

Systemes de fichiers répartis



INF346

Samuel Tardieu

sam@rfc1149.net

Pourquoi partager des fichiers?



- Travail en équipe
 - Projets à plusieurs développeurs
 - Mise à jour de base de connaissances
- Travail personnel
 - Utilisation de machines banalisées
 - Utilisation de machines différentes
 - Station de travail avec un large disque et sauvegardée
 - Ordinateur portable
- Accès optimisé aux données

Ce problème n'est pas récent



- En 1985, DEC fournit VAXClusters
 - Partage de périphériques de mémoire de masse comme s'ils étaient locaux (notion de cluster)
 - Basé sur une politique de droits rigide, à la VMS
- Au même moment, Sun livre NFS
 - Distinction entre fichier local et fichier distant
 - Plus souple à administrer, plus orienté Unix

Comment partager des fichiers ? (plan du cours)



- Accès exclusif à distance
 - VAXClusters, partage de périphériques
- Accès partagé à distance
 - NFS, CIFS
- + redondance
 - AFS, Coda, Intermezzo
- + haute-disponibilité
 - GoogleFS, Hadoop
- Disques intelligents

NFS: le système le plus basique

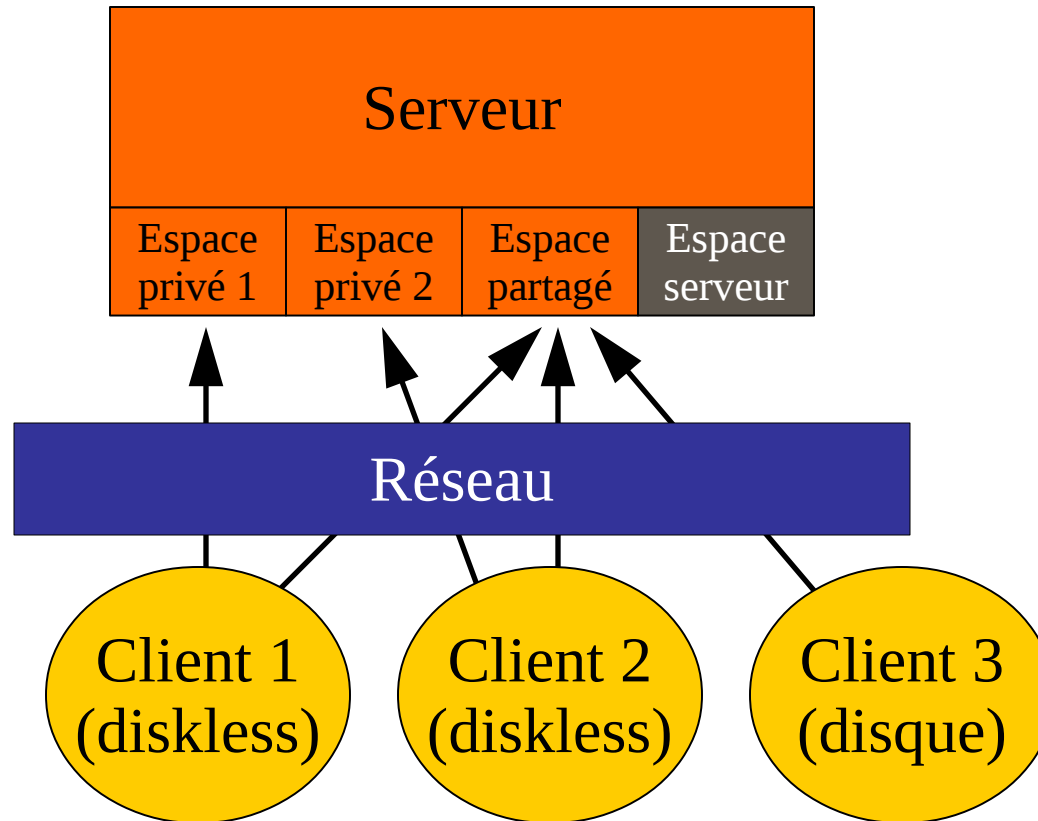
- NFS a été prévu pour ressembler au système de fichiers Unix
- NFS utilise les RPC de Sun
- Le protocole ne stocke pas l'état du serveur
- Le client n'utilise pas (ou peu) de cache
 - Impossibilité de savoir si une donnée a changé sur le serveur
 - Certains clients utilisent un cache à sémantique douteuse (exemple: « réutilise un bloc de données s'il date d'il y a moins de 30 secondes »)

NFS: un protocole simple



- Opérations en lecture
 - getattr, lookup, readdir, read, readlink, statfs, mount
- Modifications de la hiérarchie de fichiers
 - mkdir, link, symlink, rename, unlink, rmdir
- Modifications de fichiers
 - setattr, write
- Ces opérations ressemblent à celles accessibles sur un système de fichiers local

