Bisoños Usuarios de GNU/Linux de Mallorca y Alrededores | Bergantells Usuaris de GNU/Linux de Mallorca i Afegitons

Systèmes de Fichiers Journalisés sur Linux (21483 lectures)

Per **Daniel Rodriguez**, ***DaniRC*** (http://www.ibiza-beach.com/)

Creado el 29/01/2002 19:41 modificado el 15/02/2002 10:26

Celle-ci est une version en français legèrement modifiée de deux articles publiés sur Novatica[1] et Upgrade[2] où Mr. Ricardo Galli explique l'implémentation de tous les systèmes de fichiers avec journaling disponibles en Linux.

Co-Traduction: Daniel R.C. et Xavier Verne
Remérciments de Daniel R.C. à Xavier pour les corrections et les aportations a la traduction.
Article en Anglais[3]
Article en Espagnol[4]

Ceci est une version en français légèrement modifiée de deux articles publiés sur Novatica et Upgrade où Mr. Ricardo Galli explique l'implémentation de tous les systèmes de fichiers avec journaling disponibles en Linux.
Co-traduit par Daniel R.C. et Xavier Verne.

Systèmes de fichiers journalisés sous Linux

Ricardo Galli
gallir@uib.es
Dépt. de Mathématiques et Informatique
Université des Iles Baléares

Premièrement, il n'y a pas vainqueur absolu, XFS est meilleur sur certains aspects et dans certains cas, ReiserFS dans d'autres, et les deux sont meilleurs que ext2 dans la mesure où ils ont des performances comparables (encore une fois, soit un peu plus rapide, soit légèrement plus lents), mais ils constituent des systèmes de fichiers journalisés, et vous savez déjà quels en sont les avantages...Et le point le plus important à retenir reste que le Buffer Cache Linux est vraiment impressionnant et a amélioré toutes mes compilations de test, copies et lectures/écritures aléatoires. Alors, je dirais, achète de la mémoire et installe un système de fichiers journalisé...

Mots-clefs : Linux, systèmes d'exploitations, système de fichiers journalisé, ext3, xfs, reiserfs, jfs.

Introduction

Le paragraphe ci-dessus est le premier d'un article publié sur le site Web du groupe d'utilisateurs de Linux des Iles Baléares. Il décrit une deuxième série de tests comparatifs entre les systèmes de fichiers journalisés Linux et les systèmes de fichiers traditionnels Unix.

Bien que relativement informels, les tests de comparaison couvraient FAT32, Ext2, ReiserFS, XFS, et JFS dans différentes situations : les outils Mongo de Hans Reiser, la copie de fichiers, la compilation du noyau, et un petit programme C qui simulait des accès courants d'une base de données.

Ces deux articles furent parmi les premiers publiés sur le sujet, et notre serveur fut submergé par "l'effet slashdot" qui fit suite à la publication des liens sur *slashdot.org*. Leur principale utilité fut de faire tomber le mythe selon lequel les systèmes de fichiers journalisés étaient significativement plus lents que les systèmes de fichiers Unix traditionnels (UFS) et par extension, namely ext2fs, le système de fichiers de Linux.

Depuis ce temps, plus de tests comparatifs se publient, mais la conclusion est la même : il n'y a pas de vainqueur toutes catégories. Certains systèmes ont des performances meilleurs dans certaines situations, à l'exemple de ReiserFS qui est excellent en lecture pour des fichiers de taille petite à moyenne, tandis que XFS se comporte mieux pour les fichiers plus importants, et que JFS semble faciliter la migration de systèmes fondés sur OS/2 Warp et AIX.

Cet article présente tous les systèmes de fichiers journalisés disponibles sous Linux: Ext3, ReiserFS, XFS, et JFS. Nous introduisons aussi les concepts de base des systèmes de fichiers, de buffer cache, et de page cache implémentés dans le noyau Linux. La performance de ces différents systèmes de fichiers est fortement augmentée par ces deux techniques d'optimisation. En effet, non seulement elle s'en ressent, mais aussi l'implémentation et le portage des différents systèmes de fichiers. SGI a introduit un nouveau module, *pagebuf*, qui sert d'interface entre leurs propres techniques de bufferisation XFS et le page cache de Linux.

Le Système de fichiers virtuel Linux

Un *fichier* est un important concept dans le domaine de l'informatique. Les fichiers servent à stocker les données de manière permanente, et offrent un petit jeu d'instructions simples mais très efficaces aux programmeurs. Usuellement, les fichiers sont organisés hiérarchiquement, via une structure en arbre, dont les n uds intermédiaires sont les répertoires, capables de contenir des fichiers et des sous-répertoires.

Le système de fichiers est la manière dont l'OS organise, maintient la hiérarchie des fichiers au sein des périphériques de stockage, normalement des disques durs. Tous les systèmes d'exploitation modernes supportent plusieurs systèmes de fichiers, éventuellement très différents. Pour préserver la modularité de l'OS et pour fournir des applications avec une interface de programmation identique (API), une nouvelle couche d'abstraction qui implémente les fonctionnalités communes des différents systèmes de fichiers sous-jacents est ajoutée au noyau : c'est le Système de Fichiers Virtuel (VFS pour Virtual File System).

