Gestion répartie de données - 1

Sacha Krakowiak
Université Joseph Fourier
Projet Sardes (INRIA et IMAG-LSR)

http://sardes.inrialpes.fr/~krakowia

Gestion répartie de données

- **Plan de la présentation**
  - Introduction
    - Besoins et domaine couvert
    - Données technologiques
  - Principes de la gestion répartie de données
  - Systèmes répartis de gestion de fichiers
    - Systèmes client-serveur
    - Systèmes pair à pair
  - Mémoire virtuelle répartie
  - Mode déconnecté

Gestion répartie de données : besoins (1)

Accès à un ensemble de données réparties sur un ensemble de machines, avec tout ou partie des propriétés suivantes

- **Partage**
  - Les données sont partagées entre un ensemble d'utilisateurs
  - Les droits d'accès (lecture, écriture, etc.) diffèrent selon les utilisateurs
- **"Transparence"**
  - Localisation : la désignation d'une donnée est indépendante de sa localisation physique
  - Accès : la procédure d'accès à une donnée distante est la même que pour une donnée locale
- **Persistance**
  - Une donnée est persistante si elle survit au processus qui l'a créée (sinon, elle est temporaire)

Gestion répartie de données : besoins (2)

- **Intégrité**
  - Pas de modification de données autrement que par une opération explicite (et autorisée). En particulier, pas de modification ou de destruction de données lors d'une défaillance matérielle ou logicielle
- **Cohérence**
  - On impose des contraintes (dépendant de l'application) sur la cohérence interne des données. Exemple :
    - Intégrité en présence d'opérations concurrentes (lecteurs-rédacteurs, etc.)
- **Efficacité d'accès**
  - On cherche à améliorer les performances d'accès aux données distantes. Deux mesures :
    - débit : quantité d'information transmise par unité de temps (bit/s)
    - latence : délai avant l'accès au premier bit (s)

Influence de la technologie sur l'accès aux données

- **Évolution de la technologie des disques**
  - Capacité : 70 GB sur 3,5", 25 GB sur 2,5", 0,34 GB sur 1"
  - Débit : 180 MB/s
    - progrès rapides sur capacité et débit
  - Latence : <10 μs - reste la principale limitation
  - Tolérance aux fautes : RAID
- **Connexion directe des disques au réseau**
  - Gain de latence globale
- **Évolution des réseaux**
  - Réseaux à haut débit et faible latence
    - Répartition du stockage et des caches
  - Réseaux sans fil, mobiles
    - Mode déconnecté

Gestion répartie de données : domaine

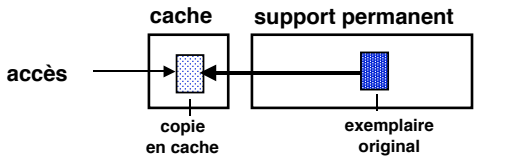
La gestion répartie de données est présente sous des formes très variées

- **Systèmes répartis de gestion de fichiers**
- **Mémoire virtuelle répartie**
- **Accès aux données sur le Web**
- **Bases de données réparties**

Tous ces systèmes utilisent quelques techniques communes

Désignation répartie
Gestion de caches, données dupliquées
Gestion de cohérence
Gestion de la persistance

Gestion de caches (1)



	cache	support permanent
tps accès	-	+
taille	-	+
coût/bit	+	-

Motivation : localité d'accès
Les accès récents sont souvent un bon prédicteur des accès dans un futur proche

Performances (pour données de taille fixe)

$$t = p_c \cdot tc + (1 - p_c) \cdot ts$$

- t : temps d'accès
- p_c : probabilité de succès
- tc : temps d'accès cache
- ts : temps d'accès support permanent

Typiquement (cache du processeur), $p_c > 90\%$

Gestion de caches (2)

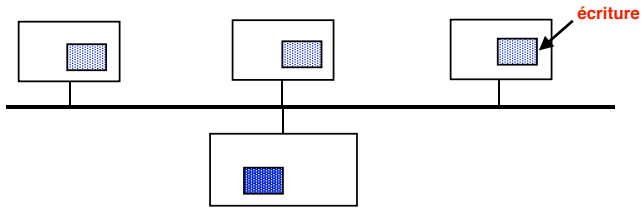
Choix de conception

- taille des données
  - fixe (unités physiques)
  - variable (unités logiques)
    - gestion d'espace complexe
- propagation des modifications
  - immédiate (*write through*)
  - différée (*write back*)
- algorithme de chargement
  - à la demande
  - préchargement (critère ?)
- algorithme de remplacement
  - LRU : le plus utilisé (localité)
  - autres : LFU, FIFO, etc.
  - vidage anticipé (critère ?)

