Computação Distribuída – Cap. V

Licenciatura em Engenharia Informática

Universidade Lusófona

Prof. José Rogado
Prof. José Faísca

Sistemas de Ficheiros Distribuídos

- Conceito e caracterização
- Arquitecturas de SGF distribuídos
- Implementações: NFS e AFS
- Problemática do *caching*: performance e consistência

Partilha de Recursos de Armazenamento

- Uma das principais funcionalidades dos Sistemas Distribuídos é o acesso a informação remota
  - A informação constitui valor essencial
    - *Information-Centered Society*
  - A quantidade de informação disponível é cada vez maior
  - É impraticável existirem réplicas de toda informação necessária nos locais onde é necessária
- A partilha de recursos de armazenamento é pois fundamental para permitir o seu acesso generalizado
- Existem vários níveis de partilha de informação
  - Acesso em modo consulta em larga escala (Web, Peering, …)
  - Acessos escrita/leitura em larga escala (AFS, CODA, …)
  - Armazenamento de objectos persistentes (RMI, CORBA)
  - Acesso escrita/leitura redes locais e Intranet (NFS, Samba, AFS)

Propriedades de Sistemas de Armazenamento Remotos

	Sharing	*Persis-tence*	*Distributed cache/replicas*	*Consistency maintenance*	*Example*
Main memory	×	×	×	1	RAM
File system	×	✓	×	1	UNIX file system
Distributed file system	✓	✓	✓	2	Sun NFS
Web	✓	✓	✓	×	Web server
Distributed shared memory	✓	×	✓	2	Ivy (DSM, Ch. 18)
Remote objects (RMI/ORB)	✓	×	×	1	CORBA
Persistent object store	✓	✓	×	1	CORBA Persistent Object Service
Peer-to-peer storage system	✓	✓	✓	3	OceanStore (Ch. 10)

Types of consistency:
1: strict one-copy. 2: slightly weaker guarantees. 3: considerably weaker guarantees.

Sistemas de Gestão de Ficheiros (SGF)

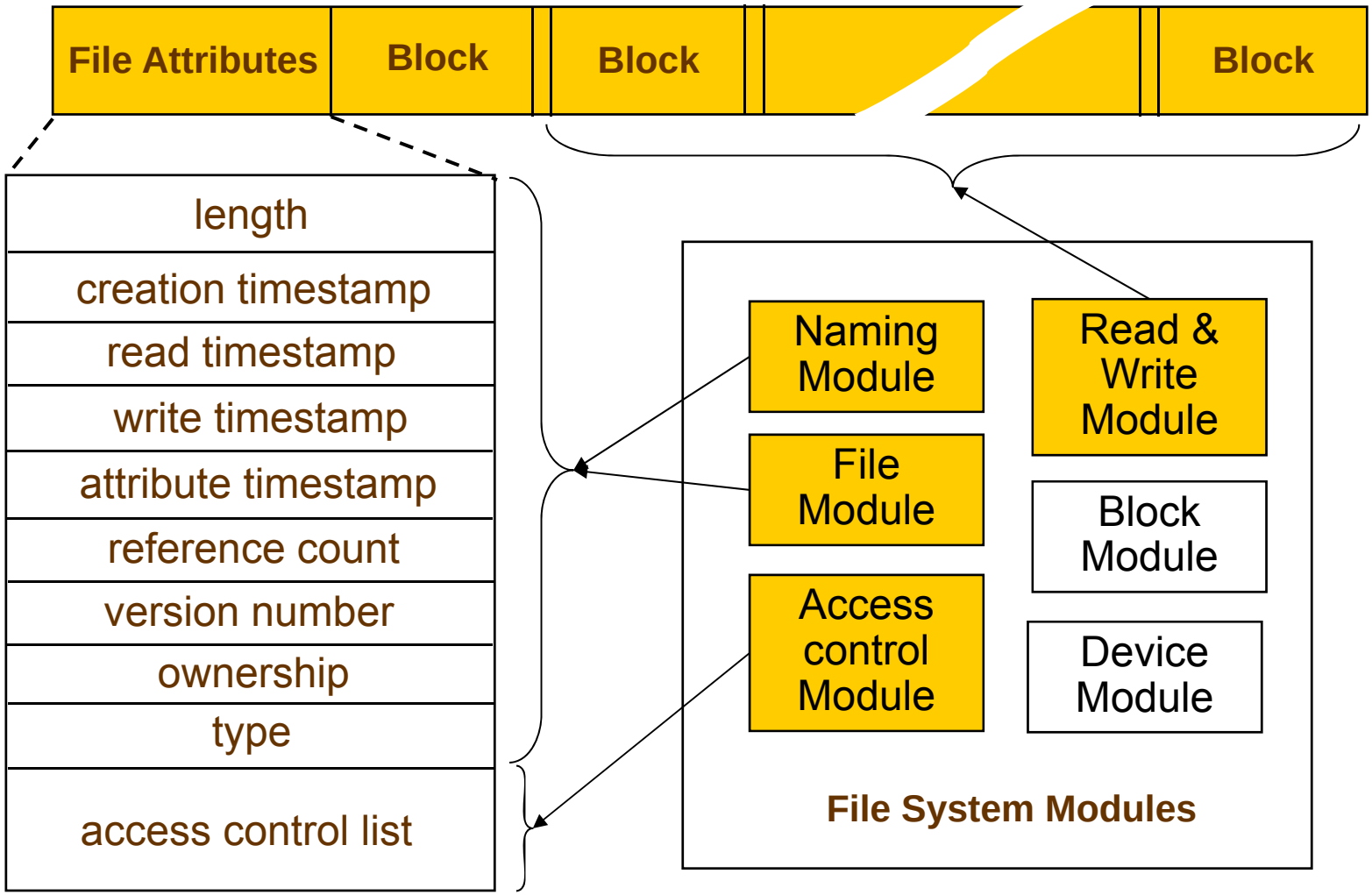
- Um SGF implementa um nível de abstracção em cima de um meio físico de armazenamento
  - Noção de directórios, ficheiros e atributos
    - Um ficheiro é um conjunto de dados associados a um tipo e um nome
    - Um directório é um ficheiro especial que mapeia nomes em descritores internos
    - Os atributos caracterizam os ficheiros e estão guardados nos directórios
- A gestão de ficheiros inclui
  - Nomeação e localização dos ficheiros
  - Gestão dos seus atributos
  - Alocação e libertação de espaço de armazenamento
  - Segurança e protecção dos acessos
  - Provimento de uma API de acesso e respectivos métodos
- Os SGF mantêm informação sobre ficheiros no meio físico
  - *Metadata*

Módulos de um SGF local

Naming module:	relates file names to file IDs
File module:	relates file IDs to particular files
Access control module:	checks permission for operation requested
File access module:	reads or writes file data or attributes
Block module:	accesses and allocates disk blocks
Device module:	disk I/O and buffering

- Organização em camadas
- Cada nível utiliza os serviços das camadas inferiores
- Um SGF distribuído necessita de todas estas camadas mais as necessárias à nomeação, localização, invocação, mensagens e comunicações
- Estas funcionalidades existem em todas as camadas acima excepto nas duas inferiores

Componentes de um Ficheiro



API de Acesso a Ficheiros (exemplo Unix)

fileid = open (name, mode)	Opens an existing file with the given *name.*
fileid = create (name, mode)	Creates a new file with the given *name*. Both operations deliver a file descriptor referencing the open file. The mode is read, write or both.
status = close (filed)	Closes the open file *fileid*.
count = read (fileid, buffer, n)	Transfers n bytes from the file referenced by *fileid* to buffer.
count = write (fileid, buffer, n)	Transfers n bytes to the file referenced by *fileid* from buffer. Both operations deliver the number of bytes actually transferred and advance the read-write pointer.
pos = lseek (fileid, offset, rel)	Moves the read-write pointer to *offset* (relative or absolute, depending on *rel* value).
status = link (name1, name2)	Adds a new name (*name2*) for a file (*name1*).
status = unlink (name)	Removes the file name from the directory structure. If the file has no other names, it is deleted.
status = stat (name, buffer)	Gets the file attributes for file name into buffer.