Les systèmes de fichiers supportés par le Système de Fichiers Virtuel Linux se divisent en trois catégories :

1. Fondés sur des disques, incluant les disques durs, les disquettes et les CD-ROM, avec par exemple ext2fs, ReiserFS, XFS, ext3fs, UFS, iso9660,etc;
2. Systèmes de fichiers réseaux : NFS, Coda, et SMB;
3. Systèmes de fichiers spéciaux : /proc, ramfs et devfs.

Le modèle commun de fichier peut-être vu comme un modèle orienté objet, les objets étant des portions de code (structure de données ainsi que les méthodes/fonctions associées) des types suivants :

- **Super bloc** : stocke les informations relatives à un système de fichiers monté. Cela correspond à un bloc de contrôle résidant sur le disque (pour les systèmes de fichiers basés sur des disques).
- **I-node** : stocke l'information relative à un seul fichier. Cela correspond aussi à un bloc de controle stocké sur le disque. Chaque i-node possède la méta-information du fichier : propriétaire, groupe, date de création, date de dernier accès ainsi qu'un ensemble de pointeurs aux différents blocs physiques contenant les données du fichier.
- **Fichier** : stocke l'information relative à l'interaction entre un fichier ouvert et un processus. Cet objet existe seulement lorsqu'un processus interagit avec un fichier.
- **Entrée de répertoire (Dentry)** : relie une entrée de répertoire (chemin d'accès ou pathname) avec le fichier correspondant. Depuis peu, les objets de ce type sont cachés pour accélérer la translation entre un chemin d'accès et l'i-node correspondant au fichier.

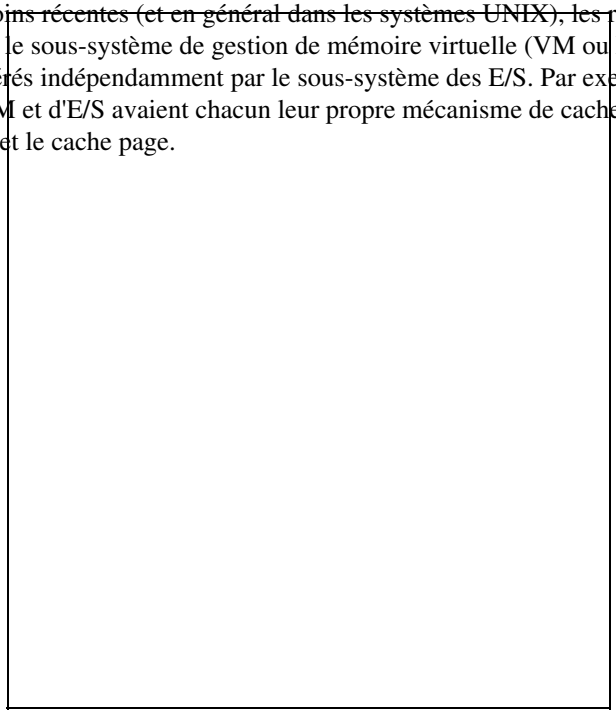
Tous les systèmes Unix modernes autorisent deux méthodes d'accès aux données du système de fichiers(figure 2).

1. Association de mémoire (memory mapping) avec **mmap** : l'appel système mmap() donne à l'application l'accès direct aux données du page cache. Le but de mmap et de translater les données d'un fichier dans un espace d'adressage du VMS, de manière à ce que le fichier puisse être traité comme un tableau ou une structure standard en mémoire. Les données du fichier sont lues dans le cache paresseusement lorsque le processus essaie d'accéder à ces données créés par mmap et génère un défaut de page.

2. Appel système en mode bloc tel que **read** ou **write** : l'appel système read() copies les données du périphérique bloc dans le cache du noyau (évité pour le CD et le DVD au moyen du paramètre O_DIRECT de la fonction *ioctl*), et ensuite il copie les données du cache du noyau dans l'espace mémoire de l'application. L'appel système write() copie les données dans l'autre sens, i.e. depuis l'espace mémoire de l'application vers le cache du noyau pour finalement, dans un futur proche, écrire les données depuis le cache vers le disque. Ces interfaces sont implémentées en utilisant soit le *buffer cache* soit le *page cache* pour stocker les données dans le noyau.

Le page cache et le buffer cache Linux

Dans des versions de Linux moins récentes (et en général dans les systèmes UNIX), les requêtes de mapping de mémoire étaient contrôlées par le sous-système de gestion de mémoire virtuelle (VM ou MM), tandis que les appels Entrées/Sorties (E/S) étaient gérés indépendamment par le sous-système des E/S. Par exemple, jusqu'aux versions 2.2.x de Linux, les sous-systèmes VM et d'E/S avaient chacun leur propre mécanisme de cache de données pour améliorer les performances : le buffer cache et le cache page.



1. Buffer Cache et Page Cache

Le Buffer Cache

Le buffer cache contient des copies de blocs de disque individuels. Le périphérique et les numéros de blocs servent à l'indexer. Chaque tampon (buffer) de mémoire réfèrence un et un seul bloc arbitraire sur le disque dur, et il est composé d'un en-tête (header) et d'une zone de taille *égale* à celle du bloc du périphérique associé.

