Systèmes de fichiers répartis

INF346

Samuel Tardieu
sam@rfc1149.net

Télécom ParisTech

16 mars 2012

Pourquoi partager des fichiers ?

- Travail en équipe (-)
  - Projets à plusieurs développeurs
  - Mise à jour d'une base de données
- Travail personnel (+)
  - Utilisation de machines banalisées
  - Utilisation de machines différentes
    - Ordinateur fixe avec un large disque et copie de sauvegarde
    - Ordinateur portable
- Accès optimisé aux données (+)
  - Répartition des données sur plusieurs machines (sharding)
  - Redondance des informations

Ce problème n'est pas récent

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

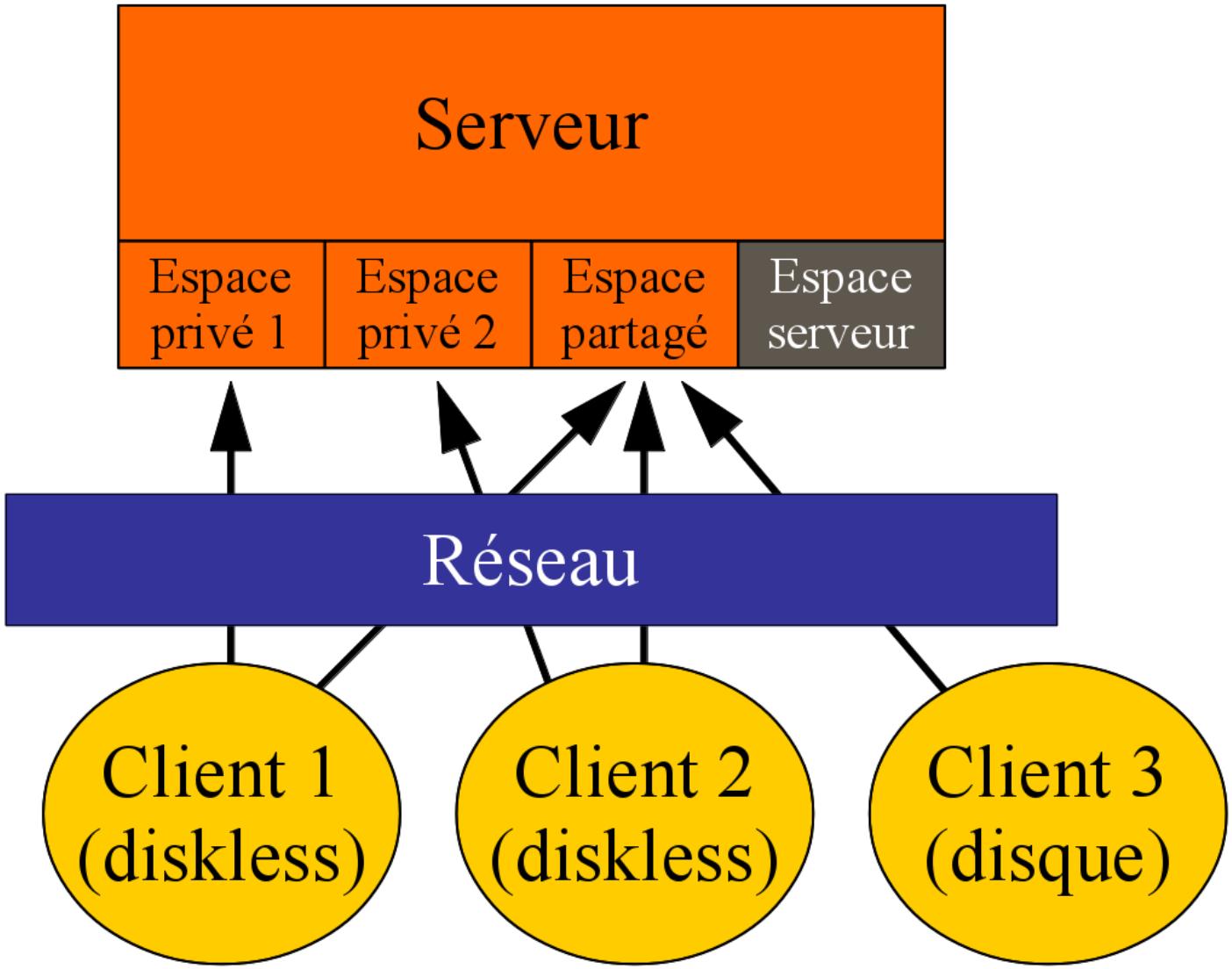
NFS, avec quelques améliorations entre temps, est encore utilisé aujourd'hui.