- A implementação da API associa estado a cada ficheiro utilizado
  - Lista dos ficheiros abertos e I/O pointer por processo
- A realização dos métodos da API está condicionada à verificação de permissões

Requisitos de SGF Distribuídos (i)

- Transparência
  - No acesso: a mesma API deve ser utilizada para aceder a ficheiros locais ou remotos
  - Na localização: os utilizadores e as aplicações devem ver um espaço de nomeação uniforme
  - Na mobilidade: nada deve ser modificado na estrutura dos SGFs quando um ou mais ficheiros são movidos de um local para outro
  - No desempenho: o acesso remoto não deve ser (muito) penalizado
  - Na escalabilidade: o sistema distribuído deve escalar como o local
- Replicação
  - Um ficheiro pode ser replicado em vários exemplares por motivos de eficiência e tolerância a falhas
- Concorrência de Acessos
  - O sistema deve providenciar meios para permitir modificações simultâneas de ficheiros de forma coerente e sem bloqueios

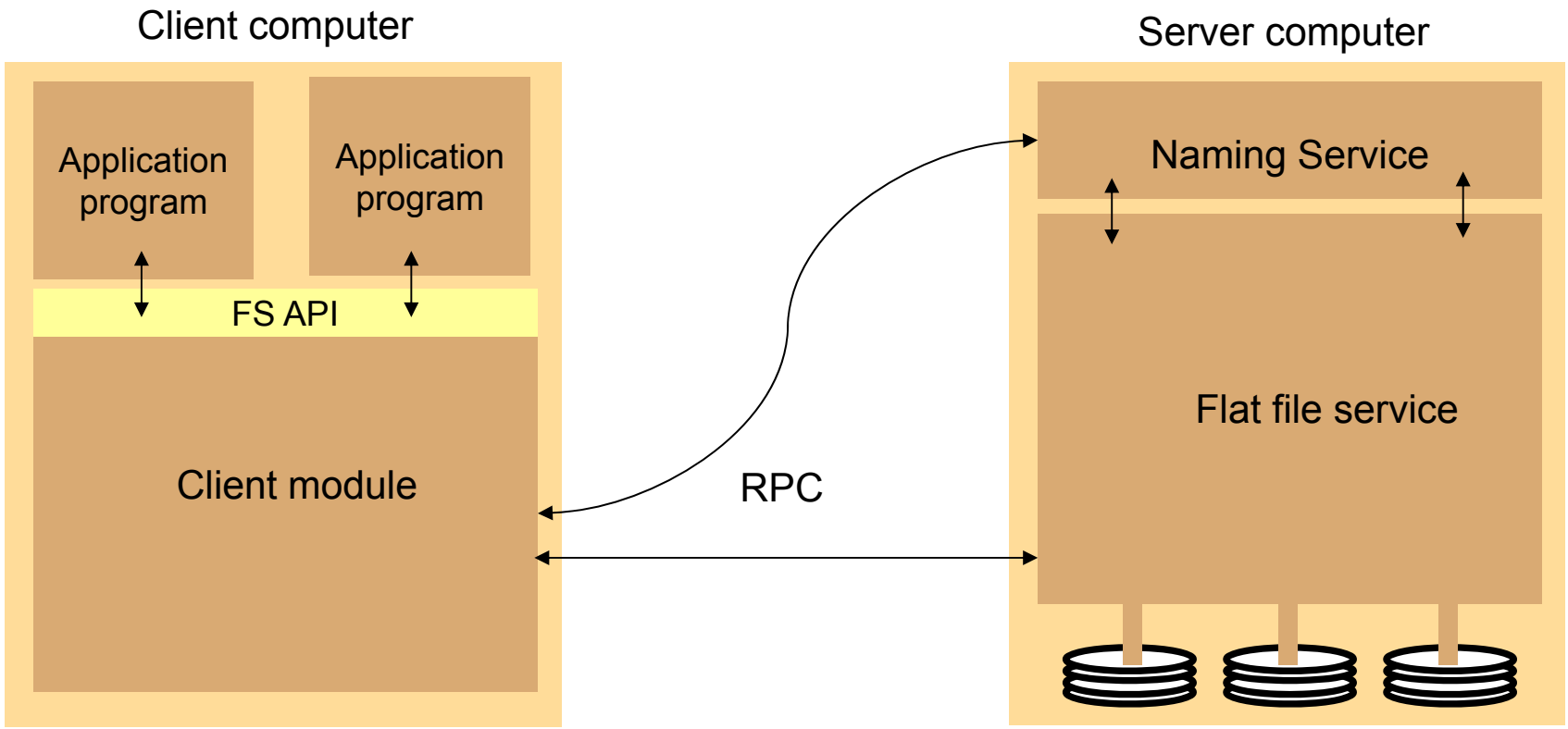
Requisitos de SGF Distribuídos (ii)

- Consistência
  - Semântica de cópia única: todos os processos que acedem a um ficheiro devem ver um conteúdo idêntico, mesmo se o ficheiro estiver replicado em vários locais
- Tolerância a falhas
  - Semântica de evocação "*at most once*"
  - Semântica de evocação "*at least once*" mas fornecendo uma API com métodos idempotentes
- Segurança
  - Controle de Acessos baseado em Listas de Controle de Acesso (ACLs) ou Capabilities
  - Autenticação das invocações remotas para permitir o controle baseado na identidade
- Desempenho
  - Garantir tempos de acesso comparáveis com os SGF locais

Modelo de um Serviço de Ficheiros Distribuído

- Modelo abstracto baseado em 3 serviços distintos:
- Serviço de Nomeação
  - Implementa uma estrutura hierárquica através da criação do conceito de directoria e efectua a resolução de nomes
  - Utiliza a API do serviço de acesso básico para aceder ao armazenamento
- Serviço de Acesso Básico
  - Implementa as operações de base sobre o conteúdo dos ficheiros designados por identificadores internos únicos (UFID – *Unique File Identifiers*)
  - Operações: *create, delete, read, write, get_attribute, set_attribute e access_control, etc…*
- Serviço Cliente
  - Integra e expande os serviços anteriores fornecendo uma API integrada que estende as funcionalidades do SGF local
  - Mantém informação actualizada sobre a localização dos serviços de acesso e de directório
  - Optimiza o desempenho dos acessos através de caching de dados e de atributos de ficheiros

Arquitectura de um Serviço de Ficheiros Distribuído



Métodos da Interface do Serviço de Nomeação

FileId = Lookup (Dir, Name) throws NotFound	Locates the text name in the directory and returns the relevant UFID. If Name is not in the directory, throws an exception
FileId = AddName(Dir, Name) throws DuplicateName	If Name is not in the directory, adds (Name, File) to the directory and updates the file's attribute record. If Name is already in the directory: throws an exception.
DeleteName (Dir, Name) throws NotFound	If Name is in the directory: the entry containing Name is removed from the directory. If Name is not in the directory: throws an exception
NameSeq = ReadDir (Dir, Pattern)	Returns all the text names in the directory that match the regular expression Pattern.

- O objectivo do serviço de nomeação é realizar a resolução de nomes, através de ficheiros especiais contendo mapeamento entre nomes textuais e UFIDs
- Sistema de Ficheiros Hierárquico é construído com base na inserção recursiva de ficheiros de tipo directoria, começando numa raiz
- O módulo cliente fornece às aplicações uma função que realiza a resolução de um *pathname* por invocação recursiva do método *Lookup* para cada elemento do nome na directoria anterior e obtendo o UFID da directoria seguinte
- Podem ser definidos **grupos de ficheiros** como conjuntos de ficheiros com uma localização comum, podendo ser movidos em bloco. Neste caso, o UFID contém um campo com a identificação do grupo - GroupID.