Pour minimiser la surcharge de gestion, tous les buffers sont organisés en listes chaînées. Chacune d'entre elles contient les buffers dans le même état : inutilisé (unused), libre (free), propre (clean), sale (dirty), verrouillé (locked), etc.

A chaque lecture, le sous-système de buffer cache doit savoir si le bloc demandé est déjà dans le cache. Pour le retrouver facilement, une table de hachage (hash table) est tenue à jour de tous les buffers présents dans le cache. Le buffer cache est aussi utilisé pour améliorer les performances en écriture : au lieu de répercuter toutes les écritures disques immédiatement, le noyau stocke les données temporairement dans le buffer cache, attendant de voir s'il est possible de grouper plusieurs écritures. Un buffer contenant des données attendant d'être écrites sur le disque est dit "sale" (dirty).

Le Page Cache

Le page cache, lui, contient des pages mémoires complètes (4Ko sur les plates-formes Linux x86). Les pages viennent de fichiers du système de fichiers, et en fait, les entrées du page cache sont partiellement indexées par le numéro d'i-node et son décalage (offset) à l'intérieur du fichier. Une page est presque invariablement plus *grande* qu'un bloc disque logique, et les blocs qui constituent une page de cache ne sont éventuellement pas contigus sur le disque.

Le page cache (tampon de pages) est très largement utilisé pour interfacer les besoins du sous-système de mémoire virtuelle, qui utilise une taille de page fixe de 4Ko, et du sous-système VFS, qui utilise des blocs de taille variable ou d'autres techniques, tel que "extents" dans XFS et JFS.

Intégration des Page et Buffer Cache

Les deux mécanismes présentés opéraient de manière semi-indépendante l'un de l'autre. Le système d'exploitation devait porter une attention toute particulière pour synchroniser ces deux caches et ainsi prévenir l'envoi de données invalides aux applications. En outre, si le système est à court de mémoire, il doit prendre des décisions arbitraires sur la façon de libérer la mémoire du page cache et du buffer cache.

Le page cache a tendance à être plus facile à utiliser, puisqu'il représente plus clairement les concepts utilisés dans des couches plus hautes du noyau. Le buffer cache possède aussi la limitation suivante : les données cachées doivent *toujours* être mappées dans l'espace virtuel du noyau. Cela ajoute une limite artificielle sur la quantité totale de données qui peuvent être mises en cache (cachées) puisque que le matériel moderne a facilement plus de RAM que l'espace mémoire du noyau.

Ainsi avec le temps, **des parties du noyau ont progressivement favorisé l'utilisation du Page Cache vis à vis du Buffer Cache**. Les blocs individuels d'une entrée de page cache sont toujours mis à jours par le buffer cache. Mais accéder le buffer cache directement peut entraîner une confusion entre les deux niveaux de tampons.

Ce manque d'intégration conduisit à une performance faible de tout le système et un défaut de flexibilité. Pour avoir de bonnes performances il est dès lors nécessaire d'intégrer complètement les sous-systèmes de mémoire virtuelle et d'entrées/sorties.

L'approche suivie par Linux pour réduire l'inefficacité des copies multiples consiste à **stocker les données d'un fichier uniquement dans le page cache**(figure 3).

2. Les données sont partagées par les Page Cache et Buffer Cache.

Des mappings temporaires de pages du cache de pages pour supporter read() et write() sont presque obligatoires puisque Linux mappe en permanence toute la mémoire physique dans la mémoire virtuelle du noyau. Un mécanisme intéressant est ajouté par Linux : les numéros de bloc du périphérique où une page réside sur le disque sont cachées **avec la page** sous la forme d'une liste de structure de type *buffer_head*. Quand une page modifiée doit être écrite sur le disque, les requêtes d'entrée-sortie peuvent être envoyées directement au pilote de périphérique, sans avoir nul besoin de lire d'autres blocs pour savoir où les données doivent être écrites.

Unification du Page Cache

Imitant le "sous-système E/S et de tampon de mémoire unifié pour NetBSD", Linus Torvalds envisageait sérieusement de changer le comportement du tampon de page pour Linux. Le 4 mai 2001, dans un message à la liste des développeurs du noyau Linux, il écrivait :

> *Je veux vraiment réécrire block_read/write pour utiliser le cache de pages, mais pas parce que cela influencerait quoi que ce soit dans cette discussion. Je veux le faire très tôt dans 2.5.x parce que:*
>
> *- cela accélérera les accès;*

- cela améliorera la réutilisation du code et simplifiera les concepts (i.e. cela transformera le disque en un système de fichiers très_simple constitué d'un seul énorme fichier ;) ;

- cela rendra la gestion de bien meilleure pour des choses telle que fsck - la pression de la mémoire est faite pour travailler sur des choses relatives au page cache.

- cela fera une chose de moins qui utilise le buffer cache comme un "cache" (Je vous invite à réfléchir au buffer cache comme à une entité d'entrées-sorties, pas comme à un cache, et à l'utiliser en tant que tel).