Les besoins ont évolué

- Les développements se font à travers un système de gestion de version (VCS, *version control system*).
- Tous les VCS supportent un accès à distance.
- Les fichiers de l'utilisateur peuvent être eux aussi versionnés.
- Les réseaux sont de plus en plus rapides.

Plan du cours

NFS : initialement pour les clients sans disque



NFS : système basique au départ

- Prévu pour ressembler au système de fichiers Unix
- Utilise les RPC de Sun
  - Données codées selon le protocole XDR
  - Utilisation d'un service de *portmapping*
- Pas d'état stocké sur le serveur
  - Plus facile à mettre en œuvre
  - Plus difficile à optimiser
- Pas ou peu de cache utilisé par le client
  - Le cache n'est utile que pour les gros fichiers
  - La sémantique du cache est douteuse sans état sur le serveur

NFS : disponibilité

- Conçu à l'origine par Sun Microsystems
- Disponible pour tous les Unix
- Disponible pour Windows et macOS

NFS : opérations

- Opérations en lecture : `getattr`, `lookup`, `readdir`, `read`, `readlink`, `statfs`, `mount`, ...
- Modifications de la hiérarchie de fichiers : `mkdir`, `link`, `symlink`, `rename`, `unlink`, `rmdir`, ...
- Modification des fichiers : `setattr`, `write`

Les opérations ont été choisies pour ressembler à celles d'un système de fichier local.

NFS : support des incidents

NFS gère le scénario suivant :

- Le client charge un fichier dans son éditeur.
  - Le fichier est demandé au serveur et récupéré à travers ses identifiants.
- La connexion au serveur est interrompue, ou le serveur redémarre.
- 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 identifiants restent valables.
  - Par contre, il y aura un problème si la configuration du serveur change (*NFS stale file handle*).

NFS : sécurité

- Support de Kerberos, très contraignant
- Authentification basée sur les adresses IP
- La sécurité implique l'intégrité du réseau et celle des administrateurs

CIFS : 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 utilsiateur
- Disponible sous Unix (samba)
- Permet de partager également les ressources (imprimantes par exemple)

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*
  - Possibilité pour le client d'utiliser sa copie locale, supposée valide
  - *Callback* appelé par le serveur en cas d'invalidation

  Le serveur maintient une connaissance de l'état des caches des clients.

AFS : propriétés du cache

- Cache persistant
  - Résiste au redémarrage de la machine
  - Stocke un numéro de version associé à chaque donnée
- Accès concurrents
  - Accès en lecture : ne nécessite aucune opération réseau une fois le fichier dans le cache
  - Accès en écriture : provoque une demande d'invalidation de cache par le serveur à tous les clients pour les données modifiées

AFS : organisation

- Espace de nommage unique : `/afs`
- Domaine administratif : la cellule
  - Un ou plusieurs serveurs
  - Zéro ou plusieurs clients accédant aux données
  - Chaque cellule exporte des volume montés sous `/afs`
- Administration du client minimale
  - Le client contactera de lui-même la cellule pour trouver un serveur offrant le volume demandé

AFS : réplication

La stratégie de réplication d'AFS est simple et pessimiste :

- Un seul serveur autorise les écritures.
- Tous les autres agissent comme des esclaves fournissant les fichiers en lecture seule.

AFS : protocole

- Du client vers le serveur
  - Lecture et écriture des données
  - Modification de la hiérarchie de fichiers
  - Gestion des volumes
  - Vivacité et sémantique
  - Verrouillage
- Du serveur vers le client
  - Invalidation du cache
- Avec de la sécurité
  - Gestion de jetons par Kerberos
  - ACL (*Access Control List*) fines par répertoire

AFS ne cherche pas à imiter un système de fichiers local.