Métodos da Interface do Serviço Básico

Data = read (FileId, i, n) throws NoSuchFile, BadPosition	If 1 ≤ i ≤ Length (File): Reads a sequence of up to n items from a file starting at item i and returns it in Data
write (FileId, i, Data) throws NoSuchFile, BadPosition	If 1 ≤ i ≤ Length (File)+1: Writes a sequence of Data to a file, starting at item i, extending the file if necessary
FileId = Create()	Creates a new file of length 0 and delivers a UFID for it
Delete (FileId) ***throws NoSuchFile***	Removes the file from the file store
Attr = GetAttributes(FileId) throws NoSuchFile	Returns the file attributes for the file
SetAttributes (FileId, Attr) throws NoSuchFile	Sets the file attributes

- Os métodos são acedidos por RPC pelo módulo cliente, não directamente por aplicações
- O identificador FileId só é válido se referenciar um ficheiro presente no repositório do servidor
- Todos os métodos da interface retornam uma excepção de autorização se o cliente não tiver direitos para a sua invocação
- Todos os métodos, excepto *create()*, são **idempotentes** permitindo utilizar um RPC com semântica *at-least-once*
- Os métodos podem ser implementados por um servidor sem estado (*stateless*) pois nenhum método pressupõe a manutenção de informação sobre a operação que o cliente efectuou previamente

Controle de Acessos

- O controle de acessos é implementado no lado do servidor
  - Cada acesso do cliente é validado visto a mensagem poder atravessar um meio desprotegido
  - A validação é feita usando credenciais enviadas em cada RPC
- Duas opções são possíveis
  - Envio da identificação no acto da resolução do nome sendo devolvido o UFID e uma capacidade (*capability)* de acesso para operações subsequentes nesse ficheiro
  - Envio da identificação em cada mensagem
- Cada opção permite uma implementação de servidores diferente
  - Statefull: a primeira opção é usada em AFS
  - Stateless: a segunda opção é usada em NFS
  - Nenhuma delas resolve o problema da substituição da identificação que só pode ser resolvido com recursos a técnicas de encriptação

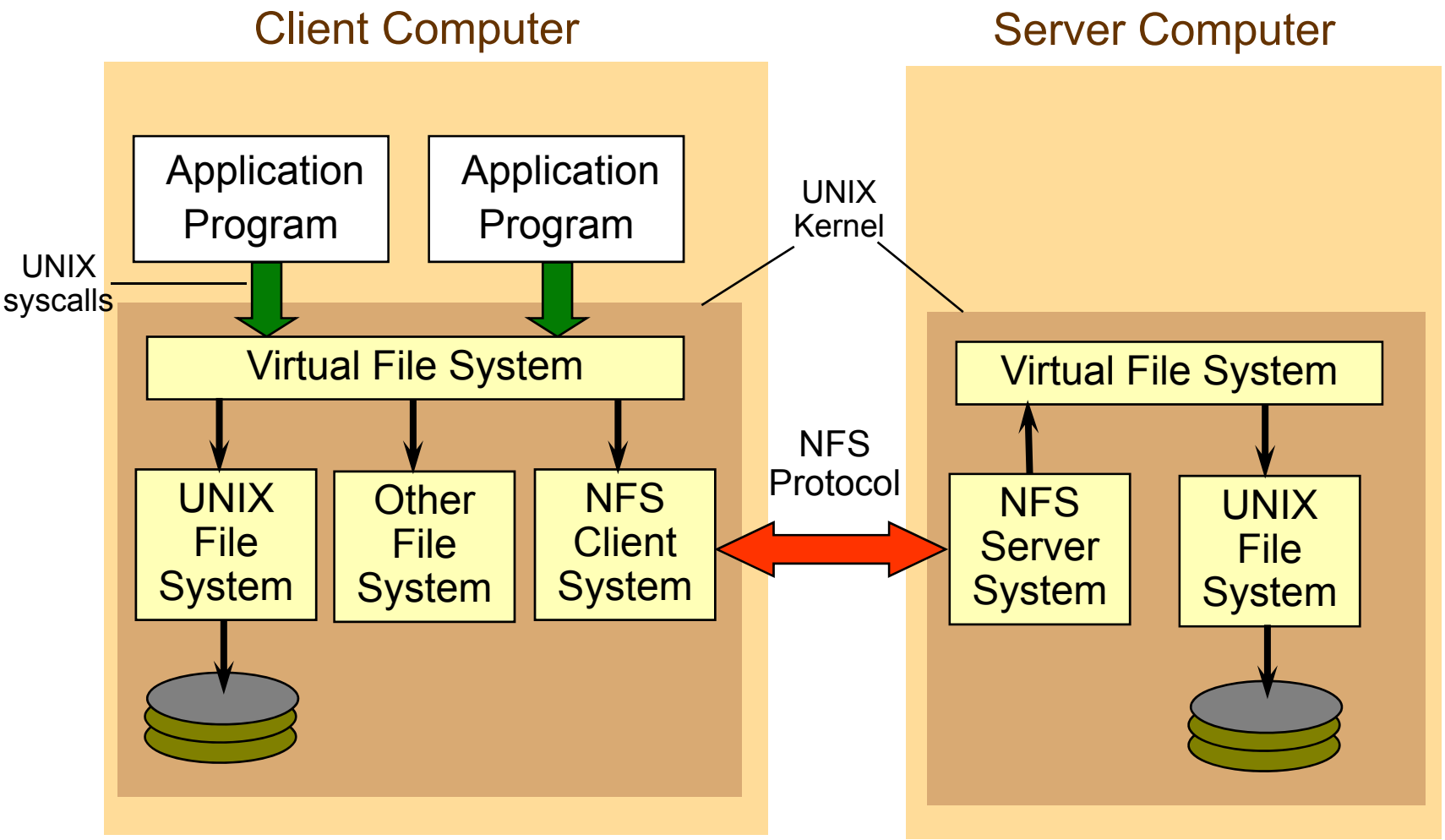
NFS - Network File System

- O Sistema NFS segue o modelo descrito
- Sistema desenvolvido pela Sun nos anos 80
  - Inicialmente em sistema Unix mas suportado por outros
- Utiliza o protocolo NFS para comunicação entre o Cliente e o Servidor
  - Baseado no Sun RPC já estudado
  - Pode funcionar com transporte UDP ou TCP
  - Protocolo aberto que aceita invocações de qualquer cliente que tenha credenciais válidas para o efeito
- Os módulos cliente e servidor residem no kernel
  - Inicialmente o servidor era implementado em modo utilizador
- A integração em sistemas Unix é feito através de uma camada adicional de virtualização do SGF
  - Virtual File System
  - Permite a coexistência de vários SGFs distintos no mesmo sistema

Virtual File System

- Trata-se de um nível de abstracção adicional introduzido no *kernel* entre os *system calls* e a sua implementação em cada SGF
- Permite realizar de forma transparente a distinção entre vários tipos de SGFs locais ou remotos
- Para cada tipo de SGF montado no sistema, o nível VFS
  - instancia um objecto (*vfs*) com as características e operações associadas
  - Guarda uma referência interna para a directoria do volume remoto que fica associada à directoria local onde foi montada
- Para cada ficheiro aberto, o nível VFS
  - Instancia um objecto (*v-node*) com as características e operações associadas ao tipo de ficheiro
    - Se é local aponta para um identificador obtido do SGF local: ***i-node***
    - Se é remoto aponta para um identificador obtido do servidor NFS montado: ***r-node***
  - A realização de uma operação sobre o SGF ou sobre ficheiro é feita de modo transparente por invocação indirecta no objecto de virtualização
    - #define VFS_ROOT(vfsp, vpp) (*(vfsp)->vfs_op->vfs_root)(vfsp, vpp)
    - #define VOP_READ(vp, uiop, iof, cr) (*(vp)->v_op->vop_read)(vp, uiop, iof, cr)
- O VFS mantém um objecto *vfs* por cada volume montado e um *v-node* por cada ficheiro aberto

Arquitectura NFS



- O servidor também pode ser implementado em modo utilizador (versão inicial)

NFS: Implementação Cliente

- Implementado no *kernel* do sistema operativo
- Quando um volume NFS é montado, o módulo cliente
  - Obtém um identificador da directoria raiz do volume remoto
    - O identificador NFS é designado por *file handle*
    - É instanciado um *v-node* associado ao ficheiro remoto, contendo as operações suportadas pelo SGF
    - O v-node neste caso aponta para um *r-node* onde são armazenadas as características do ficheiro remoto, e armazenado o file handle
  - Todas as operações realizadas sobre os ficheiros do SGF remoto vão ter como ponto de partida o *root v-node*
    - O *root v-node* é mantido no objecto *vfs* associado ao SGF remoto e pode ser obtido através do método VFS_ROOT
- Sempre que um novo ficheiro remoto é referenciado
  - É instanciado um novo par (*v-node, r-node*), construído da mesma forma
  - As operações que criam novos *r-nodes* são:
    - *Lookup, create, mkdir*
- Nas operações de leitura e escrita, realiza as transferências de dados por blocos de dados de tamanho constante (tipicamente 4-8 kbytes)
  - Efectua o caching dos dados, indexados pelo *r-node* associado ao ficheiro