Cela ne changera rien au "cache de démarrage" (puisque même dans le page cache, il n'y a rien de commun entre le mapping virtuel d'un fichier (ou de méta-données) et le mapping virtuel de disque.

Bien que ces changements utiles n'étaient pas attendus avant 2.5.x, Linus opta finalement pour intégrer un ensemble de patches de Andrea Arcangeli, ainsi que des changements personnels, et sortit le noyau 2.4.10, **qui unifie finalement les Page et Buffer Cache** (Figure 3).Ainsi, une importante amélioration dans les opérations E/S est attendue, conjointement à une meilleure optimisation de la mémoire virtuelle, spécialement dans des situations de manque de mémoire.

3. Unification des Page Cache et Buffer Cache

Systèmes de fichier journalisés

Le système de fichiers standard était *ext2fs*. Ext2 fut créé par Wayne Davidson en collaboration avec Stephen Tweedie et Theodore Ts'o. C'est une amélioration du système de fichiers précédent *ext* fait par Rémy Card. Ext2fs est un système fondé sur l'i-node, qui contient les méta-données associées aux fichiers et les pointeurs vers les blocs physiques de celui-ci.

Pour accélérer les opérations d'E/S, les données résident temporairement en RAM au moyen des sous-systèmes de buffer cache et page cache. Le problème intervient lorsqu'un crash du à une coupure électrique survient avant que les données modifiées de la RAM (tampons "sales") n'aient été écrites sur le disque. Cela impliquerait une incohérence du système tout entier, par exemple un nouveau fichier qui n'avait pas été créé sur le disque ou des fichiers effacés mais dont les i-nodes et blocs de données correspondants résideraient encore sur le disque.

L'outil **fsck** (file system check) était dédié à résoudre ce genre d'incohérences. Mais fsck doit scanner toute la partition et vérifier les interdépendances parmi les i-nodes, les blocs de données et le contenu des répertoires. Avec l'augmentation de la taille des disques, restaurer l'intégrité d'un disque est devenu une tâche très coûteuse en temps, ce qui implique de sérieux problèmes de disponibilité pour des gros serveurs. C'est la raison principale pour laquelle les systèmes de fichiers héritent des technologies issues des bases de données et de la récupération de données, et expliquent l'apparition des *Systèmes de Fichiers journalisés*.

Un système de fichiers journalisé (SFJ) est un système qui réagit aux erreurs : l'intégrité des données est assurée au moyen de mises à jour des méta-données des fichiers dans des fichiers journaux (log files) écrits avant que les blocs disques originaux soient eux-mêmes mis à jour. Lors d'un évenement telle qu'une défaillance du système, un fichier exhaustif relatif au système de fichiers assure que le système de fichier est restauré. L'approche la plus commune est une méthode de *journalisation* ou de consignation (journaling or logging) des métadonnées des fichiers. Dès lors, à chaque fois que quelque chose est modifié dans ces métadonnées, cette nouvelle information est sauvegardée dans un endroit spécifique du système de fichiers. Celui-ci écrira les données elles-même seulement après l'écriture complète de ce log. Quand un crash surviendra, le programme de récupération du systéme analysera ces fichiers de métadonnées et essaiera de nettoyer seulement les fichiers incohérents en rejouant le fichier de log.

Les pionners des systèmes de fichiers, conçus au milieu des années 80, incluaient Veritas (VxFS), Tolerant, et JFS d'IBM. Ces dernières années, du fait de l'exigence croissante des systèmes de fichier pour qu'ils supportent des terabytes de données, des milliers et des milliers de fichiers par répertoire et des capacités 64-bits, l'intérêt des SFJ pour Linux s'est accru.

Ces derniers mois, Linux s'est enrichi de nouveaux prétendants dans ce domaine : ReiserFS[5] de Namesys, XFS[6] de SGI, JFS[7] d'IBM, et Ext3[8] développé par Stephen Tweedie, co-créateur de Ext2.

Tandis que ReiserFS est un système complètement nouveau écrit à partir de rien, XFS, JFS, et Ext3 dérivent tous de produits commerciaux ou de SF existants. XFS est fondé sur le système développé par SGI pour ses stations de travail et serveurs. Il en partage d'ailleurs des parties de code. JFS fut conçu et développé par IBM pour son OS/2 Warp, qui est lui même un dérivé du système de fichiers AIX.

ReiserFS est le seul inclus dans l'arborescence du noyau standard Linux, pour les autres il est prévu des les inclure dans les versions 2.5.x, et même si XFS et JFS sont parfaitement fonctionnels et opérationnels, ils sont officiellement publiés en tant que patchs au noyau.

Ext3 est une extension de ext2. Il lui ajoute deux modules indépendants, un module de transaction et un module de log. Ext3 est proche de sa version finale, REdHat 7.2 l'inclue déjà comme une option et il sera le système de fichiers officiel des distributions RedHat.