Exemples

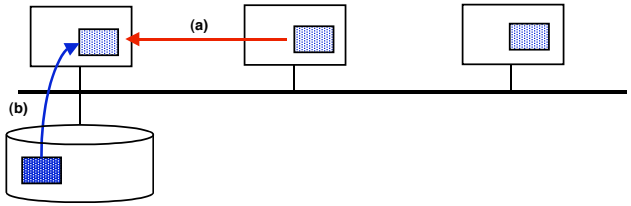
- caches du processeur
  - multiniveaux (L1, L2)
- caches de fichiers
  - côté client ou côté serveur
- caches de noms
  - résolution des noms dans DNS
- caches pour le Web
  - côté client ou côté serveur

Caches multiples : problèmes



- **Problèmes**
  - Cohérence mutuelle
    - entre les différents caches (si modification d'une copie en cache)
    - entre les copies en caches et l'exemplaire original (si écriture immédiate)
- **Principes de solutions**
  - Invalidation des copies non modifiées
  - Propagation des modifications

Caches multiples : avantages potentiels

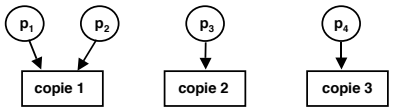


- **Caches coopératifs**
  - Si les réseau est assez rapide (faible latence), il peut être plus avantageux de charger une donnée à partir d'un autre cache (a) qu'à partir du disque local (b)
  - Nécessite une gestion soigneuse pour
    - éviter les incohérences
    - garantir les gains de performances

Partage de données

Le partage de données entre processus (éventuellement répartis) soulève la question de la "sémantique" du partage vis-à-vis des modifications des données partagées.

Le problème se pose parce que les processus qui partagent une donnée peuvent accéder à des copies différentes (par exemple caches multiples).



Question : à quel moment une modification faite par un processus est-elle rendue visible aux autres processus?

Partage de données : "sémantique" du partage

- **Visibilité immédiate**
  - Toute lecture "voit" l'état résultant de la dernière modification effectuée
  - Nécessite de savoir définir "dernière" (non trivial en réparti) et de mettre en œuvre les mécanisme nécessaires (estampilles)
- **Session**
  - Tout accès à des données partagées est réalisé à l'intérieur d'une "session" (nécessite de définir début et fin de session)
  - Pendant une session, un processus ne voit pas les modifications faites par les autres processus sur les objets qu'il utilise
  - Ces modifications sont rendues visibles lors de la session suivante
- **Transaction**
  - Tout accès à des données partagées est réalisé à l'intérieur d'une transaction
  - Méthode sûre, mais coûteuse

Systèmes répartis de gestion de fichiers : un peu d'histoire (très simplifiée)

- Origines : projets de recherche
  - 1974 : Newcastle Connection : espace unifié de fichiers, super-racine ; couche d'interposition sous Unix
  - 1979 : Locus : système unique de fichiers, mais système réécrit
- Produits
  - 1984 : NFS, initialement Sun, standard *de facto* aujourd'hui
  - 1988 : AFS / DFS, initialement projet de recherche, puis produit
- Avancées techniques (matériel)
  - 1988 : disques RAID
  - actuel : disques sur réseau (SAN, NAS)
- Recherches
  - 1990 : Coda : mode déconnecté
  - 1991 : Log-structured File System (LFS) : fichiers journalisés
  - 1994 : xFS : RAID réparti sur réseau, + LFS + caches coopératifs
  - 1996 : Petal : "disques virtuels" sur réseau
- Actualité
  - systèmes "pair à pair" (Napster, Gnutella, Groove)