NFS: adapté aux clients sans disque



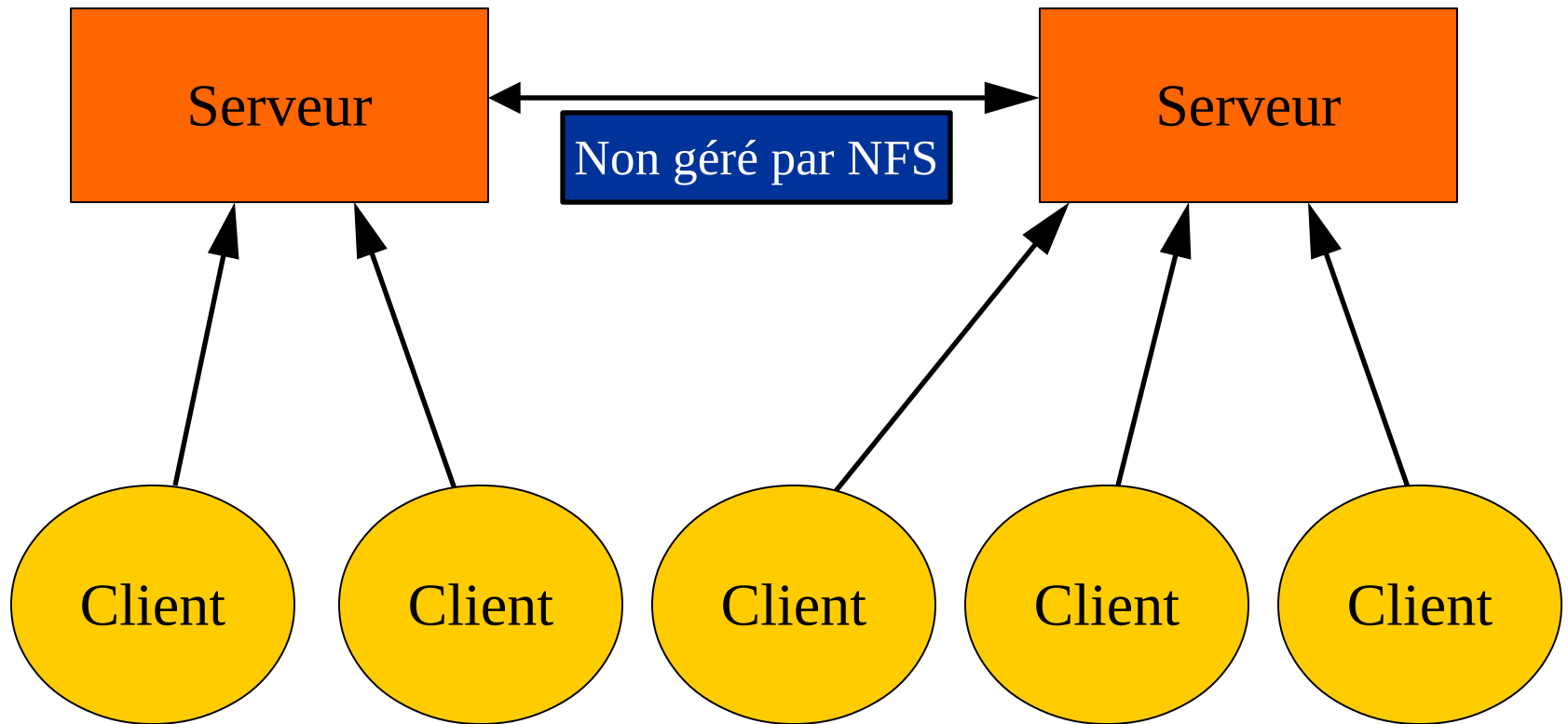
NFS: exemple de scénario

- Le client charge un fichier dans son éditeur
 - Le fichier est demandé au serveur et récupéré
- Le serveur est arrêté
- Le client modifie le fichier dans son éditeur
 - Il ne remarque pas la « disparition » du serveur
- Le serveur redémarre
- Le client sauve le fichier
 - **Tout se passe bien**, le serveur ne conserve pas d'état, les identificateurs restent valables
 - Problème en cas de changement de configuration du serveur (*NFS stale file handle*)

NFS: quel environnement?

- Aucune donnée n'est cachée localement
 - La vitesse du réseau influence le débit des données disponibles
 - Le temps d'aller/retour influence la durée de chaque opération
- NFS n'est adapté qu'aux réseaux locaux rapides
- NFS v3 et v4 réduisent ces problèmes
 - Certaines opérations sont asynchrones
 - Il est possible de lire plusieurs entrées de répertoire en une seule opération

NFS: répliquer les données est possible, mais...



...pas de cohérence forte

Nouveautés de NFS v4

- Un client peut informer le serveur de ses intentions (lock, lecture, écriture, ...) et savoir ce que prévoient les autres clients
- En v4, possibilité d'utiliser des callbacks (cf. plus loin dans le cours) et d'utiliser un cache
- Streams privilégiés (lien avec le serveur), datagrammes obsolètes
- Cache en mode « close-to-open consistency »

Sécurité dans NFS



- La sécurité est basée sur les adresses IP
- Possibilité de se substituer à une machine légitime
- La sécurité implique l'intégrité du réseau et de ses administrateurs

NFS: disponibilité

- Conçu à l'origine par Sun Microsystems
- Disponible pour tous les clones d'Unix
- Implémentations commerciales disponibles pour Windows NT et VMS
- Disponible pour les systèmes embarqués (notamment ceux sous Linux)
- v1 et v2 obsolètes
- v3 très utilisé, v4 en cours de déploiement

NFS: inconvénients



- Il n'y a quasiment plus de stations sans disque
- On a toujours besoin de fichiers partagés
 - On souhaiterait utiliser le disque local
- NFS ne permet pas cela