B-Trees

L'outil de base pour améliorer les performances des systèmes de fichiers type UNIX traditionnels est d'éviter l'utilisation des listes chaînées - pour les blocs libres, les entrées de répertoire et l'adressage des blocs de données - qui ont des problèmes intrinsèques de changement d'échelle (la complexité des algorithmes de recherche est en O(n)) et ne sont pas adéquates pour les nouveaux disques à très grande capacité. Tous les nouveaux systèmes utilisent des *arbres équilibrés* ou (B-Trees: Balanced Trees) ou des variations (comme les B+Trees).

Un arbre équilibré est une structure bien étudiée, plus robuste en terme de performances mais en même temps plus complexe à maintenir et à équilibrer. La structure dite en B+Tree est utilisée depuis longtemps dans les systèmes d'indexation des bases de données. Elle fournit aux bases de données une manière rapide d'accéder aux enregistrements, *presque* indépendamment de la taille du disque. Le symbole + signifie que le B-Tree est une version modifiée de l'original telle que :

- elle place toutes les clés aux feuilles;
- les feuilles (noeuds terminaux) peuvent être liées entre elles;
- les noeuds (internes) et les feuilles peuvent être de taille différente;
- il n'y a jamais besoin de modifier le père lorsqu'une clé dans une feuille est effacée;
- cela rend les opérations séquentielles plus faciles et moins coûteuses.

ReiserFS

ReiserFS implémente des B+Tree "rapides" (fast balanced trees) pour organiser les objets du système de fichiers. Ces objets sont les structures utilisées pour conserver l'information relative au fichier : (dernier) temps d'accès, permissions, etc. En d'autres termes, l'information contenue dans l'i-node, les répertoires et les données du fichier. ReiserFS appelle ces objets respectivement, *stat data* items, *directory* items and *direct/indirect* items. ReiserFS fournit seulement la journalisation des métadonnées. En cas de reboot non voulu, les blocs de données en cours d'utilisation au moment du crash pourraient avoir été corrompues; ainsi ReiserFS ne garantit aucunement que le contenu des fichiers est intègre.

Les noeuds non formatés (unformatted nodes) sont des blocs logiques sans format predefini, utilisés pour stocker des données de fichiers, et les *direct items* sont les données du fichier lui-même. Aussi ces items sont de taille variable et stockés dans les noeuds feuilles de l'arbre, quelquefois avec d'autres si la place est suffisante dans le noeud. L'information relative au fichier est stockée près des données de ce fichier, puisque le système de fichiersessaye toujours de mettre les *stat data* items et les *direct/indirect* items d'un même fichier **ensemble**. A l'inverse des *direct* items, les données pointées par des *indirect* items ne sont pas sauvegardées dans l'arbre. Cette méthode originale des *direct* items est due au support des petits fichiers : c'est le *tail packing* (littéralement, empaquetage des queues).

Le *tail packing* est une caractéristique spécifique à ReiserFS. Les tails sont des fichiers plus petits qu'un bloc logique, ou les fins de fichiers (la **queue** du fichier) qui ne remplissent pas complètement un bloc. Pour économiser de l'espace disque, ReiserFS utilise le *tail packing* pour ranger les queues dans le minimum d'espace possible. Généralement, cela permet à ReiserFS une économie qui tourne autour de 5% par rapport à un système fichier Ext2 équivalent. Les *direct* items sont prévus pour conserver les données des petits fichiers et même les queues de fichiers. De plus, plusieurs queues peuvent résider à l'intérieur du même noeud feuille.

ReiserFS est très performant pour les petits fichiers car il est capable d'incorporer ces queues au sein même du B-Tree de manière à les garder vraiment proches de leur *stat data*. Puisque les queues ne remplissent pas totalement un bloc, elle peuvent amener à perdre de l'espace disque.

Le problème est qu'en utilisant cette technique qui rassemble les queues des fichiers on augmente la fragmentation externe, puisque les données du fichier sont maintenant éventuellement loin de sa queue. En outre, l'empaquetage des queues est consommateur de ressources et implique une baisse des performances. C'est une conséquence des déplacements de mémoire nécessaires quand quelqu'un ajoute des données à un fichier existant. Namesys connaît ce problème et autorise l'administrateur système à désactiver l'empaquetage des queues en ajoutant l'option *notail* lorsque le système de fichiers est monté ou même remonté.

ReiserFS utilise une taille de blocs fixe (4Ko) - orientée allocation - et donc pénalise les opérations d'E/S des gros fichiers. L'autre faiblesse de ReiserFS vient de la performance relative aux fichiers épars sur le disque (blocs non contigus), qui est significativement moins bonne que celle de ext2, bien que Namesys travaille à son optimisation.

4. Allocation basée sur le bloc.

XFS

Le 1er Mai 2001, SGI rendait disponible la release 1.0 de son SFJ XFS pour Linux. XFS est reconnu pour son support des fermes de disques à grande capacité et pourde très hauts débits d'entrées/sorties (jusqu'a 7Go/s). XFS fut développé pour le système d'exploitation de type Unix IRIX 5.3 SGI, et sa première version fut introduite en décembre 1994. L'objectif de ce système de fichiers était de supporter de très gros fichiers et des taux de transferts élevés pour jouer et enregistrer en temps réel de la vidéo.