Systèmes répartis de gestion de fichiers

- Données statistiques (sur Unix)
  - La plupart des fichiers sont petits (<10 Ko) - mais quelques uns très gros (Go)
  - La plupart des accès sont des lectures (6 pour 1 écriture)
  - L'accès aux fichiers est en général séquentiel ; l'accès aléatoire est exceptionnel
  - La plupart des fichiers sont utilisés par un seul usager
  - La plupart des fichiers partagés sont modifiés par un seul usager
  - Localité : un fichier récemment utilisé a de fortes chances d'être réutilisé
  - Beaucoup de fichiers ont une durée de vie courte ; mais ceux qui survivent subsistent longtemps
  - La fonction de communication du SGF est au moins aussi importante que celle de stockage

Caractéristiques des systèmes répartis de gestion de fichiers

Ces caractéristiques servent de grille d'analyse pour les études de cas

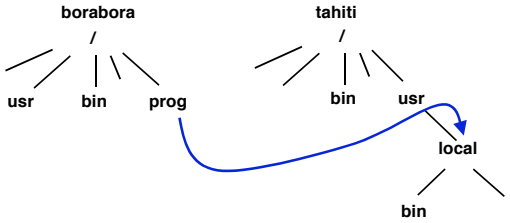
- Structure de l'espace de noms
  - uniforme, composé par montage, composé par partage
- Avec ou sans état
  - le serveur peut conserver ou non l'état courant de la session
- Sémantique du partage
  - "Unix", session, transaction
- Caches et gestion de la cohérence
  - caches clients, caches serveurs, invalidation, écriture immédiate ou différée
- Tolérance aux fautes, disponibilité
  - duplication, reprise

Étude de cas : NFS

- Histoire
  - Initialement, SGF réparti pour Unix (SunOS)
  - Devient un standard indépendant du système
- Principes de base
  - Espace de noms composite, construit par montage
  - Toute station peut être client ou serveur
  - Serveurs sans état
  - Gestion de caches : vise à préserver la sémantique Unix
    - mais n'y parvient qu'approximativement
  - Disponibilité
    - duplication non prévue (doit être réalisée séparément)
- Situation
  - Système d'usage très courant sur réseaux locaux
  - Bonnes performances, mais capacité de croissance limitée

NFS : désignation

Utilise le montage, technique initialement introduite dans Unix pour insérer des volumes amovibles dans le SGF.



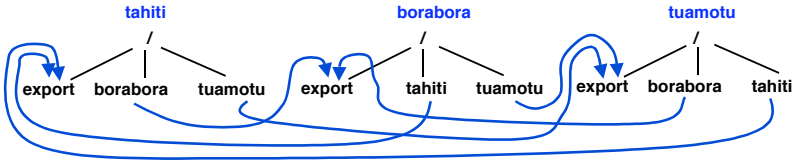
Primitive *mount*
Utilisation restreinte :
- la machine cible doit "exporter" les répertoires à monter
- l'opération *mount* nécessite les droits de l'administrateur (*root*)

Deux sortes de montage
- *hard mount* : une faute d'accès bloque le client
- *soft mount* : une faute d'accès est non bloquante (renvoie message d'erreur)

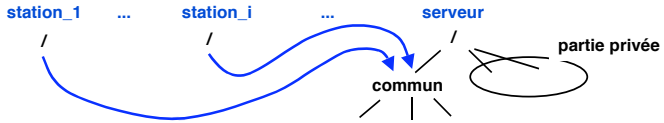
NFS : désignation

La gestion de l'espace de noms peut très vite devenir complexe. Il faut fixer des règles pour le montage.

Exemple 1 : Espace de noms uniforme sur toutes les machines (vision commune)