Common Internet File System



- Développement
 - Créé par IBM sous le nom Server Message Block (SMB)
 - Repris et amélioré par Microsoft
 - Renommé plus tard en CIFS
- Principes proches de NFS
- Supporte l'authentification par utilisateur
- Disponible sous Unix (samba)
- Partage également les ressources (imprimantes)

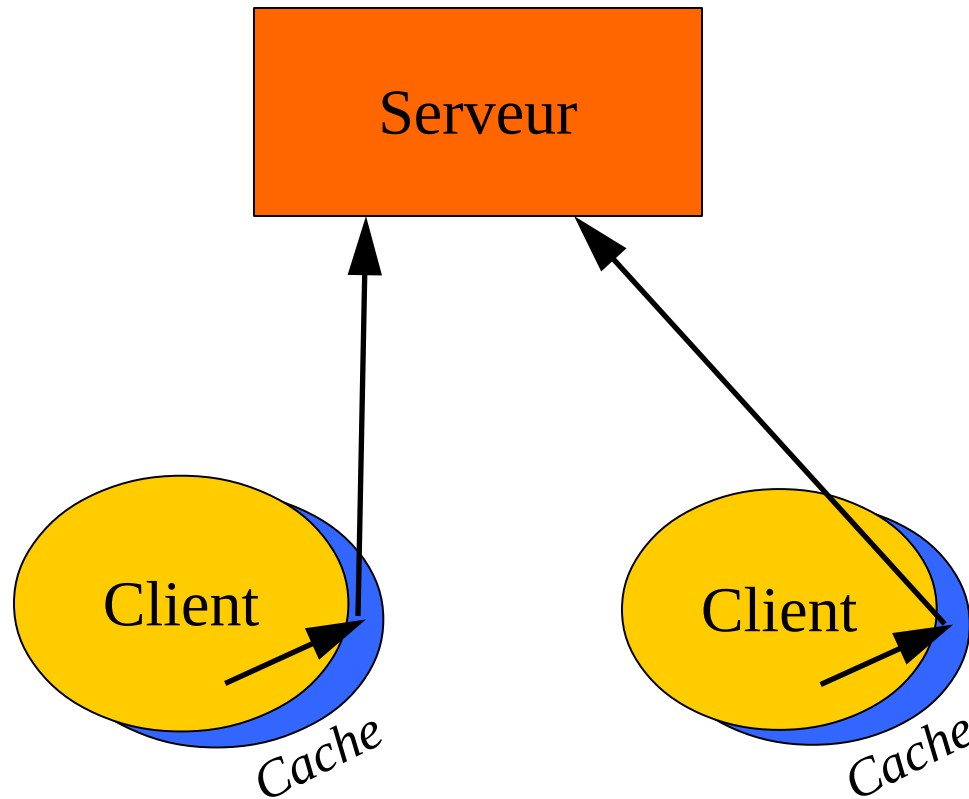
AFS: utilisation d'un cache local

- AFS: Andrew File System
- Utilisation d'un cache côté client
 - Stocke les fichiers
 - Stocke les données de répertoire
- Utilisation d'un système de « callbacks »
 - Le client peut utiliser sa copie locale en la considérant valide
 - Le serveur le prévient lorsque les données cachées sont modifiées

AFS: propriétés du cache

- Le cache est persistant
 - Résiste au redémarrage de la machine
 - Stocke un numéro de version associé à chaque donnée
- Accès concurrents
 - Accès en lecture
 - Ne nécessite par défaut aucune opération réseau
 - Accès en écriture
 - Provoque un demande d'invalidation de cache par le serveur à tous les autres clients pour les données modifiées