Pour avoir cette flexibilité vis à vis de la taille des fichiers, XFS utilise intensivement les B+Trees. Ils sont utilisés pour tracer des zones libres, des index de répertoires, et pour garder la trace des i-nodes alloués dynamiquement répartis à travers tout le système de fichiers. En plus, XFS utilise une écriture asynchrone avec un mécanisme qui protège les mises à jour des métadonnées et autorise une récupération rapide du système.

XFS utilise une allocation de l'espace spécifique, et possède des fonctionnalités telle que l'allocation retardée, la préallocation d'espace, la coalescence d'espace après effacement, et cherche activement a disposer les fichiers en utilisant les "extents" (??) les plus larges possibles. Pour rendre la gestion de grandes quantités d'espace contigus efficace, XFS utilise des descripteurs de fichiers très grand pour faire la carte du système. Chaque descripteur peut renseigner jusqu'a 2 millions de blocs du système. Décrire un grand nombre de blocs avec un seul descripteur étendu élimine le travail du CPU consistant à scanner les entrées de la carte pour déterminer si les blocs d'un fichier sont contigus, puisqu'il suffit de lire la longueur de l'extension plutôt que d'examiner chaque entrée pour voir si elle est contiguë avec la précédente.

5. Allocation étendue.

XFS autorise des tailles de blocs variable, de 512 octets à 64 Kilooctets sur un système fondé sur le fichier . Changer cette taille de bloc modifie la fragmentation. Les systèmes avec de grands nombres de petits fichiers utilisent typiquement des tailles de bloc plus petites avec pour objectif d'éviter au maximum la perte d'espace due à la fragmentation interne. Les systèmes comportant de gros fichiers ont tendance à faire le choix inverse et utilisent des blocs plus grands pour réduire la fragmentation externe du système de fichierset des extents de ses fichiers.

XFS est un énorme morceau de code IRIX très tourné vers IRIX, de telle sorte que le portage vers Linux de cette interface fut conçu à nouveau et réécrit en partant de zéro. Le résultat est le module Linux *pagebuf*, qui fournit

l'interface entre XFS et le sous-système de mémoire virtuelle ainsi qu'entre XFS et la couche Linux relative au périphérique par blocs.

XFS supporte ACL's - Access Control Lists - (intégrées avec le serveur Samba) et les quotas transactionnels. Sur Linux il supporte les quota par groupe en lieu et place des quotas par projet du système IRIX, en accord avec les autres systèmes de fichiers Linux (et Linux ne possède pas d'équivalent à la notion de projet). D'autres fonctionnalités plus ésotériques de XFS sur IRIX qui fournissent des services pour des applications spécifiques (par exemple pour un serveur de vidéo temps-réel) n'ont, à ce jour, pas été portées sous Linux.

Le mode normal de fonctionnement de XFS utilise un fichier *de log écrit de manière asynchrone*. Cela permet d'assurer que le protocole d'écriture du fichier de log est suivi de telle sorte que des données modifiées ne peuvent pas être écrites sur le disque tant que les données ne sont pas validées au niveau du fichier de log. XFS gagne sur deux tableaux avec l'écriture asynchrone du log :

1. Des mises à jour mutiples peuvent être regroupées dans une seule opération d'écriture du log. Cela augmente l'efficacité des écritures dans le log (il y a un disque physique).
2. La performance des mises à jour des métadonnées est normallement rendue indépendante de la vitesse intrinsèque des périphériques concernés. Cette indépendance est limitée par la taille de la mémoire tampon dediée au log, mais c'est - de loin - bien meilleur que la mise a jour synchrone des anciens systèmes de fichier.

XFS possède aussi tout un jeu d'outils de gestion d'espace utilisateur : vider, restaurer, réparer, agrandir, faire une image ainsi que d'autres outils pour utiliser les ACLs et les quotas disque,etc.

Extra de la traduction française: Lire sur les commentaires de la dernière page le commentaire de Seth Mos, entreteneur des pages de FAQ de XFS.

JFS

JFS (Journaled File System) est le système de fichiers UNIX introduit par IBM, avec la version initiale 3.1 de AIX. Il a depuis introduit un second système de fichiers fait pour les systèmes AIX appellé Enhanced Journaled File System (JFS2), qui est disponible dans les versions 5.0 et suivantes de AIX. Le code open source de JFS provient à l'origine de la migration des serveurs OS/2 Warp pour l'e-Business.

JFS est fondamentalement étudié pour des serveurs exigeants de très hauts débits et une grande disponibilité. JFS utilise des structures d'adressage étendues (extent-based), ainsi qu'une politique d'allocation des blocs par groupe, pour produire des structures compactes, efficaces et flexibles pour translater les offsets logiques des fichiers vers les adresses physiques du disque. Une "extent" (ndT: unité d'allocation d'espace variable...) est une séquence de blocs contigüs alloués à un fichier en une seule fois. Elle est décrite par un triplet <offset logique, longueur, adresse physique > (ndT: l'offset logique donne la position à l'intérieur du fichier). La structures d'adressage est un arbre équilibré (B+Tree) peuplé de ces descripteurs, prenant racine dans l'i-node et repérés par l'offset logique dans le fichier.