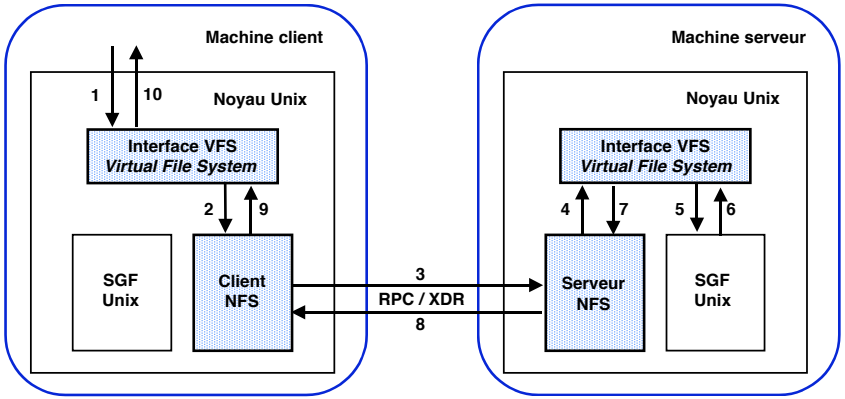


Exemple 2 : Stations sans disque montées sur un serveur



NFS : architecture d'ensemble

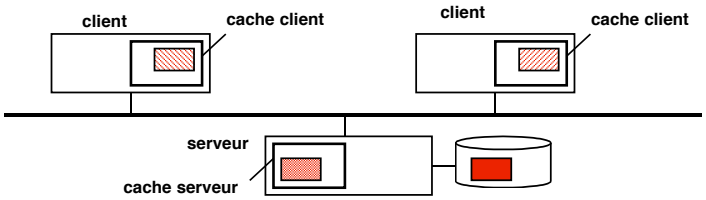
Les modules NFS sont inclus dans le noyau, sur chaque machine
Interface VFS : *vnodes* (*inodes* + informations sur fichiers distants)



NFS : serveurs sans état

- **Principe**
  - Un serveur ne conserve aucune donnée permanente sauf
    - le contenu des fichiers
    - des indicateurs (accélérateurs d'accès)
  - En particulier : aucune donnée conservée sur l'état d'une session en cours
- **Conséquences**
  - Un serveur peut tomber en panne et redémarrer sans causer de perturbation
  - Tous les échanges RPC client-serveur doivent être idempotents
    - Une requête spécifie toujours un déplacement absolu
  - Un serveur ne maintient pas la liste des fichiers ouverts
    - Une requête spécifie toujours le nom du fichier
    - Les droits sont vérifiés à chaque accès

NFS : gestion des caches (1)



- Données en cache : blocs de données, méta-données (attributs)
- Cache serveur
  - Évite des accès au disque
  - Conserve résultats des dernières opérations (*read*, *write*, *dir-...*)
  - Écriture immédiate (toute modification est immédiatement reportée sur disque)
    - avantage : serveur sans état (pas de perte d'information en cas de panne)
    - inconvénient : performances (le client doit attendre que l'écriture soit faite)

NFS : gestion des caches (2)

- Cache client
  - Conserve résultat des *read*, *write*, *dir-...*
  - Problème : cohérence de caches
    - entre les différents caches clients
    - entre cache client et cache serveur
  - Solution
    - Le serveur conserve pour chaque fichier la date de la dernière modification (soit T)
    - Le client conserve pour chaque fichier la date du dernier accès aux données en cache (soit t)
    - Si T > t, les données du fichier sont invalidées dans le cache client (doivent être rechargées)
    - Ce contrôle est fait
      - à chaque ouverture et à chaque accès
      - au moins toutes les 3 secondes (30 secondes pour un répertoire)

NFS : gestion des caches (3)

- Critique de la solution
  - Cache serveur : OK (sauf coût des écritures)
  - Cache client
    - La "sémantique" n'est pas définie de manière rigoureuse
    - En principe : sémantique Unix (une lecture voit le résultat de la dernière écriture)
    - En fait : incohérence possible si modifications rapprochées (< 3 secondes)
    - Nécessite synchronisation précise des horloges
  - En pratique
    - Solution acceptable dans les situations usuelles

NFS : performances

- Performances généralement bonnes
- Principaux problèmes
  - Écriture immédiate (sur disque) dans le cache serveur
    - Solution possible : cache disque matériel (RAM) tolérant aux fautes (alimentation de secours par batterie)
  - Vérification fréquente des dates de modification (communication client-serveur)
    - Nécessaire pour cohérence
  - Recherche du nom composant par composant dans arborescence de catalogues
    - Nécessaire pour conserver sémantique Unix

