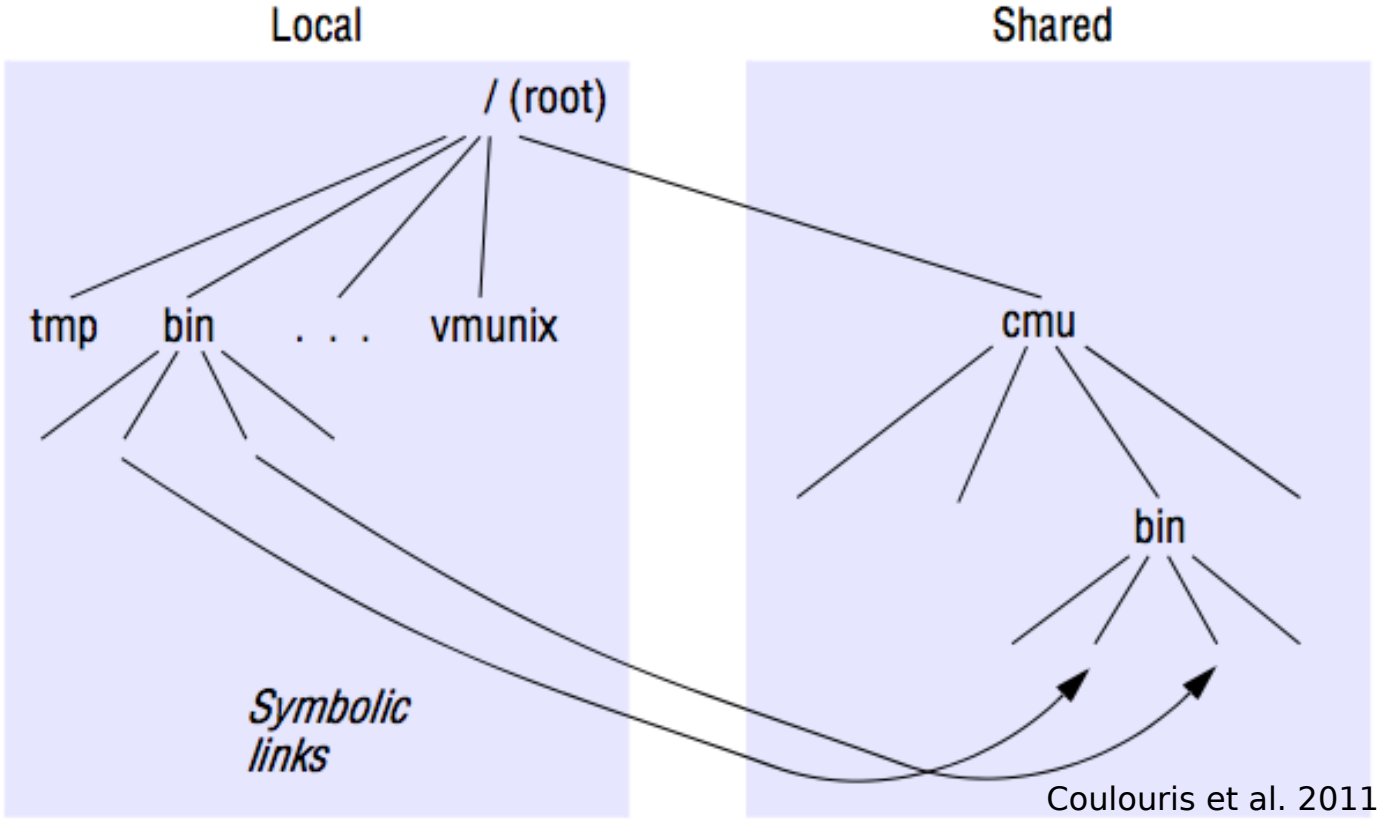


Coulouris et al. 2011

AFS

Tipos de archivo y acceso

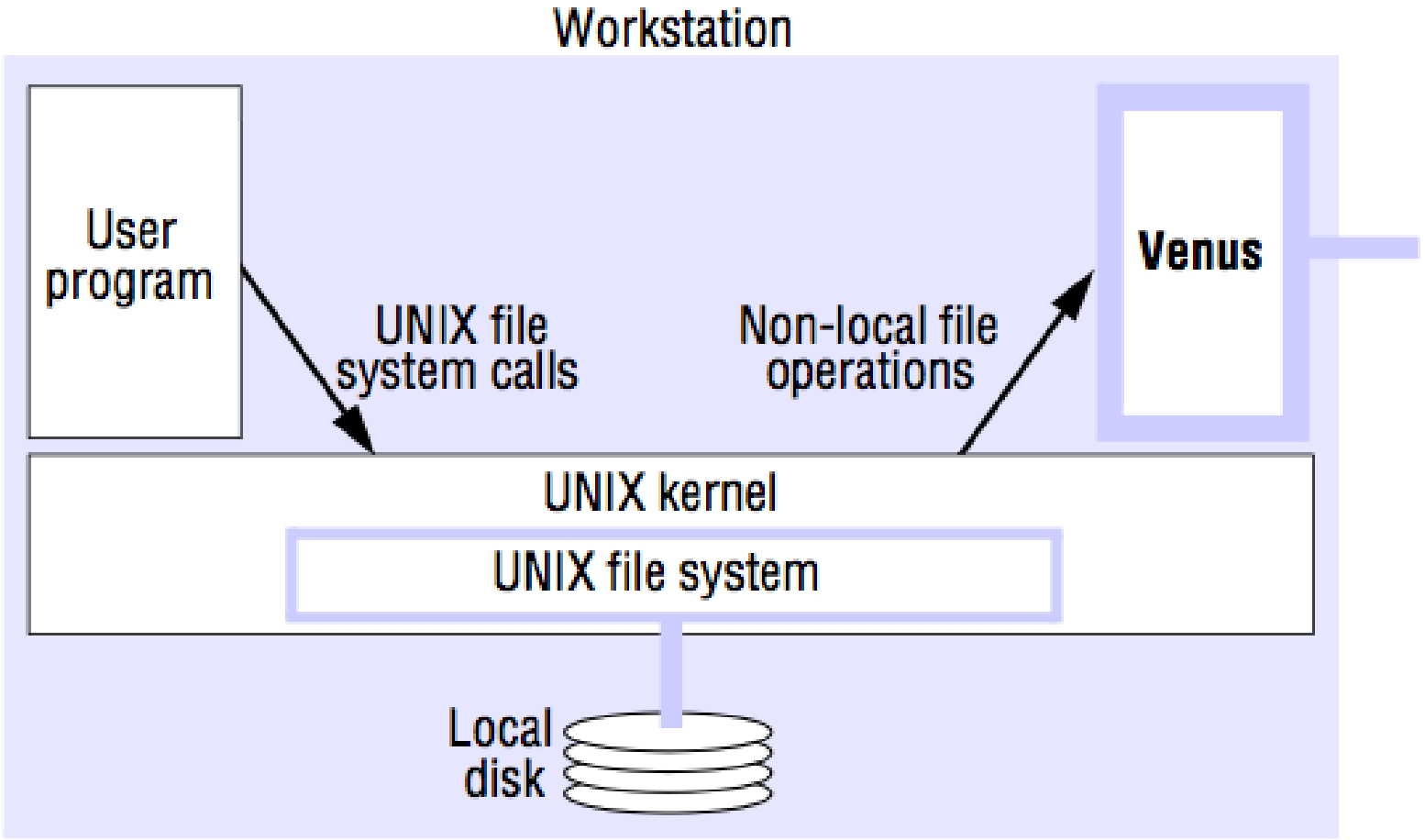
- Archivos locales o compartidos (en el servidor + copia local)
  - Los archivos compartidos se encuentran bajo el directorio /cmu
    - Si debieran estar en otro lugar (p. ej./bin) se usan enlaces simbólicos



Coulouris et al. 2011

AFS

Sistema de llamadas



Coulouris et al. 2011

User process	*UNIX kernel*	*Venus*	*Net*	*Vice*
open(FileName, mode)	If *FileName* refers to a file in shared file space, pass the request to Venus.	Check list of files in local cache. If not present or there is no valid *callback promise*, send a request for the file to the Vice server that is custodian of the volume containing the file.	→	Transfer a copy of the file and a *callback promise* to the workstation. Log the callback promise.
	Open the local file and return the file descriptor to the application.	Place the copy of the file in the local file system, enter its local name in the local cache list and return the local name to UNIX.	←	
read(FileDescriptor, Buffer, length)	Perform a normal UNIX read operation on the local copy.			
write(FileDescriptor, Buffer, length)	Perform a normal UNIX write operation on the local copy.			
close(FileDescriptor)	Close the local copy and notify Venus that the file has been closed.	If the local copy has been changed, send a copy to the Vice server that is the custodian of the file.	→	Replace the file contents and send a *callback* to all other clients holding *callback promises* on the file.