Les fichiers de log JFS sont mis à jour dans chaque système de fichiers et utilisés pour enregistrer les opérations sur les métadonnées. Le format de ce fichier de log est défini par l'utilitaire de création du système de fichiers.

Les règles d'utilisation de ces logs sont telles que, lorsqu'une opération du système de fichiersimpliquant des modifications des métadonnées renvoie une valeur synonyme de succès, les effets de cette opération sont déjà effectifs au système de fichiers et seront vus même si le système plante immédiatement après l'opération.

L'ancien style de log introduisait une écriture synchrone au fichier disque de log dans i-node ou VFS qui modifiait les métadonnées. En termes de performance, c'est un désavantage certain par rapport aux autres systèmes de fichiers, tels que Veritas VxFS et XFS, qui utilisent d'autres politiques pour le log et écrivent paresseusement les données de log sur le disque. Quand des accès concurrentiels interviennent, le cout de l'opération est réduit en regroupant les opérations, qui combine plusieurs opérations synchrones d'écriture en une seule. Le style de log de JFS a été amélioré et fournit actuellement une méthode asynchrone, qui augmente les performances du système.

JFS supporte des tailles de bloc de 512, 1024, 2048, et 4096 octets sur un système dont l'unité est le fichier. Des blocs plus petits reduisent globalement la fragmentation interne. Pourtant, de petits blocs peuvent augmenter la longueur des

chemins puisque l'allocation de blocs intervient plus souvent qu'avec un bloc plus grand. La taille par défaut est 4096 octets.

JFS alloue l'espace pour les i-nodes dynamiquement, libérant l'espace quand il n'est plus utile. Deux organisations des répertoires sont disponibles.

1. La première organisation est utilisée pour des petits répertoires et stocke le contenu des répertoires à l'intérieur même de l'i-node de ce répertoire. Cela évite de recourir à des E/S séparées pour les répertoires ainsi que des besoins d'allocation d'espace séparés.

2. La deuxième organisation est utilisée pour des répertoires plus importants et représente chaque répertoire comme un arbre équilibré repéré par un nom. Cela fournit un moyen de parcours rapide des répertoires, et des possiblités supplémentaires d'insertion et d'effacement.

JFS supporte et les fichiers denses (ndT: contigus) et les fichiers épars (non contigus), sur un système fondé sur le fichier. Les fichiers épars autorisent les données à être écrites au hasard dans un fichier sans instancier les autres blocs du fichier non écrits. La taille de fichier reportée est le plus grand octet qui a été écrit, mais la véritable allocation d'un bloc donné dans un fichier n'intervient pas jusqu'a ce qu'une opération d'écriture soit effectuée sur ce bloc.

Ext3

Ext3 est compatible avec Ext2, en fait c'est un système ext2fs avec un fichier de journal. Ext3 n'est qu'un système de fichiers journalisé *partiel*, c'est simplement une couche au dessus du traditionnel ext2fs qui enregistre dans un fichier l'activité d'un disque et accèlere ainsi la récupération après un shutdown incorrect en comparaison de ext2 seul. Mais, parce qu'il est lié à ext2, il souffre des limitations de celui-ci et ce faisant n'exploite pas tout le potentiel d'un système de fichiersjournalisé "pur". Par exemple, son unité de base est encore le bloc, et il recherche les noms des fichiers séquentiellement dans les répertoires.

Ses avantages principaux sont :

- Ext3 journalise et maintient la cohérence des données **et** des métadonnées. A la différence des autres systèmes de fichiers journalisés précedemment décrits, la cohérence est assurée aussi pour le contenu des fichiers. Le niveau de journalisation peut être contrôlé avec des options de montage.
- Les partitions Ext3 ont la même structure de fichiers que ext2, ce qui rend le portage et la sauvegarde d'un ancien système directe, que ce soit par choix ou si le fichier de journal lui même venait à être corrompu.

Ext3 réserve un des i-nodes spéciaux de ext2 pour stocker le journal, mais celui-ci peut résider sur n'importe quel i-node de n'importe quel système de fichiers et peut même être dans un sous-ensemble de blocs contigüs sur un périphérique bloc donné. Il est possible d'avoir plusieurs systèmes de fichiers partageant le même journal.

Le travail du fichier de journal est d'enregistrer le contenu nouveau des blocs de métadonnées du système lorsque ceux-ci sont en cours de validation. Le seul prérequis est que le système doit assurer l'atomicité de ces transactions.

Trois types de blocs sont écrits dans le journal :

1. Des métadonnées;
2. Des descripteurs de blocs; et
3. Des en-têtes de blocs.

Un bloc de métadonnées contient le bloc entier du système de fichier mis à jour lors d'une transaction. A chaque fois qu'un changement est effectué sur le système de fichiers, un bloc entier doit être écrit sur le journal. Pourtant, cela est relativement peu coûteux car les opérations d'E/S sur le journal peuvent être regroupées (batched) et les blocs peuvent être écrits directement depuis le système de page cache, exploitant ainsi la structure *buffer_head*.

Les blocs descripteurs décrivent d'autres blocs de métadonnées du journal de telle sorte que le mécanisme de récupération puisse copier ces métadonnées vers le système de fichiers principal.