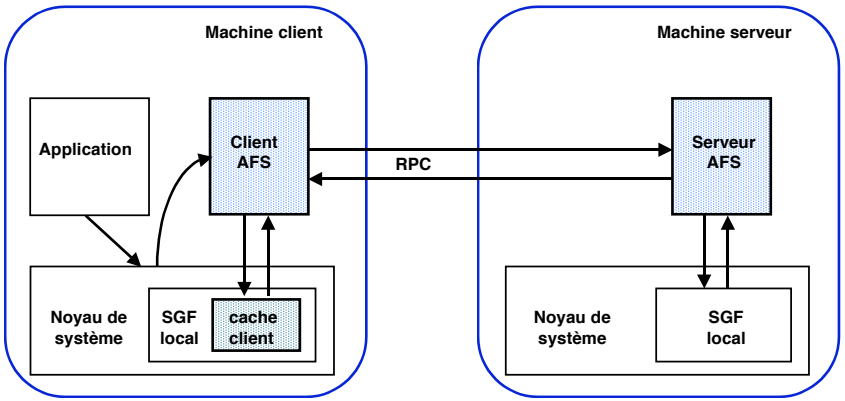
Étude de cas : AFS

- **Histoire**
  - Initialement, projet de recherche (*Andrew File System*, Univ. Carnegie Mellon + IBM) sur fédération de réseaux locaux
  - Devient un produit (Transarc, puis IBM), puis un standard (DFS, Distributed File System) dans OSF DCE
- **Principes de base**
  - Espace de noms partiellement partagé
  - Séparation clients - serveurs
  - Gestion de caches : sémantique de session
  - Disponibilité : intégrée (duplication de serveurs)
- **Situation**
  - Utilisé comme base commune sur réseaux grande distance
    - mises à jour peu fréquentes
  - Performances acceptables, mais disponibilité difficile à garantir sur réseau à grande distance
  - Bonne capacité de croissance

AFS : caractéristiques générales

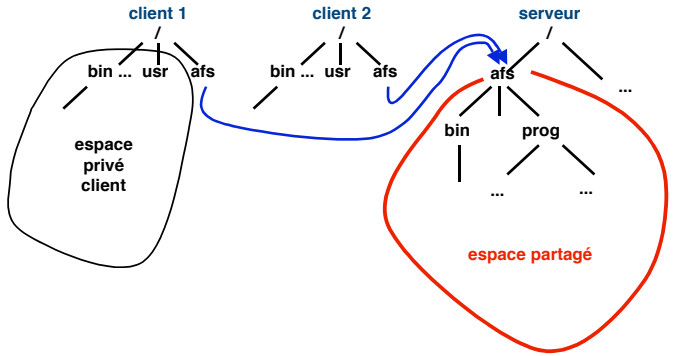
- **Unité de transfert**
  - Fichier entier (ou tranche de grande taille pour très gros fichiers)
- **Cache client**
  - Cache client sur disque du client
  - Cache client de grande taille (1 Go)
  - Mise en cache de fichiers entiers (ou tranche de grande taille)

AFS : architecture d'ensemble



AFS : désignation

Les clients partagent un espace commun géré par le serveur
L'accès à cet espace est protégé (équivalent d'un login, avec mot de passe)
Chaque client conserve un espace privé



AFS : gestion des caches (1)

- **Principe : notion de session**
  - Session (pour un fichier) : période pendant laquelle un client détient dans son cache (sur disque) une copie du fichier
  - Accès à un fichier *f* par un client *C*
    - si *C* a une copie de *f* en cache, il utilise cette copie
    - sinon, il ouvre une session pour *f*
      - demande *f* au serveur qui le détient (voir plus loin comment le trouver)
      - le serveur envoie à *C* une copie de *f* (et un témoin de rappel, voir plus loin)
      - le serveur note un témoin pour *C* et *f*
    - le client vérifie la présence du témoin à chaque ouverture
  - Fin d'utilisation (fermeture) d'un fichier *f* par un client *C*
    - si la copie en cache a été modifiée, elle est renvoyée au serveur

AFS : gestion des caches (2)