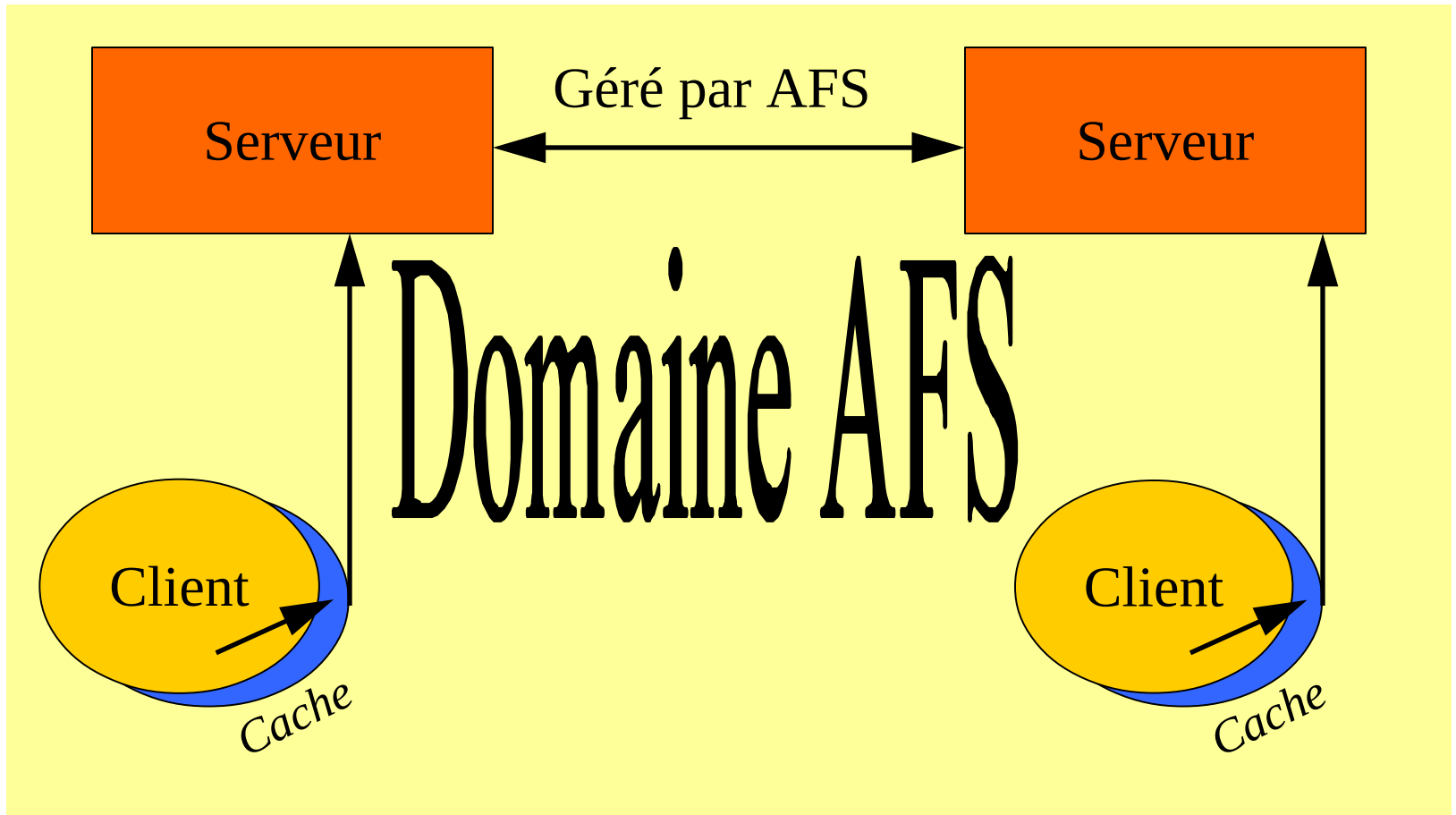
AFS: illustration du cache



AFS: organisation

- Espace de nommage unique: /afs
- La cellule est un domaine administratif
 - Un ou plusieurs serveurs
 - Zéro ou plusieurs clients accédant à ces données
 - Chaque cellule exporte des volumes montés sous /afs
- Administration du client minimale
 - Nom des cellules AFS accédées
 - Le client contactera de lui-même la cellule pour trouver un serveur offrant le volume demandé

AFS: réplication illustrée



AFS: protocole simple mais puissant

- Client -> Serveur
 - Lecture des données
 - FetchData, FetchACL, FetchStatus
 - Ecriture de données
 - StoreData, StoreACL, StoreStatus
 - Modification de la hiérarchie de fichiers
 - RemoveFile, CreateFile, Rename, Link, Symlink, MakeDir, RemoveDir
 - Gestion des volumes
 - GetVolumeStatus, SetVolumeStatus, GetRootVolume

AFS: protocole... (suite)

- Vivacité et sémantique
 - GetTime, BulkStatus, GiveUpCallbacks
- Verrouillage
 - SetLock, ExtendLock, ReleaseLock
- Serveur -> Client
 - Sémantique
 - Probe, Callback, InitCallbackState
- Sécurité prise en compte
 - Gestion de jetons par Kerberos
 - ACL (Access Control List) par répertoire

AFS: disponibilité



- Premières implémentations développées par CMU (Carnegie Mellon University)
- Implémentation commerciale développée par TransArc
- Implémentation libre, ARLA, en cours de développement
- Très utilisé dans les universités américaines

Comparaison NFS/AFS

- Authentification et confidentialité des données
 - NFS fait confiance au client
 - AFS impose une vérification d'identité externe (Kerberos)
- Gestion du cache
 - NFS ne gère aucun cache
 - AFS gère un cache local persistant
- Architecture
 - NFS: le client appelle le serveur
 - AFS: le serveur peut également appeler le client

Comparaison NFS/AFS (suite)

■ Sémantique

- NFS: la dernière écriture gagne (comme Unix)
- AFS: la dernière fermeture de fichier gagne