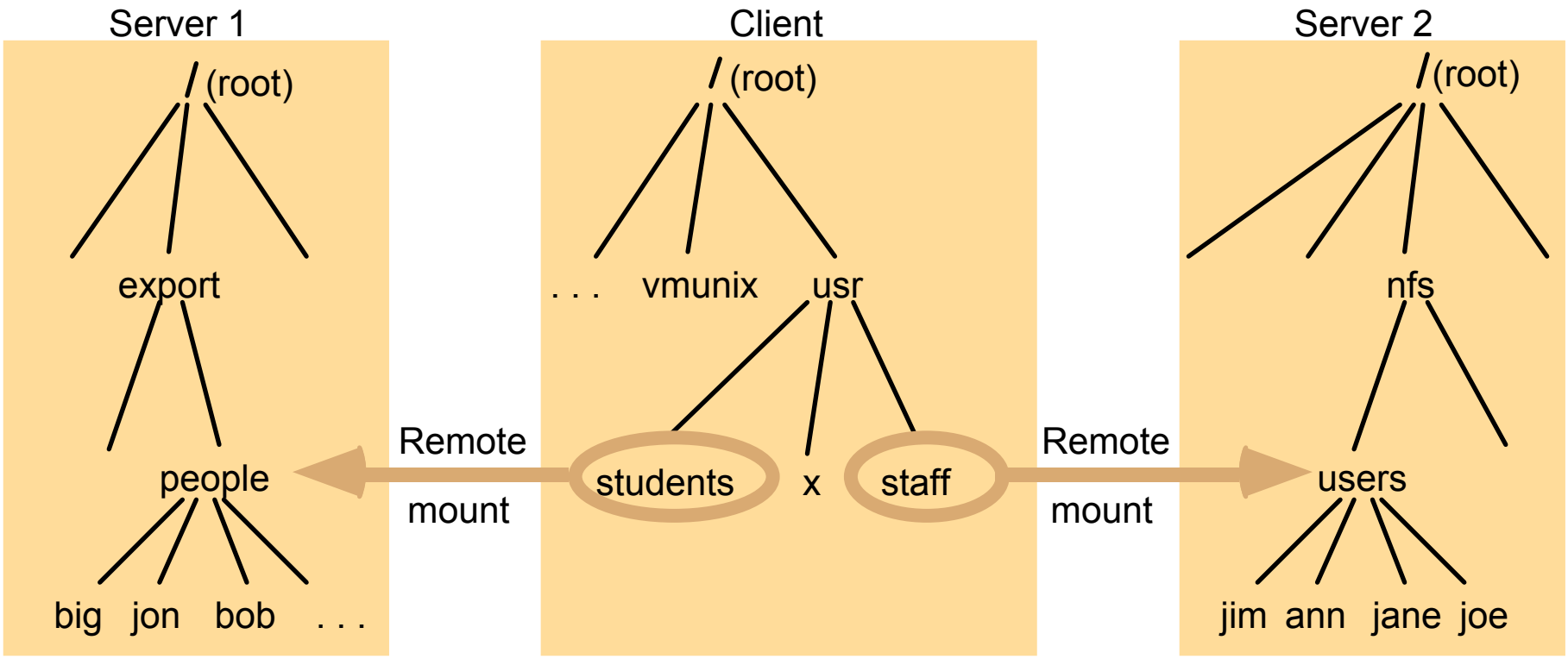
NFS: Implementação Servidor

- O módulo NFS implementa a interface RPC de acesso ao sistema de ficheiros remoto
  - Tem o papel de servidor de directório e de serviço básico de acesso
- Quando um volume NFS é montado, o módulo servidor
  - Cria um identificador para a directoria raiz
  - File handle:

File System Identifier	i-node number of file	i-node generation number of file

- O módulo servidor interage com os volumes locais através do nível VFS
  - Qualquer tipo de SF suportado a nível de VFS pode ser montado por NFS
- A montagem de volumes remotos é realizada através de um serviço separado que corre em modo utilizador : *mountd*
  - Mantém uma tabela de volumes locais e dos clientes que os podem montar
    - /etc/exports
  - O comando Unix *mount* do cliente invoca o serviço *mountd* por RPC indicando o pathname da directoria de montagem
  - *Mountd* verifica as permissões, resolve o nome localmente e retorna um *file handle* para a directoria, que é referenciada no *root v-node* do cliente
- A montagem de volumes pode ser feita de dois modos
  - *Hardmount*: em caso de erro, a operação bloqueia
  - *Softmount*: em caso de erro, a operação retorna erro de timeout depois de um intervalo de tempo

Exemplo de Configuração NFS



Note: The file system mounted at */usr/students* in the client is actually the sub-tree located at */export/people* in Server 1; the file system mounted at */usr/staff* in the client is actually the sub-tree located at */nfs/users* in Server 2.

Métodos da Interface NFS (i)

lookup (dirfh, name) -> fh, attr	Returns file handle and attributes for the file *name* in the directory *dirfh*.
create (dirfh, name, attr) -> newfh, attr	Creates a new file name in directory *dirfh* with attributes *attr* and returns the new file handle and attributes.
remove (dirfh, name) status	Removes file name from directory *dirfh*.
getattr (fh) -> attr	Returns file attributes of file *fh*. (Similar to the UNIX *stat* system call.)
setattr (fh, attr) -> attr	Sets the attributes (mode, user id, group id, size, access time and modify time of a file). Setting the size to 0 truncates the file.
read (fh, offset, count) -> attr, data	Returns up to *count* bytes of data from a file starting at *offset*. Also returns the latest attributes of the file.
write (fh, offset, count, data) -> attr	Writes *count* bytes of data to a file starting at *offset*. Returns the attributes of the file after the write has taken place.
rename (dirfh, name, todirfh, toname) -> status	Changes the name of file *name* in directory *dirfh* to *toname* in directory to *todirfh*
link (newdirfh, newname, dirfh, name) -> status	Creates an entry *newname* in the directory *newdirfh* which refers to file *name* in the directory *dirfh*.

Continues on next slide ...

Métodos da Interface NFS (ii)

symlink (newdirfh, newname, string) -> status	Creates an entry *newname* in the directory *newdirfh* of type symbolic link with the value *string*. The server does not interpret the *string* but makes a symbolic link file to hold it.
readlink (fh) -> string	Returns the string that is associated with the symbolic link file identified by *fh*.
mkdir (dirfh, name, attr) -> newfh, attr	Creates a new directory *name* with attributes *attr* and returns the new file handle and attributes.
rmdir (dirfh, name) -> status	Removes the empty directory *name* from the parent directory *dirfh*. Fails if the directory is not empty.
readdir (dirfh, cookie, count) -> entries	Returns up to *count* bytes of directory entries from the directory *dirfh*. Each entry contains a file name, a file handle, and an opaque pointer to the next directory entry, called a *cookie*. The *cookie* is used in subsequent *readdir* calls to start reading from the following entry. If the value of *cookie* is 0, reads from the first entry in the directory.
statfs (fh) -> fsstats	Returns file system information (such as block size, number of free blocks and so on) for the file system containing a file *fh*.