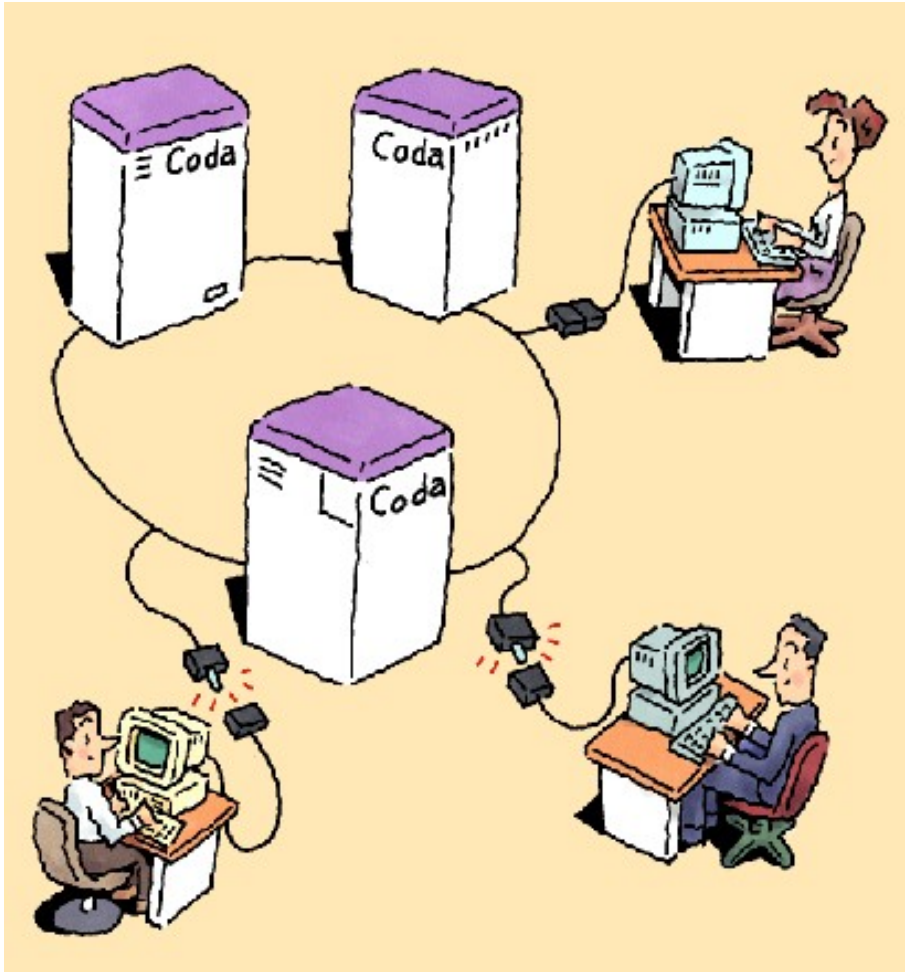
AFS : disponibilité

- Premières implémentations développées par CMU (Carnegie Mellon University)
- Implémentation commerciale développée par TransArc, achetée par IBM, et libérée sous le nom d'OpenAFS
- Implémentation libre : Arla
- Implémentation partielle par RedHat dans Linux depuis le noyau 2.6.10
- N'a jamais vraiment pris sauf dans certaines universités américaines (notamment CMU)

Coda : le successeur d'AFS 2

- Développé (encore) à CMU
- Stratégie de réplication optimiste (ou paresseuse)
- Plusieurs serveurs en lecture / écriture
- Cache local avec possibilité de préchargement (*hoarding*)
- Support des opérations en mode déconnecté
- Support des partitions du réseau de serveurs
- Adaptation à la bande passante disponible
  - Priorisation des fichiers échangés
  - Réconciliation en tâche de fond des changements non requis immédiatement

Coda en une image



(Illustration de Gaich Muramatsu, `http://www.coda.cs.cmu.edu/`)

Coda : réintégration des changements

- Le client enregistre chaque changement fait dans un fichier de modifications (*log*) sauf s'il est connecté par une liaison rapide (mode synchrone).
- Les changements successifs sont compactés.
- Il l'envoie au serveur lorsqu'il se connecte.
- Le serveur lui demande de lui envoyer les modifications effectuées dans chaque fichier modifié (*backfetch*).
- Le serveur gère les conflits provoqués par les modifications concurrentes.

Coda : gestion des conflits de clients

- Dans les cas 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 effectutent les mêmes opérations.
- Dans le cas d'écritures asynchrones :
  - Coda essaie de résoudre les faux conflits.
  - L'utilisateur doit résoudre les autres conflits à la main.
  - Coda ne prend jamais de décision sur une stratégie de réconciliation de changements concurrents.

Coda : réplication des serveurs

- Modèle « *write all, read one* »
- VSG (*Volume Storage Group*) pour chaque volume
- VS (*Version stamp*) pour chaque répertoire
- VS comparés par un client auprès des serveurs atteignables avant de faire une écriture
- En cas d'absence de serveurs, fichier de *log* stocké sur les serveurs joignables pour préparer la réconciliation

Coda : gestion des conflits de serveurs

- Résolution basique
  - Identification éventuelle d'un réplica dominant
  - Permet surtout d'importer un objet sur un réplica qui ne le connaît pas
- Résolution complexe
  - Protocole en 5 étapes
  - Élection d'un coordinateur pour mener la réconciliation
  - Demande d'aide à l'utilisateur en cas de conflits non solubles automatiquement

Coda : disponibilité

- Peu populaire notamment en raison du système de fichiers dédié
- Développement irrégulier mais toujours en cours
- Focus mis sur la robustesse de l'implémentation pour en faire un produit commercial

Intermezzo : l'expérimentation ouverte

- Inspiré de Coda
- Travaille au dessus des systèmes de fichiers existants
- Développé à... CMU
- Séparation du protocole en deux composants
  - Presto, dans le noyau, fait le lien entre le système de cache local et le composant de synchronisation Lento.
  - Lento, hors du noyau, s'occupe des opérations de synchronisation avec les autres réplicas.
- Lento peut être écrit dans n'importe quel langage (première version en Perl).