■ Performances

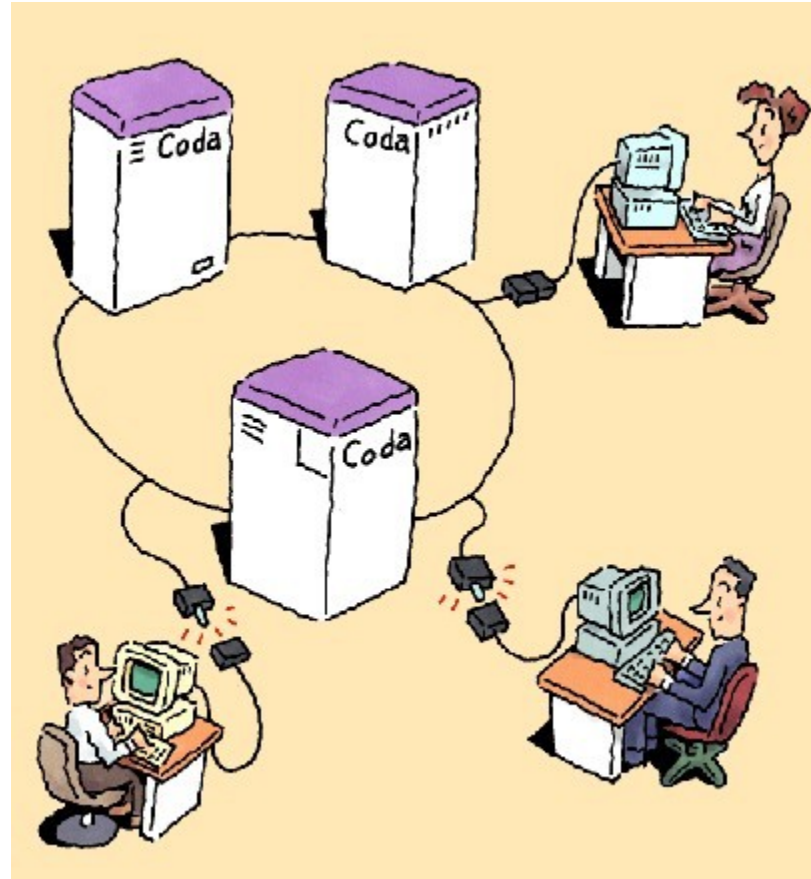
- NFS: nécessite un réseau local rapide
- AFS: nécessite un réseau local
 - local: beaucoup d'opérations synchrones
 - rapide: uniquement pour les lectures après modification, ou la première lecture

Coda: la souplesse en action



- Développé par CMU
- Dérive de AFS version 2
- Un volume peut être fourni par plusieurs serveurs
- Synchronisation des réplicas
- Adaptation à la bande passante disponible
- Support des opérations en mode déconnecté

Coda en une image



Coda: scénario typique

- Un portable est préchargé avec les fichiers utilisateurs
- L'utilisateur travaille sur ses fichiers dans l'avion
- Il se connecte à bas débit (PPP) depuis son hôtel pour synchroniser son portable
- Il arrive sur le site de la filiale, où les fichiers sont répliqués, d'où un accès rapide

Le mode déconnecté est amené à devenir de plus en plus populaire (informatique nomade)