Resolução de *pathnames*

- Os *pathnames* relativos a ficheiros locais são resolvidos pelo sistema de nomeação local
  - Em Unix, os *pathnames* são transformados em referências internas para *i-nodes* através de um processo iterativo aplicado a cada nível da hierarquia de directórios
- Quando o *pathname* contém um ponto de montagem, a parte do nome respeitante ao SGF remoto não pode ser integralmente resolvida pelo servidor
  - Podem existir mais pontos de montagem locais ou *symbolic links* no resto do *pathname*
  - A resolução do resto do *pathname* continua iterativamente no cliente com base na operação *lookup* aplicada a cada elemento do nome
    - Lookup retorna a referência e atributos do ficheiro procurado no directório
  - Se a referência devolvida for a de um directório e houver mais elementos no *patname*, o processo continua
  - O facto da operação de lookup ser invocada nível de VFS implica que a resolução do nome é independente do número e tipo de pontos de montagem atravessados

Automounter

- É um serviço do cliente que permite montar um volume remoto sempre que um ponto de montagem vazio é referenciado pelo cliente
  - Mantém uma lista de *pathnames* de pontos de montagem com a indicação de um ou mais servidores NFS
- Quando o cliente NFS tenta resolver um *pathname* que atravessa ponto de montagem vazio, é passado um pedido de *lookup()* para o automounter
  - O *automounter* localiza o SF remoto na tabela e envia um pedido ao servidor associado
  - O primeiro a responder ao pedido é montado no ponto indicado
- Pode constituir uma forma primária de repartição de carga por servidores NFS replicados

Caching no Cliente (i)

- Os sistemas Unix usam *caching* de ficheiros locais em memória central
  - O algoritmo utilizado tenta satisfazer os pedidos de leitura a partir do *cache* e usar *read-ahead* e *delayed-write* (30 s máx.) dos ficheiros locais.
- O módulo cliente do NFS efectua *caching* a dois níveis, guardando
  - Os atributos dos ficheiros abertos (*metadata*)
  - Os blocos de dados desses ficheiros que recebe do servidor
  - Utiliza um sistema de *timestamps* para controlar a respectiva validade.
- A cada item no cache são associados dois *timestamps*:
  - Tc - momento o item foi validado pela última vez (hora local)
  - Tm - momento em que o item foi modificado pela última vez
  - No momento T, considera-se válida uma entrada no *cache* sse:
    - $(T - Tc < t)$ ou $(Tm_{client} = Tm_{server})$.
    - Sendo t um intervalo de confiança variável determinado empiricamente
  - No caso de não se verificar a primeira condição, o *timestamp* Tm_{server} é obtido através do método *getattr()*

Caching no Cliente (ii)

- Determinação do intervalo de confiança
  - Compromisso entre consistência e eficiência
  - Valor de t muito grande provoca incoerência, muito pequeno provoca ineficiência do cache
- É determinado de forma adaptativa, sendo menor para ficheiros mais frequentemente modificados
  - Tipicamente t pode variar entre 3 e 30 segundos para ficheiros
  - Entre 30 e 60 segundos para directórios
- Quando um item do *cache* é modificado no cliente é marcado para ser sincronizado no servidor
  - Pode ser escrito imediatamente – *write-through*
  - Pode ser escrito assincronamente quando houver uma sincronização - *sync()* ou quando o ficheiro for fechado
  - A sincronização pode ser feita pelos *bio daemons*
- Não é garantida a semântica de "cópia única"

Caching no Servidor

- Blocos de dados e atributos de ficheiros e directórios lidos por conta do cliente são normalmente guardados no *buffer cache* do servidor
- Na leitura é utilizado o mesmo algoritmo que para os ficheiros acedidos localmente
  - A técnica de *read-ahead* permite antecipar a leitura de blocos contíguos seguintes
- Na escrita de dados ou atributos modificados provenientes de um cliente, podem ser utilizado várias opções
  - O *delayed-write* em que o item modificado pode permanecer no *cache* e só ser sincronizado em disco mais tarde
  - O *write-through* em que todos os itens modificados são sincronizados antes de ser enviado o *reply* da mensagem
  - A segunda opção garante maior consistência mas pode ser penalizante no caso de ser usada sistematicamente
- Na versão NFS v3 é fornecido um método *commit()* adicional
  - Permite sincronizar selectivamente itens modificados que estejam no *cache* do servidor
  - Permite melhorar consideravelmente o desempenho dos *writes*

Crítica do Modelo NFS (i)

- A arquitectura é semelhante ao do modelo abstracto
  - Implementação simples e fácil de adaptar
- Transparência de Acesso
  - A utilização de VFS permite que as aplicações acedam de forma transparente a ficheiros remotos
- Transparência de Localização
  - A utilização de *mount points* permite a criação de um espaço de nomeação global, determinado pelo cliente
- Transparência de Mobilidade
  - A mobilidade de sistemas de ficheiros não é suportada dinamicamente pois necessita a reconfiguração dos *mount points*

Crítica do Modelo NFS (ii)

- Escalabilidade
  - Consegue suportar o aumento de carga de forma correcta em modo de acesso *read-only* através de instalação de mais servidores
  - O facto de não suportar réplicas modificáveis limita a utilização em casos em que um conjunto de ficheiros são modificados muito frequentemente
- Replicação
  - Não suporta a gestão de réplicas modificáveis directamente
  - A utilização do NIS (*Network Information Service*) permite gerir réplicas de ficheiros de autenticação e administração de rede, usando um esquema de replicação *master/slave*.
- Tolerância a falhas
  - O modelo *stateless* e os métodos idempotentes permitem um modelo de falhas semelhante ao dos ficheiros locais
    - Falhas de clientes não afectam os servidores
  - O modo *hardmount* é frequentemente utilizado pois muitas aplicações não são capazes de gerir erros de *timeout*, diminuindo a possibilidade de recuperar de falhas do servidor

Crítica do Modelo NFS (iii)

- Consistência
  - O algoritmo de gestão da validade do *cache* permite um grau de coerência próximo da semântica de cópia única
  - A utilização de ficheiros NFS para realizar comunicação ou coordenação entre processos não é recomendada
- Segurança
  - O nível de segurança é baixo, mas pode ser aumentado com a utilização de Kerberos para autenticação e RPCSEC_GSS para a comunicação
    - Generic Security Services Application Programming Interface: http://www.ietf.org/rfc/rfc2203.txt
- Conclusão
  - NFS é uma opção aceitável para partilha de ficheiros em ambientes sem grandes exigência nos níveis de coerência e segurança
    - Intranets, desenvolvimento partilhado, consulta de informação
  - Em ambientes mais exigentes é necessário adoptar outras soluções
    - AFS, SMB/CIFS

AFS: Andrew File System

- AFS: SGF distribuído desenvolvido na *Carnegie-Mellon University* nos anos 80
  - Chefe de Projecto: Mahadev Satyanarayanan (Satya)
  - http://www.cs.cmu.edu/afs/cs/user/satya/Web/home.html
- Requisitos específicos
  - Permitir acessos distribuídos e seguros através de WANs
  - Garantir excelente escalabilidade a nível do número de utilizadores simultâneos
  - Garantir níveis de despenho e funcionalidades comparáveis ao do acesso a FS locais utilizando técnicas de *caching* avançadas
- Principais características de caching
  - Caching persistente de ficheiros (e não de blocos) no disco local
  - Semântica de cópia única na modificação de ficheiros
  - Utilização de mecanismos de “callback promise”