Coulouris et al. 2011

AFS

Callbacks y consistencia de caché

- Cada vez que Vice provee con un archivo a Venus, le adjunta una variable (callback promise)
  - Le garantiza que le avisará si otro proceso modifica el archivo
  - Esta promesa puede tener el valor:
    - '*válida*': cuando Vice sirve el archivo a Venus
    - '*cancelada*': cuando Venus recibe una llamada *callback* de Vice
- Si el cliente falla, mantiene su caché y al reiniciar
  - Chequea con Vice si cada una de sus promesas sigue siendo válida o se deben cancelar
- Este diseño obliga al servidor a mantener información sobre el estado de los clientes

AFS

AFS vs NFS

	NFS	AFS
Diseño	Centrado en la consistencia	Centrado en el rendimiento
Estado del servidor	sin estado, *no debe mantener información de los clientes*	conoce información de los clientes (*callback promises*)
Consistencia	Se consigue con operaciones *write-through* (o se chequea con *commit*)	Se chequea en las operaciones *open* y *close* -> si dos procesos acceden concurrentemente a un archivo, sólo persistirán las modificaciones del último que haga *close*
Escala	Penalización de las operaciones *write-through* si hay ficheros muy grandes o muchos usuarios	Muchas operaciones *write* o muchos usuarios no penalizan al sistema de cachés locales

Optimismo y pesimismo

Un último apunte

- AFS utiliza una estrategia optimista (OCC, Optimistic Concurrency Control) para la concurrencia
  - *"La mayoría de las transacciones se completan sin interferencias"*
- NFS utiliza una estrategia semi-pesimista (*write through*), siendo la versión más pesimista la que utiliza *bloqueo de ficheros* (NLM)
- Optimismo = mejor rendimiento/problemas si hay conflicto
- Pesimismo = mucho peor rendimiento/sin problemas de conflicto

Concurrencia optimista

Fases

- **Comienzo:** se registra del momento de comienzo de la operación
- **Modificación:** lectura normal, escritura *tentativa*
- **Validación:** comprobar si otra operación ha modificado el archivo
- **Commit/Rollback (consumación/marcha atrás)**:
  - Si no hay conflicto, se formaliza la escritura
  - Si hay conflicto, se aborta la escritura o se aplica algún otro protocolo de resolución de conflictos.

Operación	AFS
Comienzo	`open`
Modificación	`write + callback promise`
Validación	`callback`
Commit/Rollback	`close`

Veremos estrategias similares en la ***replicación***

Contenido distribuido en la web

HTTP y OCC

- HTTP es *sin estado:* no puede haber bloqueo de archivos
  - Ejemplo: tenemos una entrada de una wiki que un usuario bloquea para editar, pero luego abandona sin cancelar o salir de su sesión -> el servidor no tiene manera de saber que el bloqueo ha terminado, salvo algún sistema de timeout.
- HTTP permite OCC de manera nativa
  - El método GET permite el atributo opcional de cabecera *Etag*
    - Normalmente algún tipo de hash, aunque HTTP no lo especifica
  - El método PUT puede usar el *Etag* en su cabecera *If-Match* para comprobar si el archivo ha sido modificado
- MediaWiki* o Bugzilla implementan estrategias OCC sobre HTTP