NFS : une norme qui évolue

- La norme évolue
  - NFS v4 : décembre 2000, avril 2003, influencé par AFS et CIFS
  - NFS v4.1 : janvier 2012
- Efficacité
  - Regroupement des requêtes
  - Système de cache avec invalidation possible
  - TCP privilégie pour détecter les ruptures de connexion
  - Délégation à des serveurs secondaires
  - Accès aux fichiers en parallèle (*striping*) et notion de session
- Sécurité
  - Simplification du modèle par rapport aux versions précédentes
  - Possibilité de chiffrement
  - Possibilité d'imposer des contraintes sur le niveau de sécurité requis, côté client et côté serveur

GoogleFS : les besoins

- 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

GoogleFS : 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

GoogleFS : 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

GoogleFS : 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.)

GoogleFS : 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 : les réplicas sont placés prioritairement près des utilisateurs qui en ont besoin.

GoogleFS : map / reduce

- Accès aux données : fichiers potentiellement de taille énorme
- Algorithmes de type map / reduce qui travaillent sur des portions de données et rassemblent les résultats
- Possibilité de travailler sur des fichiers locaux aux machines qui effectuent les opérations

GoogleFS : disponibilité

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

HDFS : un GoogleFS libre

- Écrit en Java
- Disponible sur plusieurs supports :
  - son propre système de fichiers
  - serveurs FTP anonymes
  - serveurs HTTP / HTTPS (lecture seule)
  - Amazon S3
  - ...
- Point de faute unique (serveur de nom), sauf dans certaines versions propriétaires

FUSE : systèmes de fichiers pour tous

- *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 : exemples

- 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

NASD : périphériques intelligents

- *Network-attached secure devices*, ou disque actifs
- Principes
  - Rendre les disques « intelligents »
  - Relier directement les clients et les données
- Défis
  - Conserver la sécurité des données
  - Laisser les disques prendre des décisions cruciales pour l'intégrité des données

NASD : idées fondatrices

- Permissions d'accès
  - Un serveur vérifie l'identité du client.
  - Le serveur donne au client des jetons lui donnant certains droits.
  - Le client transmet les jetons au disque, qui peut les vérifier auprès du serveur.
- Transferts directs entre le client et le disque sans intervention du serveur
- Échange possible d'information de vérification de droit d'accès entre disques se faisant confiance

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
- Sécurité automatique des données : un disque attaqué peut le signaler aux autres, demander des copies supplémentaires de ses données et se déclarer corrompu

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
- Sécurité automatique des données : un disque attaqué peut le signaler aux autres, demander des copies supplémentaires de ses données et se déclarer corrompu
- Mais ce modèle n'a jamais pris !

Mais encore. . .

- Amazon S3 : accessible à travers des interfaces REST, SOAP ou BitTorrent
- Dropbox : service de stockage par Internet
  - support des OS mobiles
  - officiellement confidentiel
  - ouvertement non confidentiel
- Bitcasa : actuellement en phase de test, disponible pour Windows et MacOS uniquement aujourd'hui, stockage réparti et officiellement sécurisé

Conclusion

- Les systèmes de fichiers répartis sont utilisés depuis 25 ans.
- Les usages changent :
  - du réseau local à l'Internet, puis au cloud
  - du modèle de confiance au modèle sécurisé
  - du centralisé au réparti
  - de l'unique au répliqué
- Certains modèles ont été abandonnés avant même d'être répandus (CODA / Intermezzo, NASD)
- Le modèle du disque dur est-il pérenne ?

Conclusion

- Les systèmes de fichiers répartis sont utilisés depuis 25 ans.
- Les usages changent :
  - du réseau local à l'Internet, puis au cloud
  - du modèle de confiance au modèle sécurisé
  - du centralisé au réparti
  - de l'unique au répliqué
- Certains modèles ont été abandonnés avant même d'être répandus (CODA / Intermezzo, NASD)
- Le modèle du disque dur est-il pérenne ?
- Ce sera probablement à vous d'inventer les nouveaux usages !