AFS: Andrew File System

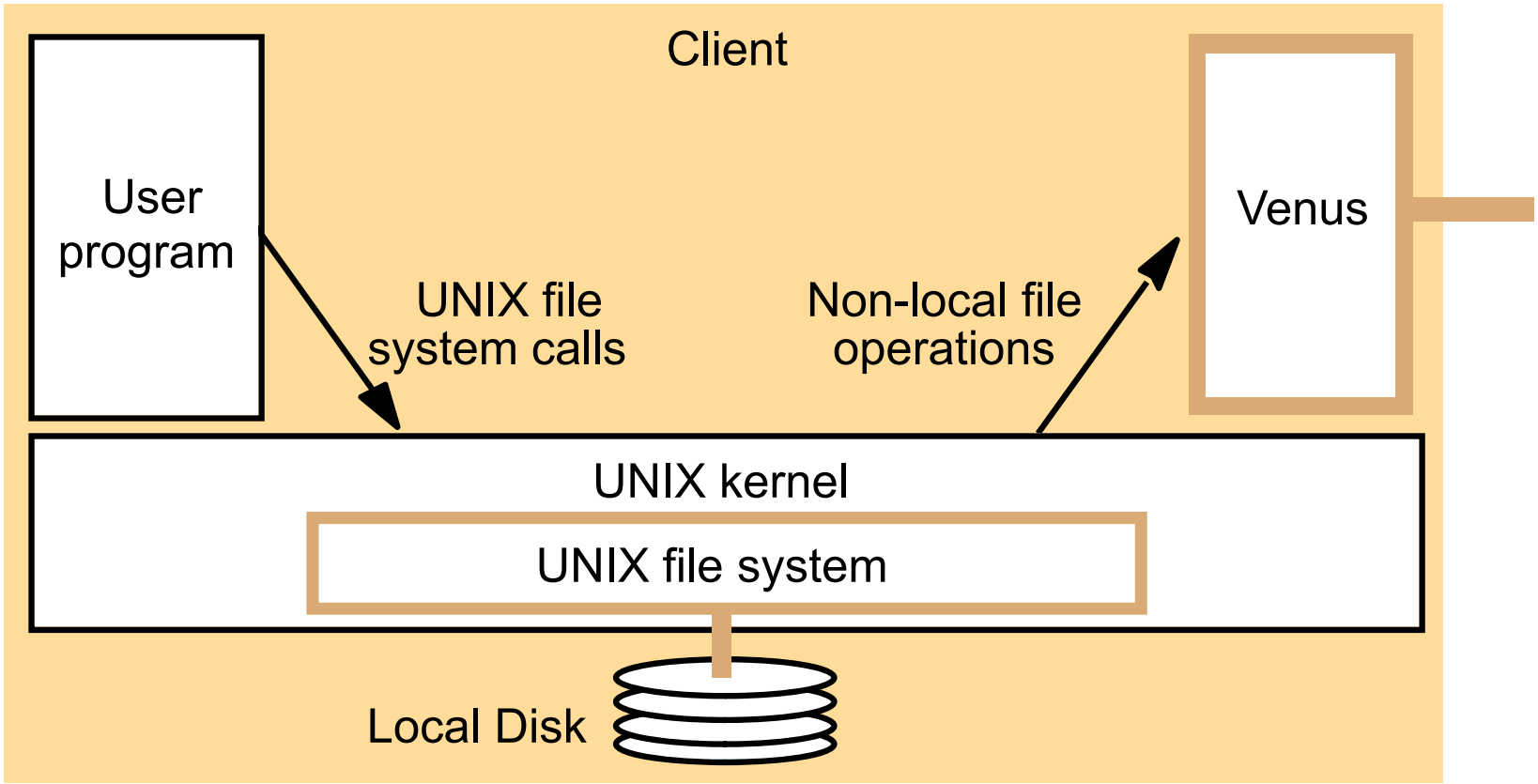
- O AFS implementa com êxito essas funcionalidades
  - Utilização ideal em ambiente com padrões de acesso com muitas operações de leitura, poucas modificações, acesso aos dados em sequencia
  - Não é aconselhável para acessos tipo bases de dados
- Sistema distribuído e suportado pela IBM (Transarc), com inúmeras evoluções
  - DCE/DFS (OSF, OpenGroup)
  - Coda (CMU)
  - OpenAFS
  - NFS v4

Implementação

O AFS é implementado por 3 componentes:

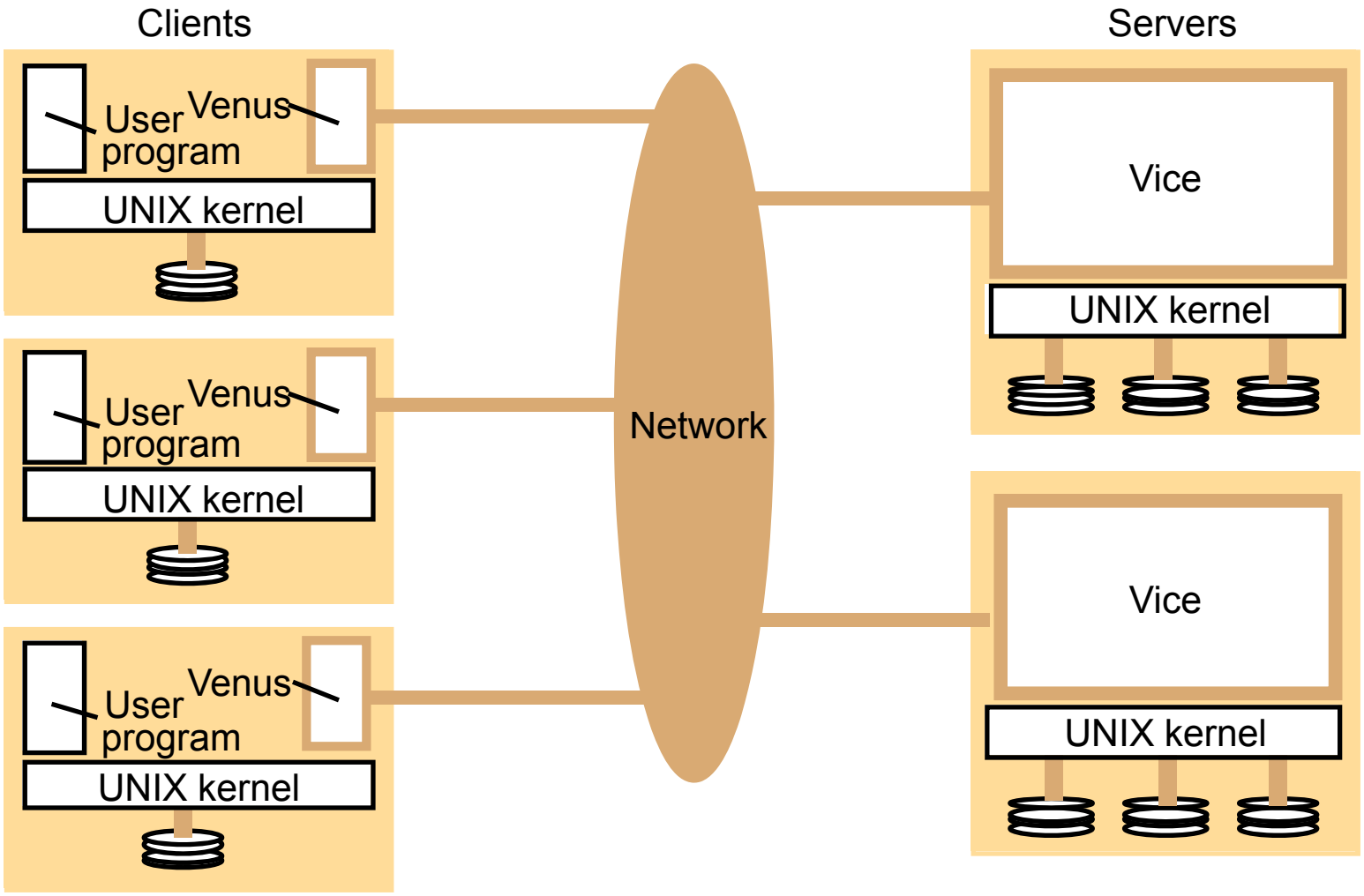
- Módulo Cliente designado por *Venus*
  - Processo em modo utilizador, distingue acessos locais de remotos
  - Funcionalidades do módulo cliente do modelo de referência
- Módulo Servidor designado por *Vice*
  - Processo em modo utilizador
  - Funcionalidades do módulo servidor do modelo de referência
- Modificações específicas do *kernel* do sistema operativo
  - Intercepção dos *system calls* API ficheiros
    - Pode usar VFS, embora inicialmente esse nível não existisse
  - Gestão da coerência do *cache* de ficheiros
  - Segurança Kerberos
- Suportado em Linux (Fedora e RedHat), Solaris, HP-UX, AIX, MacOS X, Windows XP e Vista…