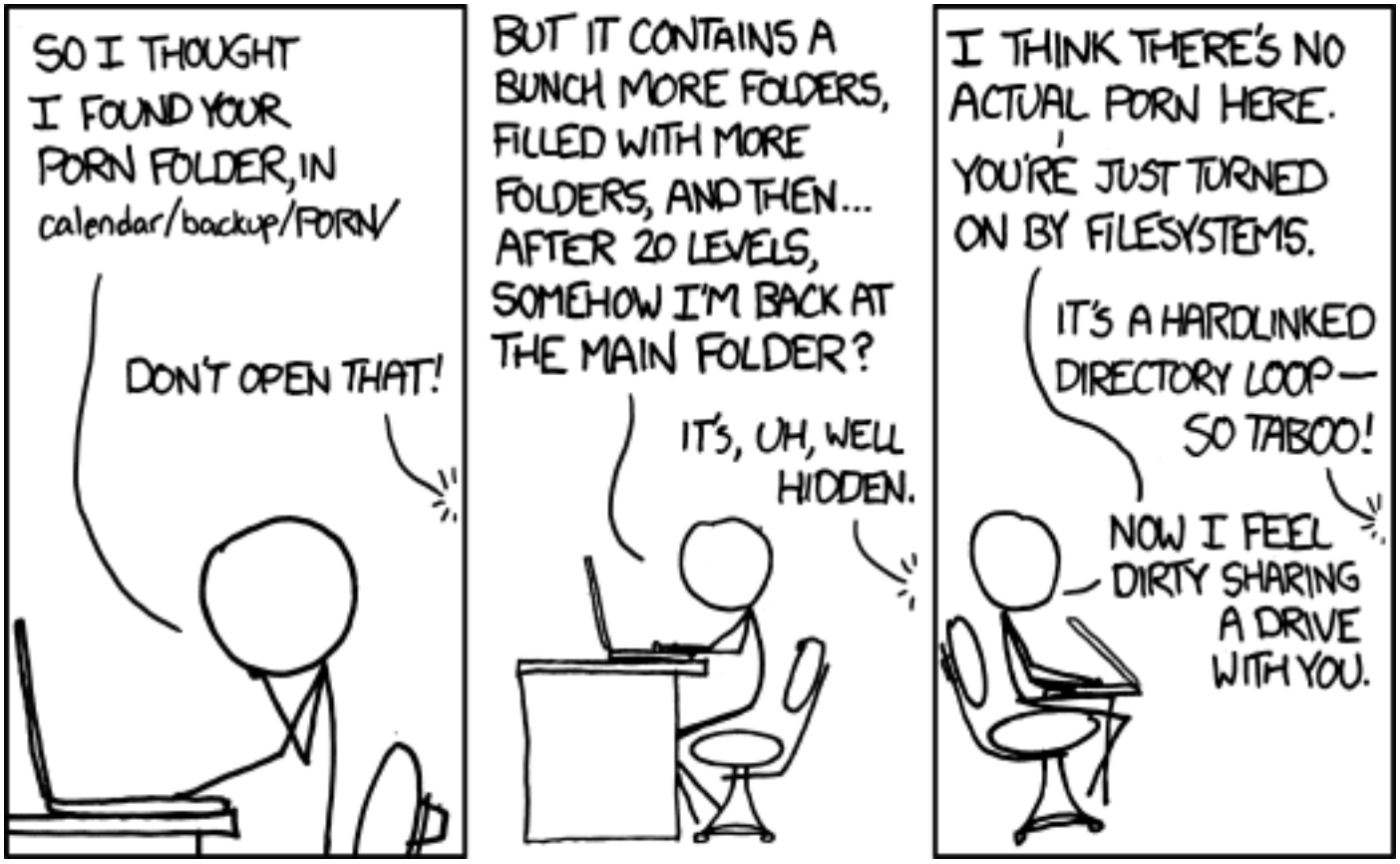
* Aplicación de wiki utilizada como infraestructura por Wikipedia

Resumen

- Un sistema de archivos distribuido busca **compartir** ficheros locales sin una pérdida relevante de **persistencia** o **consistencia** en dichos archivos
- Responden a una arquitectura **cliente-servidor** en la que el cliente debe redirigir las llamadas al sistema relacionadas con archivos remotos, y el servidor debe facilitar y mantener sus archivos compartidos
- Hay dos sistemas de archivos distribuidos principales, ambos emulando un sistema de archivos UNIX: NFS y AFS
- **NFS** se centra en la consistencia, asumiendo operaciones *write* pesadas que probablemente no sean muy frecuentes
- **AFS** se centra en la escalabilidad, manteniendo cachés locales grandes en el cliente y minimizando las actualizaciones con el servidor, al coste de una menor consistencia
- Se puede hablar de estrategias distintas según el **optimismo** con el que tratemos la posibilidad de encontrar **conflictos**

Referencias

- G. Colouris, J. Dollimore, T. Kindberg and G. Blair. *Distributed Systems: Concepts and Design (5th Ed)*. Addison-Wesley, 2011
  - Capítulo 12
- Hal Stern, Mike Eisler and Ricardo Labiaga. *Managing NFS and NIS (2nd Ed)*. O'Rilley, 2001
  - File locking: http://docstore.mik.ua/orelly/networking_2ndEd/nfs/ch11_01.htm
- Concurrencia optimista:
  - *Kung, H.T. (1981). "On Optimistic Methods for Concurrency Control". ACM Transactions on Database Systems*



http://xkcd.com/981/