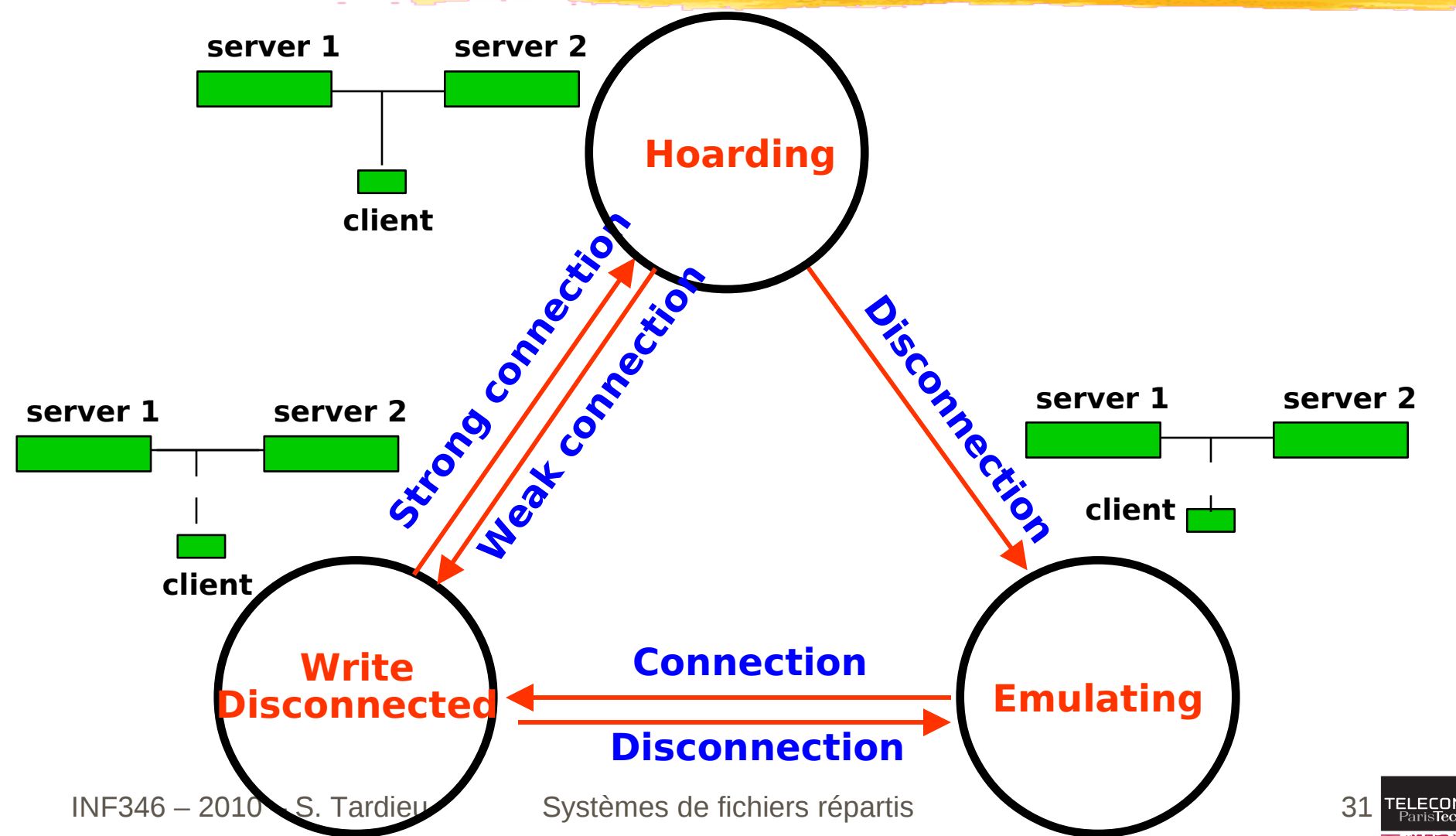
Coda: réintégration des changements

- Le client récupère le fichier de changement du client
- Le serveur intègre les changements dans l'existence des fichiers et leurs attributs
- Le serveur demande au client de lui envoyer (backfetch) les modifications effectuées dans le fichier
- Que se passe-t-il si plusieurs clients ont fait des changements contradictoires?

Coda: fonctionnement d'un client

- Si le client est bien connecté à un serveur
 - Les écritures sont synchrones
 - Le client est averti des modifications sur les fichiers qu'il possède en copie (callbacks)
- Si le client n'est pas (ou est mal) connecté
 - Les écritures se font à l'aide d'un fichier de modifications locales
 - Ces modifications sont réintégréées de manière asynchrone si le client est mal connecté, plus tard s'il n'est pas connecté
 - Les fichiers demandés sont prioritaires

Coda: configuration possibles d'un client



Coda: modifications locales



- Le fichier de modification locale...
 - ...enregistre la création, la modification, l'effacement et le changement des attributs d'un fichier
 - ...n'enregistre pas le contenu de la modification
 - ...subit un nettoyage permanent
 - Une création de fichier suivi d'un effacement
 - Deux modifications successives du même fichier
 - Des modifications suivies d'un effacement
- Ce fichier sera transmis plus tard au serveur

Coda: gestion des conflits



- Dans le cadre d'écritures synchrones
 - Chaque écriture est précédée d'une demande de vérification de la version courante
 - Cette demande est réalisée auprès de tous les serveurs
 - Tous les réplicas effectuent les mêmes opérations
- Dans le cadre d'écritures asynchrones
 - Coda essaye de résoudre les faux conflits
 - L'utilisateur doit résoudre les autres à la main

Coda: réplication des serveurs

- Le modèle est « write all, read one »
- Chaque volume possède un Volume Storage Group (VSG)
- Chaque fichier ou répertoire possède un « version-stamp », permettant de différencier les versions
- Un client accédant à une ressource vérifie l'égalité des version-stamp des serveurs disponibles (AVSG, A pour Available)

Coda: fonctionnement en groupe partiel

- Pour modifier une ressource, un client
 - demande à chaque serveur (multi-RPC) s'il accepte la modification à partir de la version connue
 - valide ou invalide la demande, selon les réponses, par le même mécanisme
 - en cas d'invalidation, le client déclenche une résolution de conflit entre serveurs
- Si $AVSG \neq VSG$, les modifications sont enregistrées par les serveurs, comme s'ils étaient des clients déconnectés

Coda: résolution des conflits entre serveurs

- Résolution basique
 - Comptage du nombre de modifications par serveur dans un « version-vector » pour chaque ressource
 - Identification éventuelle d'un réplica dominant
 - Permet surtout d'importer un objet sur un réplica qui ne le possède pas
- Résolution complexe
 - Lorsque des modifications concurrentes ont vraiment eu lieu
 - Nécessite un protocole en 5 étapes

Coda: résolution complexe

- Etape 1
 - Election d'un coordinateur parmi les réplicas
- Etape 2
 - Le coordinateur collecte les fichiers d'historique et les version-stamp de tous les réplicas, les structure et les renvoie à tous les réplicas
- Etape 3
 - Chaque réplica utilise cet historique structuré, effectue les opérations qu'il a manqué et renvoie au coordinateur les incohérences résultantes

Coda: résolution complexe (suite)

- Etape 4
 - L'assistance d'un utilisateur est requise afin de pouvoir résoudre les conflits
- Etape 5
 - Lorsqu'il n'y a plus d'incohérence, chaque serveur installe le même version-vector
 - Les fichiers d'historique sont
 - remis à zéro si $AVSG = VSG$
 - conservés sinon, car de nouveaux conflits pourront apparaître entre ce groupe (AVSG) et les autres serveurs (VSG-AVSG)

Coda: disponibilité

- Disponible à l'origine sur Linux
- Fonctionne maintenant sur Solaris et FreeBSD
- A été porté sur Windows
- Est peu utilisé actuellement
 - Projet universitaire, au rythme de développement irrégulier (version 6.9.4 en mars 2009, 6.9.0 en novembre 2006)
 - Système de fichiers peu répandu

Comparaison AFS/Coda



- Authentification et sécurité
 - Modèles comparables, basés sur des jetons d'authentification
- Accès aux données
 - Vitesse d'accès comparables
 - Possible en mode déconnecté avec Coda
 - Coda supporte un réseau à faible débit par intermittence