Enfin les blocs d'en-tête décrivent la tête et la queue du journal ainsi qu'un numéro de séquence garantissant l'ordre d'écriture pendant la restauration.

Performances et conclusions

Différentes comparaisons (voir la section ressources en dessous) ont montré que XFS et ReiserFS ont de très bonnes performances comparativement à ext2fs, pourtant largement testé et optimisé. Ext3 semble être un peu plus lent même si il se rapproche de ext2. Il faut s'attendre à ce que les performances s'améliorent considérablement aux cours des prochains mois. D'autre part, JFS a obtenu les résultats les moins bons de tous les tests comparatifs, non seulement en termes de performance mais aussi à cause de problèmes stabilité (dus au portage Linux).

XFS, ReiserFS et Ext3 ont démontré qu'ils sont d'excellents et de très fiables systèmes de fichiers. XFS est résolument performant pour les E/S de très gros fichiers, spécialement vis à vis de son adversaire le plus proche, ReiserFS. C'est compréhensible et susceptible d'évoluer, dans la mesure où il utilise la bibliothèque générique 2.4 de lecture et d'écriture Linux, tandis que XFS a porté des routines d'E/S sous Linux, la plus importante étant l'allocation étendue et les opérations d'E/S directes. De plus, la version actuelle de ReiserFS fait un passage complet de tout l'arbre pour chaque bloc de 4Ko qu'il écrit, et alors seulement ajoute un pointeur, ce qui introduit une surcharge pour équilibrer l'arbre lorsqu'il copie les données autour.

Pour le travail sur de petits fichiers, typiquement entre 100 et 10 000 octets, ReiserFS a demontré les meilleurs résultats, lorsque les fichiers concernés n'étaient pas encore dans le cache (comme pendant le démarrage par exemple). Dans le cas où le système lit des fichiers qui sont déjà cachés en mémoire vive, la différence est en pratique négligeable pour tous les systèmes (ext2, Ext3, ReiserFS et XFS).

Parmi tous les systèmes de fichiers journalisés ReiserFS est le seul inclus en standard dans l'arborescence Linux standard puisque les versions 2.4.1 et la distribution SuSE le supportent depuis maintenant 2 ans. Pourtant, Ext3 va devenir le système de fichiers standard des Red Hat et XFS est à même d'être utilisé sur de gros serveurs, tout spécialement dans l'industrie d'Hollywood, du fait de l'influence de SGI sur ce marché. IBM devra faire de gros efforts sur JFS s'ils désirent le garder parmi les systèmes de fichiers en vogue, bien qu'il constitue une alternative valide pour migrer des systèmes AIX et OS/2 vers Linux.

Ressources

- Architecture Ext3 [9]
- Introduction aux systèmes de fichiers journalisés[10]
- Page d'accueil XFS[11]
- Page d'acceuil JFS[7]
- Page d'acceuil ReiserFS[12]
- Solutions de stockage Storage Foundry[13]
- OS News[14]

Comparatifs

- Ext2, Ext3, ReiserFS, XFS et JFS benchmarks[15]
- Benchmarks Ext2, ReiserFS et XFS[16]
- Benchmarks Namesys[17]
- Benchmarks Mongo de Ext2, ReiserFS et JFS[18]

A propos de ce document...

La version anglaise a été à l'origine faite en utilisant **LaTeX**2HTML[19] translator Version 2K.1beta (1.48)
Par Ricardo Galli 2001-10-22

Traduite vers le français par Xavier Verne le 4-02-2002. Pour toutes remarques ou questions concernant cette traduction, nécessairement imparfaite, verne@enst.fr
Depuis la parution de cet article, il y a eu des changements, notamment une amélioration notable de JFS, ainsi qu'une

optimisation de ReiserFS. Les liens vers les sites des différents systèmes de fichiers sont *très* riches en information.

Lista de enlaces de este artículo:

1. http://www.ati.es/novatica
2. http://www.upgrade-cepis.org/issues/2001/6/upgrade-vII-6.html
3. http://bulma.net/body.phtml?nIdNoticia=1154
4. http://bulma.net/body.phtml?nIdNoticia=1153
5. http://www.namesys.com
6. http://bulma.net/http//oss.sgi.com/projects/xfs/
7. http://oss.software.ibm.com/developerworks/opensource/jfs/
8. http://e2fsprogs.sourceforge.net/ext2.html
9. ftp://ftp.kernel.org/pub/linux/kernel/people/sct/ext3/
10. http://www.linuxgazette.com/issue55/florido.html
11. http://oss.sgi.com/projects/xfs/
12. http://www.namesys.com/
13. http://sourceforge.net/foundry/storage/
14. http://www.osnews.com/story.php?news_id=69
15. http://wwww.mandrakeforum.org/article.php?sid=1212
16. http://bulma.net/body.phtml?nIdNoticia=642
17. http://www.namesys.com/benchmarks/benchmark-results.html
18. http://bulma.net/body.phtml?nIdNoticia=648
19. http://www-texdev.mpce.mq.edu.au/l2h/docs/manual/

E-mail del autor: danircJUBILANDOSEbulma.net