Systèmes de fichiers modernes

Caches pour les disques

Fiabilité en cas de panne

Journaled FS

Log FS

NFS

Cacher les accès aux disques

Garder dans un *cache*, en mémoire, les données lues du disque

Retarder les écritures

- Pour les ordonnancer
- Certaines deviennent inutiles ou redondantes

Unifier le *page cache* et le *buffer cache*: éviter le *double caching*

Attention à la cohérence du cache

dcache: cache spécialisé pour les *directory entries*

Cohérence en cas de panne

Ajouter un bloc à un fichier requiert les modifications suivantes sur le disque:

- Écrire le bloc
- Mettre à jour l'*inode*
- Mettre à jour la liste des blocs libres

Que se passe-t-il si le système s'arrête à mi-course?

Créer ou détruire un fichier est encore pire

Différentes formes d'incohérences

Espace disque libre mais inutilisable

Modifications perdues

Fichiers en cours de modification corrompus

Victimes innocentes

Fiabilité aux pannes

Outil de remise en forme *fsck*:

- Ne peut pas faire de miracles
- Coûteux en temps, retarde le redémarrage après une panne

Miracles: Ordonnancer avec soin les écritures

Rabais: Ajouter un *journal* (ou *log*) pour annoncer les changements

Journaling file-systems

Avant toute modification, écrire une description de la modification

Le *journal* (ou *log*) garde la description des modifications en cours

Modifications complétées peuvent être retirées du log

Il faut quand même ordonnancer les écritures avec soin:

- Écrire le log sur le disque avant d'effectuer les autres écritures

Peut doubler le nombre d'écritures!

Une description peut être: data+metadata ou metadata-only

Possibilité de retarder les modifications à volonté

Grosses écritures efficaces en RAID-5

Log-structured file-systems

Comme *Journaling* mais on ne garde que le journal.

Exemple: Sprite LFS, JFFS2

Avantages:

- Écritures par gros *segments* séquentiels: rapides!
- Les anciennes données ne sont pas écrasées: snapshots for free!

Inconvénients:

- Les inodes ne sont plus fixes: il faut un *inode-map*!
- Il faut récupérer les blocs obsolete: *cleanup* ou *GC*!
- La lecture peut être plus lente, mais surtout, le GC a tendance à devenir très lent quand le disque est plein

Systèmes de fichiers et mémoires flash

FTL (converti un *MTD* en un *bloc device*)

Système de fichiers sur mesure (e.g. JFFS2, YAFFS)

Compromis: UBI + UBIFS

JFFS2

Chaque *erasure bloc* contient une séquence de *nodes*

- Chaque *node* décrit un morceau de fichier ou de répertoire
- Chaque *node* contient un n^o d'*inode* et une version

En mémoire, JFFS2 garde une listes de *nodes* et une table des *inodes*

l'*erasure bloc* courant est rempli séquentiellement

- Lorsqu'il est plein, on en prend un autre libre
- Le GC tourne en tâche de fond pour libérer des *erasure bloc*

Lors du *mount*, il faut lire l'ensemble de la mémoire flash

Les données sont compressées

Unsorted Bloc Image (UBI)

UBI transforme les *physical erasure blocs* en *logical erasure blocs*

Fait une partie du travail d'une FTL:

- *wear-leveling*
- Gestion des erreurs (*bad blocs*)
- Mise à jour atomique d'un *logical erasure bloc*
- Ajoute la notion de *Volume*, similaire à une partition

UBI pour l'instant ne stocke pas la *bloc map* sur disque

Chaque bloc se décrit et la table et reconstruite au démarrage

Le *scan* est plus rapide que pour JFFS2 (nb de blocs $<$ nb de *nodes*)

Détour sur les systèmes distribués

Difficile de diagnostiquer les pannes

Le problème du *consensus*:

peut-on se mettre d'accord sur un misérable bit?

Ordre partiel *vs* ordre total

Le problème de l'horloge

Système de fichiers par réseau

Accéder “localement” aux fichiers d’une autre machine

Idée de départ: transférer les appels VFS par RPC

Problèmes

- Fichiers modifiés hors de notre contrôle
- Gestion du cache
- Gestion des pannes
- Authentification
- Désaccords entre VFS et le protocole utilisé

NFS: Network File System

Le serveur ne se préoccupe pas de l'état des clients (*stateless*)

- Clients pas avertis lors de modifications de fichiers
- Clients vérifient la fraîcheur de leur cache (~30s)
- Panne client: rien à faire! Panne serveur: attendre
- Pas de verrous

Authentification triviale: vérification de l'IP, uid préservé (sauf *root*)

Limitations

- Cache pas toujours à jour (problème avec `make`)
- Opérations perdues

nqNFS: Network File System

Plus *stateless*, mais pas trop loin

Clients peuvent obtenir des *lease* sur des fichiers (e.g. 30s)

- Clients avec *lease* avisés lors de modifications

Cache cohérent

Panne client: rien à faire, le *lease* expire tout seul

Panne serveur: attendre 30s que tous les *lease* expirent

- Pannes artificiellement allongées!
- Contrainte forte sur la durée maximale d'un *lease*

NFS: problème des écritures

Le cache des clients filtre les lectures plus que les écritures

$\Longrightarrow$ Le serveur reçoit beaucoup d'écritures

Les clients retardent les écritures, comme d'habitude

$\Longrightarrow$ Le serveur reçoit des écritures *urgentes*

Pire avec *save on close*

Système de fichiers du serveur reçoit une charge inhabituelle

- Usage d'un journal en NVRAM pour pouvoir écrire vite
- Usage de système de fichiers dédié, e.g. WAFL