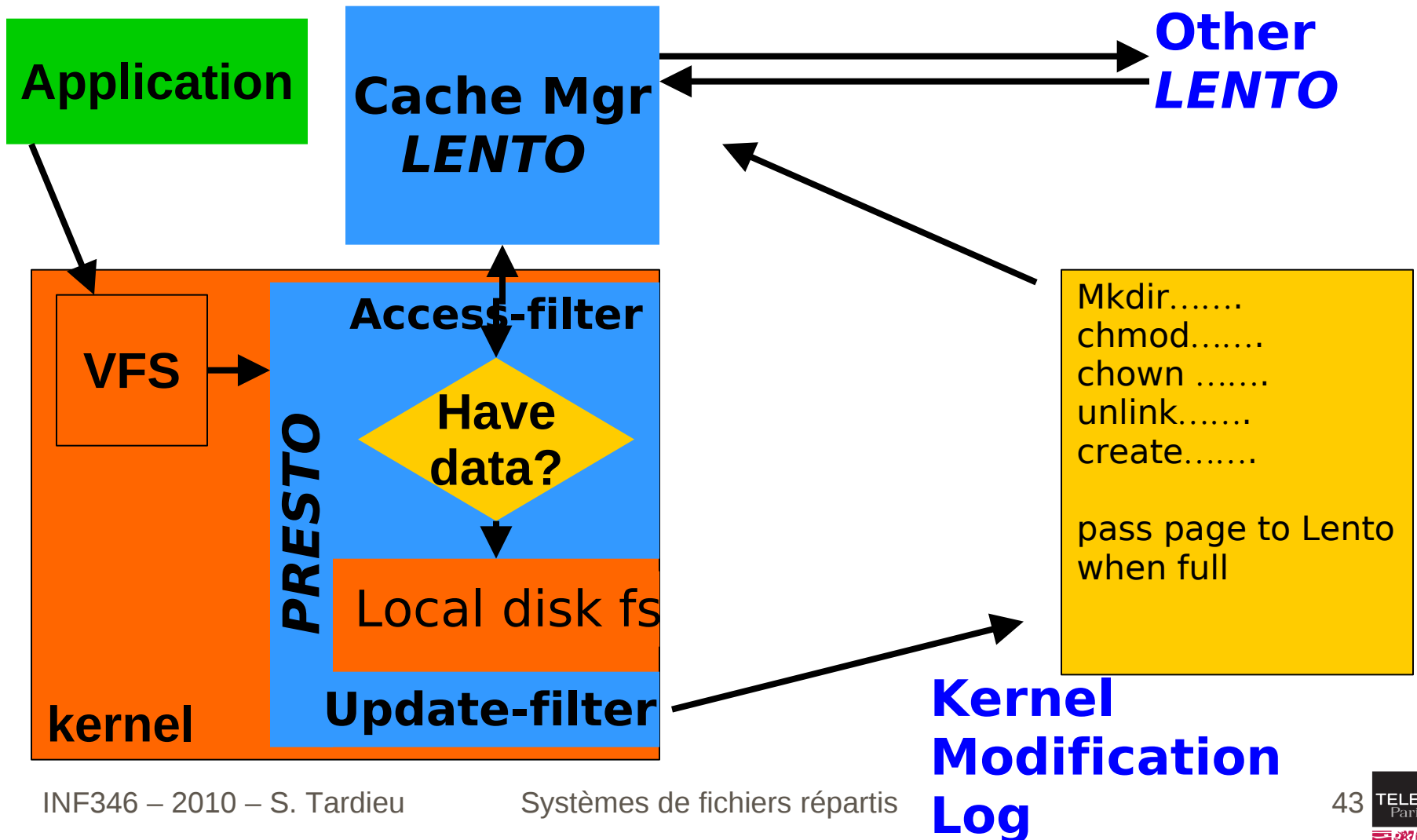
InterMezzo: Coda + robustesse

- Système de fichier au stade de prototype (et qui le restera)
- Utilise les mécanismes de Coda
 - pour la réintégration des données
 - pour les opérations en mode déconnecté
- Utilise le système de fichiers sous-jacent
 - vitesse d'accès aux données similaire à un accès local traditionnel
 - outils de réparation en cas de problèmes physiques ou de crash du système

InterMezzo: gestion des fichiers

- InterMezzo s'insère dans la gestion des fichiers entre l'utilisateur et le disque
 - module noyau sous Linux
 - service système sous Windows
- Principe de « permis de modifier »
 - un serveur négocie un permis avec les autres et l'accorde à un client
 - le client doit rendre ses modifications lorsqu'on lui retire le permis

InterMezzo: filtrage des requêtes



Use verifiable state machine approach & high level languages
Mimic future C++ implementation

Attack Windows & Unix in parallel

Existing work for security & bandwidth adaptation

Easily change and verify protocols & semantics

Performance centric

Exploit Local File Systems for cache and server

InterMezzo
Highly Versatile
Research Platform

InterMezzo: un modèle simple



- Module noyau pour Linux (Presto)
 - 2500 lignes de code C
 - Facile à intégrer dans le noyau, sources disponibles
- Programme Lento
 - 4000 lignes de Perl
 - Très hautement configurable
 - Très pratique pour tester de nouveaux prototypes
 - Sera réécrit plus tard (ou pas) en C++