- **La cohérence est gérée par le serveur**
  - Quand un client a un fichier en cache (sur disque), le serveur s'engage à lui signaler tout risque d'incohérence sur ce fichier
  - Cet engagement est représenté par un témoin de rappel (*callback promise*)
    - conservé par le serveur (pour client *C*, sur fichier *f*)
    - communiqué au client avec la copie de *f*
  - Si *f* est modifié sur le serveur (fermeture par un autre client *C1*), le serveur rappelle *C* : "votre copie de *f* est périmée", et efface le témoin de rappel
  - La réaction de *C* dépend de l'application
    - ne rien faire (et continuer d'utiliser la copie périmée)
    - demander une nouvelle copie

AFS : gestion des caches (3)

- **Sémantique du partage : "session"**
  - À l'ouverture, le client reçoit la version courante (connue du serveur)
  - À la fermeture, une nouvelle version est créée (si modifs)
  - Les clients courants sont prévenus des mise à jour
  - Attention : ce mécanisme ne garantit rien en cas d'écritures concurrentes
  - Si on veut des garanties, il faut utiliser le verrouillage (par ex. lecteurs - rédacteurs) ; les verrous sont gérés sur le serveur
- **En cas de redémarrage après panne…**
  - … du client : effacer tous les témoins (donc invalider tout le contenu du cache)
  - … du serveur : restaurer les témoins (qui doivent être gardés en mémoire stable)

AFS : gestion interne des fichiers

- **Les fichiers sont stockés sur un ou plusieurs serveurs**
  - Espace de fichiers "plat" en interne, organisé en "volumes" (unités logiques de stockage)
  - Identification interne de fichier : *uid* de 96 bits
  - Une organisation hiérarchique est réalisée par logiciel (correspondance nom - *uid*)
  - Correspondance entre *uid* et serveur conservée dans une table, dupliquée sur tous les serveurs (change rarement)
- **Disponibilité**
  - Un volume peut être dupliqué sur plusieurs serveurs
    - assure disponibilité des données (si serveur accessible, peut être difficile sur réseau longue distance)
    - permet d'améliorer les performances (choix d'un serveur proche ou peu chargé)

AFS : performances

- **Performances d'accès généralement bonnes**
  - Dépendance limitée par rapport aux communications, si écritures peu fréquentes (distribution de logiciel, documentation) : cas privilégié d'utilisation
  - Problèmes possibles si écritures fréquentes et réseau à grande distance
  - Facteurs favorables : transferts de fichiers entiers, témoins de rappel
- **Bonne capacité de croissance**
  - Séparation clients-serveurs
  - Croissance incrémentale par adjonction de serveurs
  - Duplication possible (en pratique si écritures peu fréquentes)

Mode déconnecté

- **Motivations**
  - Généralisation de l'usage des portables
  - Développement des communications sans fil
  - La plupart du temps, les portables sont utilisés
    - soit en mode autonome
    - soit connectés à un réseau
  - On cherche à assurer une transition facile entre les deux modes
- **Principe**
  - Idée de départ : mécanisme AFS (mise en cache de fichiers entiers sur le poste client)
  - Problèmes
    - choix des fichiers à conserver lors de la déconnexion
    - mise en cohérence à la reconnexion

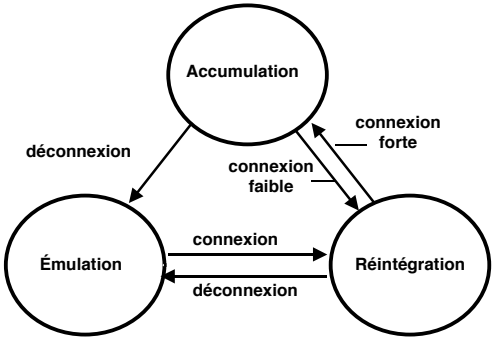
Exemple de système en mode déconnecté : Coda

- **Historique**
  - Projet de recherche à Carnegie Mellon Univ. (suite d'Andrew, 1990-...) - Coda = *Constant Data Availability*
  - Actuellement : intégré dans Linux
- **Objectifs**
  - Viser une disponibilité permanente de l'information chez le client, y compris dans les cas suivants
    - déconnexion volontaire
    - déconnexion accidentelle
    - faible connectivité
      - connexion intermittente (communications mobiles)
      - connexion à très faible débit
- **Principes**
  - Duplication des serveurs
  - Mode déconnecté