Intercepção de *System Calls*

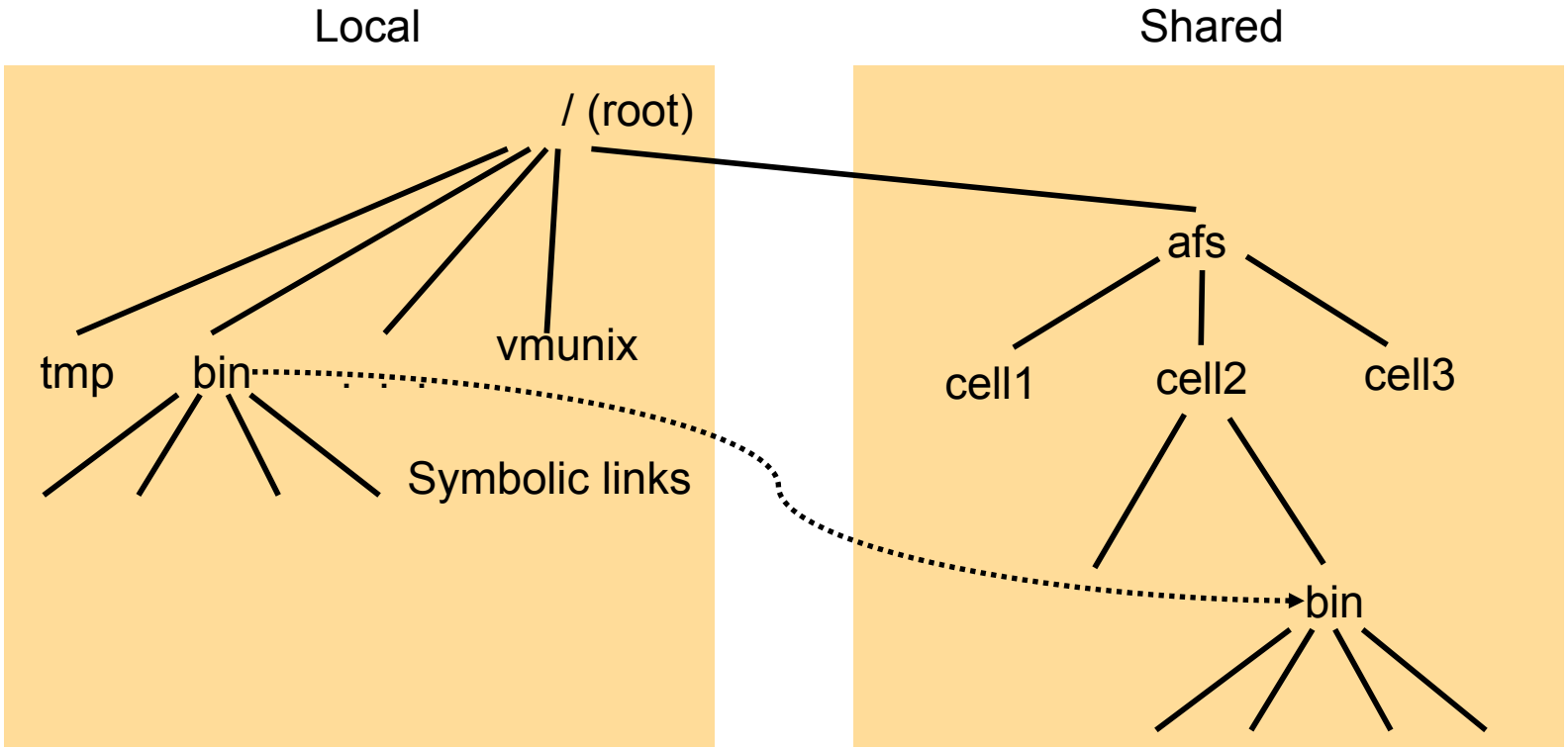


- As operações sobre ficheiros remotos são redirigidas para o processo *Venus*
- O ficheiros abertos são transferidos em blocos de 64 kbytes para uma partição do disco local
- A modificação é feita localmente e propagada para o servidor no *close*

Arquitectura Geral



Espaço de Nomeação



- O espaço de nomeação é uma árvore de tipo Unix, em que o espaço partilhado é visível na directoria /afs
- O espaço de nomes pode estar organizado por células ou domínios
  - ex: /afs/cs.cmu.edu
- A criação de nomeação transparente é feita através de symbolic links
  - /bin -> /afs/cell2/bin

Implementação dos *System Calls*

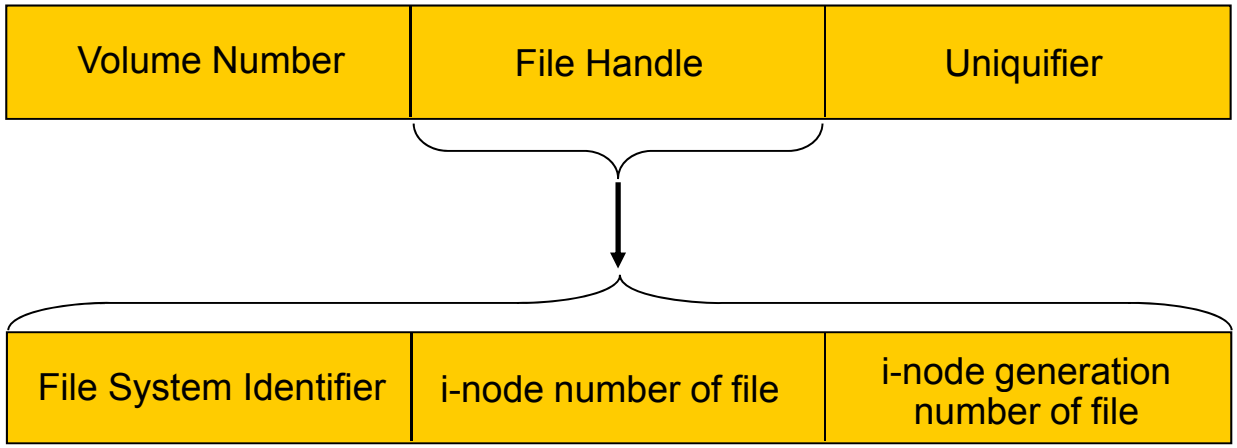
User process	*UNIX kernel*	*Venus*	*Net*	*Vice*
open(FileName, mode)	If *FileName* refers to a file in shared file space, pass the request to Venus.	Check list of files in local cache. If not present or there is no valid *callback promise*, send a request for the file to the Vice server that is custodian 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.

Segurança de Acessos em AFS

- Baseada em Kerberos e Listas de Controle de Acesso (ACLs)
  - Autenticação forte quando comparada com NFS standard
- Antes de aceder a um domínio ou célula AFS é necessário efectuar um *login* usando o comando klog
  - Obtém-se um *token* encriptado (um ticket Kerberos) que fica associado ao utilizador e é passado ao *cache manager* e enviado com todos os acessos ao servidor
  - O *token* é recebido pelo servidor e depois de validado, permite verificar se o utilizador tem direito a efectuar a operação pela consulta dos ACLs que tem sobre o recurso acedido
- ACLs dos directórios AFS
  - **r** read: ability to read files in the directory
  - **l** lookup: ability to look up directory information
  - **i** insert: ability to add a file to a directory
  - **d** delete: ability to delete a file from a directory
  - **w** write: ability to write to files in a directory
  - **k** lock: ability to have processes lock files in the directory
  - **a** administer: ability to change the permissions on the directory
- Exemplo de ACLs
  - system:administrators rlidwka
  - system:authuser rlidwk
  - system:anyuser l

Identificador de Ficheiro Remoto

- Os ficheiros remotos são designados por um *File_Id* (fid) contendo um *file_handle* NFS e dois campos adicionais:
  - Volume Number: designa um grupo de ficheiros lógico ao qual pertence o ficheiro
  - Uniquifier: número único que garante a únicidade do File_Id



Consistência do *Cache* AFS

- Quando *Vice* envia uma cópia de ficheiro a *Venus*, envia igualmente uma *callback promise (cp)*
  - *Token* que garante que o cache irá receber uma notificação (*callback*) do servidor se o ficheiro for modificado (no servidor ou noutro cliente)
  - A CP é guardada no disco local associada aos dados do ficheiro e tem dois estados
    - Valid: indica que os dados no cache estão actualizados
    - Cancelled: quando os dados estão desactualizados
  - A passagem de *valid* para *cancelled* é feita pela recepção do callback
  - Venus pode validar o estado de um *callback* através de um pedido explícito a Vice
    - Validação de ficheiros no *cache* depois de um *reboot*
  - A duração de validade de um CP é limitada a alguns minutos para obviar a perca de *callbacks* devido a erros de comunicação *Vice->Venus*
- O mecanismo obriga o servidor a manter estado sobre o cliente
  - Lista dos *callbacks* pendentes em cada cliente
  - Mas permite reduzir drasticamente as interacções entre o cliente e servidor

Métodos da Interface do Serviço Vice

Fetch(fid) -> attr, data	Returns the attributes (status) and, optionally, the contents of file identified by the *fid* and records a callback promise on it.
Store(fid, attr, data)	Updates the attributes and (optionally) the contents of a specified file.
Create() -> fid	Creates a new file and records a callback promise on it.
Remove(fid)	Deletes the specified file.
SetLock(fid, mode)	Sets a lock on the specified file or directory. The mode of the lock may be shared or exclusive. Locks that are not removed expire after 30 minutes.
ReleaseLock(fid)	Unlocks the specified file or directory.
RemoveCallback(fid)	Informs server that a Venus process has flushed a file from its cache.
BreakCallback(fid)	This call is made by a Vice server to a Venus process. It cancels the callback promise on the relevant file.