Google Filesystem

- Besoins de Google
 - Stockage de fichiers de taille importante
 - Énorme volume de données
 - Rétention des données à long terme
 - Tolérance aux pannes des disques
 - Utilisation des données dans plusieurs datacenters répartis sur plusieurs continents (moteur de recherche)
 - Données proches de l'utilisateur (gmail)
 - Utilisation de matériel bon-marché, disponible partout et interchangeable

Google Filesystem

Architecture

- Découpage des fichiers en *chunks* de 64Mo
- Chaque chunk est répliqué
 - Au moins trois fois
 - Parfois plus pour augmenter la sécurité et la disponibilité
- Un maître, seul serveur à haute disponibilité, pour gérer le catalogue et les permissions
- Un très grand nombre de noeuds de stockage à faible coût unitaire, équipé de composants standard, avec un taux de panne important

Google Filesystem

■ Rôle du maître

- Connaît la localisation de chaque fichier
- Connaît l'état de chaque esclave
- Gère le nombre de réplicas et lance les demandes de réplication
- Donne les permissions d'écriture sur un chunk sous la forme d'un jeton qui expire

Google Filesystem

- Rôle de l'esclave
 - Stocke les chunks sur le disque et utilise les données
 - Obtient des permissions d'écriture temporaires
 - Se charge de répliquer les changements qu'il est autorisé à effectuer
 - S'assure que tous les replicas sont à jour avant de considérer l'écriture comme effectuée (atomicité)
 - Peut-être mis hors-ligne à tout moment, volontairement ou en cas de panne (disque, secteur, réseau, etc.)

Google Filesystem

- Haute disponibilité
 - La probabilité de perdre tous les réplicas au même moment est extrêmement faible
 - Le réplica le plus proche en terme de temps d'accès peut être utilisé pour la consultation
 - Le nombre de réplicas peut être augmenté pour augmenter la redondance et diminuer le temps d'accès
- Localisation et proximité
 - Gmail : les réplicas sont placés prioritairement près des utilisateurs qui en ont besoin

Google Filesystem



- Mise à disposition
 - Google a publié un article décrivant GoogleFS
 - Pas de mise à disposition en logiciel libre
 - Pas de commercialisation du système de fichier
- Alternative
 - Hadoop

Hadoop (HDFS)



- Similaire à GoogleFS
- Logiciel libre
- Écrit en Java
- Supporte plusieurs supports
 - Son propre système de fichiers
 - Serveurs FTP anonymes
 - Serveurs HTTP/HTTPS (lecture seule)
 - Amazon S3
 - ...

FUSE

- Filesystem in Userspace
- Permet de créer facilement un nouveau programme (C, Python, etc.) se comportant comme un système de fichier de haut-niveau
- Offre donc la possibilité de créer également un système de fichiers répartis sans avoir besoin de privilèges importants
- Disponible sur Linux, FreeBSD, OpenSolaris et MacOS X

FUSE



- Exemple de FS répartis avec FUSE
 - SSHFS : accès par SSH (protocole SFTP)
 - curlftpfs : accès par FTP
 - GMailFS : stockage de fichiers comme attachements dans la messagerie gmail
 - WikipediaFS : lire et éditer les articles de Wikipedia en éditant simplement des fichiers

Périphériques intelligents



- NASD (Network-Attached Secure Devices)
ou disques actifs
- Principes
 - Rendre les disques « intelligents »
 - Relier directement les clients et les données
- Défis
 - Conserver la sécurité des données
 - Utiliser l'intelligence des disques le plus intelligemment possible

NASD: les idées fondatrices



- Les permissions d'accès existent toujours
 - Un serveur vérifie l'identité du client
 - Ce serveur lui donne des jetons lui donnant certains droits
 - Le schéma d'authentification n'est pas figé
- Les transferts n'utilisent pas le serveur
 - Le client utilise ses jetons pour accéder aux données
 - Le client et le disque communiquent directement

NASD: sécurité des données



- Le serveur et NASD partagent un secret
- Le serveur renvoie au client des droits chiffrés avec ce secret
- NASD utilise ce secret pour déchiffrer les droits accordés au client
- Le serveur et NASD ne communiquent pas ensemble pour autoriser l'accès à un client
 - Economie de bande passante allant vers le disque

NASD: avantages

- Chemins de données plus courts
 - Les données sont échangées directement entre le disque et le client
 - Les transferts de périphérique à périphérique sont possibles sans intervention extérieure
- Possibilité d'optimisation intelligente
 - Répartition des données (performances)
 - Duplication des données pour résister aux pannes
 - Protection des données sensibles en cas d'attaque détectée

NASD: exemples d'utilisation



- Exploitation de données gigantesques
 - Recherche simultanée sur des milliers de disque
- Base de données à haute disponibilité
 - Des milliers de moteurs transactionnels fonctionnant en parallèle pour équilibrer la charge
- Protection automatique des données
 - Chaque disque sait quelles données doivent être répliquées, déplacées ou sauvegardées
- Les applications restent à inventer

Mais de quoi sera fait l'avenir?



- Si rien ne change
 - InterMezzo ou un système similaire se déploiera pour la collaboration
 - Des systèmes GoogleFS-like seront de plus en plus répandus pour les traitements
- Si la technologie évolue
 - Les disques deviendront intelligents
 - De nouvelles classes d'application apparaîtront
 - Ce point se combine avec le précédent
- Et si elle évolue encore plus
 - Le modèle du disque dur est-il viable?
 - De nouveaux moyens de stockage existent