Coda : principes de conception

- **Extension d'AFS**
  - séparation client-serveur
  - cache client sur disque
  - cache de fichiers entiers
- **Duplication des serveurs AFS**
  - Un volume peut avoir des copies sur plusieurs serveurs
  - Augmente la probabilité qu'un volume soit accessible
- **Modification du client AFS**
  - Pendant la connexion, cherche à stocker en cache les fichiers potentiellement utiles
  - En mode déconnecté, utilise les fichiers en cache
  - À la reconnexion, processus de réintégration
    - automatiser autant que possible
    - stratégie "optimiste"

Coda : états du client



- Accumulation : mode pleinement connecté ; le client collecte les fichiers à conserver
- Émulation : mode déconnecté ; le client utilise les fichiers en cache
- Réintégration : état intermédiaire ; le client réintègre les modifications effectuées durant la déconnexion

Coda : mode accumulation (1)

- **Ouverture d'un fichier**
  - Accès à un fichier *f* d'un volume *V* : déterminer l'ensemble des serveurs (*Volume Storage Group*, VSG) stockant une copie de *V*
  - Parmi ces serveurs, déterminer ceux qui sont accessibles (*Available VSG*, AVSG), et en choisir un parmi ceux-ci ("serveur favori")
  - Charger une copie du fichier (avec témoin de rappel sur le serveur)
- **Fermeture d'un fichier**
  - Si modifié, renvoyer la copie à tous les serveurs de l'AVSG
  - Si AVSG ≠ VSG (certains serveurs non accessibles), protocole de remise à jour entre serveurs
  - Si d'autres témoins de rappel présents, le serveur favori exécute les rappels

Coda : mode accumulation (2)

- **Détection d'incohérence**
  - Si tous les serveurs accessibles n'ont pas la même version du fichier (détection par estampilles)
    - le client charge la plus récente
    - le client déclenche un protocole de mise à niveau (exécuté par les serveurs) - intervention manuelle si nécessaire
- **Chargement anticipé des fichiers**
  - En mode accumulation, le client charge des fichiers en tâche de fond pour préparer période de déconnexion
    - selon une liste de préférences fournie par l'utilisateur
    - selon un algorithme d'apprentissage (observation des accès)

Coda : mode accumulation (3)

- **Le client surveille la disponibilité des serveurs**
  - Toutes les T unités de temps (de l'ordre de 10 minutes), le client examine le VSG des fichiers qu'il a en cache
  - Si variation de l'AVSG (panne ou réinsertion d'un serveur), mise à jour éventuelle des témoins de rappel chez le client (invalidation des copies en cache)
    - réinsertion d'un serveur : invalider les fichiers dont ce serveur a une copie (car sa version est peut-être plus récente)
    - disparition d'un serveur "favori" pour un volume : invalider les fichiers de ce volume
    - disparition d'un autre serveur : rien à faire

Coda : mode émulation

- En mode émulation (déconnecté) le client utilise les fichiers en cache local
  - Les modifications sont journalisées sur le disque pour être rejouées ultérieurement
  - Erreur si défaut de fichier dans le cache (en principe rare)

Coda : mode réintégration (2)

- Restauration de la cohérence
  - Les modifications effectuées pendant la déconnexion sont “rejouées” et propagées vers les serveurs
  - Si la connectivité est faible, propagation “goutte à goutte” en tâche de fond pour ne pas dégrader les performances
- Résolution des conflits
  - Si un conflit est détecté (modifications pendant la déconnexion)
    - le système tente une résolution automatique
      - version “dominante”
      - modifications permutables (ajouts dans catalogue)
    - si impossible, alors résolution manuelle

Coda : conclusion

- Expérience utile pour le mode déconnecté
  - Plusieurs versions successives
  - Mesures et observations
- Conclusions pratiques
  - Les conflits non résolubles sont en fait très rares
    - justifie a posteriori la stratégie optimiste
  - La disponibilité est satisfaisante
    - pas de pertes de données
  - Le coût de la réintégration est acceptable

Pour en savoir plus :

http://www.cs.cmu.edu/afs/cs/project/coda/Web/coda.html