São omitidos os métodos de acesso a directórios, renomeação e teste de validade de CP

Semântica de Modificação

- A gestão do *cache* com *callback promises* permite obter uma semântica muito próxima da dos ficheiros locais
  - Todavia a garantia total de coerência não é conseguida
- Depois de um *open()* com sucesso:
  - Se o ficheiro não estava no cache, a cópia trazida do servidor é a última versão no servidor
  - Se estava, dois casos são possíveis
    - O *token* (CP) do ficheiro é válido e está dentro do período de validade T
    - Um *callback* enviado nesse espaço de tempo perdeu-se e a cópia está desactualizada T segundos relativamente à última versão
- O grau de consistência não é muito diferente do obtido num sistema Unix local, em que vários processos que modificam o mesmo ficheiro devem utilizar primitivas de sincronização
  - Utilização do system call *flock()*

Evolução

- A área dos SGFs distribuídos continua em evolução
- A partir dos modelos originais de NFS e AFS foram desenvolvidos inúmeros projectos de investigação que deram bons resultados
- NFS
  - Spritely NFS (sistema Sprite - Berkeley) e NQNFS adicionaram gestão de coerência de *cache* baseada em *call-backs e* estado no servidor garantindo exclusividade de acesso em escrita
  - WebNFS: acesso a servidores NFS pela Web
  - NFS v4: significativas modificações do protocolo para integrar file locking, segurança forte e outras funcionalidades de AFS e Coda
  - Clustering Coherent NFS: extensões para clustering
- AFS
  - DCE/DFS versão mais robusta da OSF baseada em DCE e VFS
  - Coda introduz acesso desconectado para gerir mobilidade e algoritmos de acesso mútuo baseados em *oportunistic locking*
  - *Odissey* - projecto de investigação em CMU na área de *Multimedia e Mobile Computing* que utiliza e extende os conceitos de AFS e Coda.

Coda vs. AFS *cartoon*

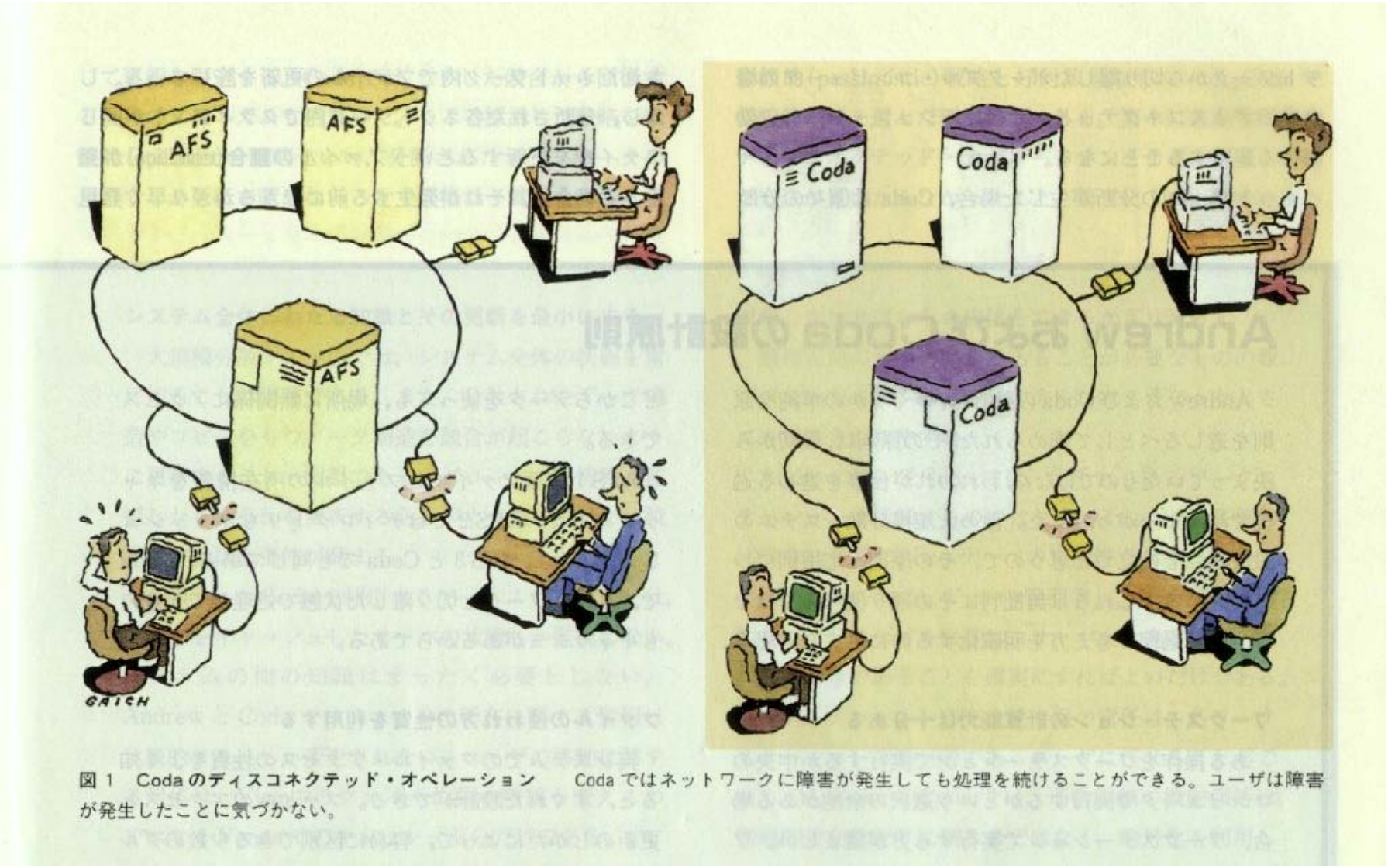


図 1　Coda のディスコネクテッド・オペレーション　Coda ではネットワークに障害が発生しても処理を続けることができる。ユーザは障害が発生したことに気づかない。

Referências e Trabalho Complementar

- Descrição Técnica de AFS e Coda (obrigatório)
  - http://www.cs.cmu.edu/~coda/docdir/scalable90.pdf
- Satya: um dos “gurus” dos SGFs distribuídos
  - http://www.cs.cmu.edu/afs/cs/user/satya/Web/home.html
    - Mais referências para AFS, Coda e Aura
- Samba/CIFS
  - http://www.ubiqx.org/cifs
- OpenAFS
  - http://www.openafs.org
- Linux NFS
  - http://wiki.linux-nfs.org
    - Links para NFS v4 e Cluster Coherent NFS
- NFS v4
  - http://www.nfsv4.org
- Trabalho complementar
  - Ler o artigo sobre AFS e Coda e comparar as várias versões de NFS

Fim do Capítulo V

Resumo dos conhecimentos adquiridos:

- Conceito de SGF distribuído
- Caracterização
  - Vários tipos de distribuição
  - Semântica de acessos e de coerência
- Arquitecturas de SGF distribuídos
  - Modelo conceptual
  - Operações básicas
- Implementações
  - NFS - sistema sem estado, métodos idempotentes, coerência fraca
  - AFS - sistema com estado, forte coerência
- Problemática do *caching*
  - Gestão do cache NFS e AFS: dois modelos típicos