Équipe d'ingénierie de l'Internet (IETF)
Request for Comments: 5663
Catégorie : Sur la voie de la normalisation
ISSN: 2070-1721
Traduction Claude Brière de L'Isle

D. Black, EMC Corporation
S. Fridella, EMC Corporation
J. Glasgow, Google
janvier 2010

Disposition parallèle de bloc/volume du système de fichier réseau (pNFS)

Résumé
NFS parallèle (pNFS, *Parallel NFS*) étend la version 4 du protocole de partage de fichier de réseau (NFSv4, *Network File Sharing version 4*) pour permettre aux clients d'accéder directement aux données de fichier sur la mémorisation utilisée par le serveur NFSv4. Cette capacité d'outrepasser le serveur pour l'accès aux données peut améliorer les performances et le parallélisme, mais exige des fonctions de client supplémentaires pour l'accès aux données, dont certaines dépendent de la classe de mémorisation utilisée. Le document des opérations principales de pNFS spécifie des extensions indépendantes de la classe de mémorisation à NFS ; le présent document spécifie les extensions supplémentaires (principalement de structures de données) à utiliser pour pNFS avec une mémorisation fondée sur le bloc et le volume.

Statut de ce mémoire
Ceci est un document de l'Internet sur la voie de la normalisation.

Le présent document a été produit par l'équipe d'ingénierie de l'Internet (IETF). Il représente le consensus de la communauté de l'IETF. Il a subi une révision publique et sa publication a été approuvée par le groupe de pilotage de l'ingénierie de l'Internet (IESG). Tous les documents approuvés par l'IESG ne sont pas candidats à devenir une norme de l'Internet ; voir la Section 2 de la RFC5741.

Les informations sur le statut actuel du présent document, tout errata, et comment fournir des réactions sur lui peuvent être obtenues à http://www.rfc-editor.org/info/rfc5663

Table des matières

1. Introduction

La Figure 1 montre l'architecture globale d'un système NFS parallèle (pNFS) :

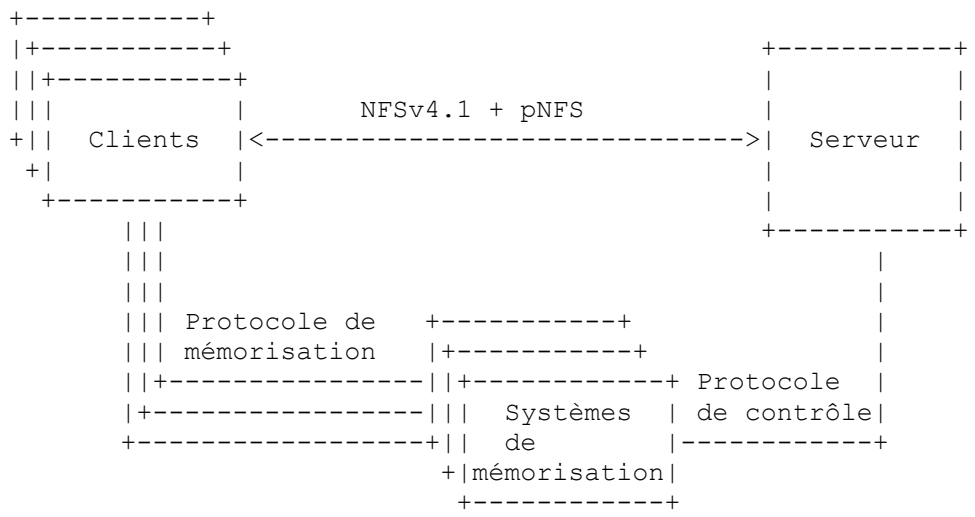

Figure 1 : Architecture de pNFS

L'approche globale est que les clients de pNFS amélioré obtiennent des informations suffisantes du serveur pour leur permettre d'accéder directement à la mémorisation sous-jacente (sur les systèmes de mémorisation). Voir la portion pNFS de la [RFC5661] pour les détails. Le présent document est concerné par l'accès à partir des clients pNFS aux systèmes de mémorisation sur des protocoles de mémorisation fondés sur des blocs et volumes, comme de la famille de protocoles d'interface de système de petit ordinateur (SCSI, *Small Computer System Interface*) (par exemple, SCSI parallèle, protocole de canal fibre (FCP, *Fibre Channel Protocol*) pour canal fibre, SCSI Internet (iSCSI, *Internet SCSI*) SCSI attaché en série (SAS, *Serial Attached SCSI*), et canal fibre sur Ethernet (FCoE, *Fibre Channel over Ethernet*)). Cette classe de mémorisation est appelée la mémorisation de bloc/volume. Bien que le protocole de système de serveur à mémorisation, appelé le "protocole de contrôle", ne pose pas ici de problème d'interopérabilité, il va normalement aussi être un protocole de bloc/volume quand les clients utilisent des protocoles de bloc/volume.

1.1 Conventions utilisées dans le document

Les mots clés "DOIT", "NE DOIT PAS", "EXIGE", "DEVRA", "NE DEVRA PAS", "DEVRAIT", "NE DEVRAIT PAS", "RECOMMANDE", "PEUT", et "FACULTATIF" en majuscules dans ce document sont à interpréter comme décrit dans le BCP 14, [RFC2119].

1.2 Définitions générales

Les définitions suivantes sont fournies pour donner un contexte approprié au lecteur.

Octet : le présent document définit un octet comme un élément de données d'exactement 8 bits.

Client : le "client" est l'entité qui accède aux ressources du serveur NFS. Le client peut être une application qui contient la logique pour accéder directement au serveur NFS. Le client peut aussi être le client traditionnel de système d'exploitation qui fournit les services de système de fichier distant pour un ensemble d'applications.

Serveur : le "serveur" est l'entité chargée de coordonner l'accès du client à un ensemble de systèmes de fichiers et est identifié par un propriétaire de serveur.

1.3 Notice de licence de composants de code

La description en représentation de données externes (XDR, *external data representation*) et les scripts pour extraire la description XDR sont des composants de code comme décrit au paragraphe 4 des "Dispositions légales relatives aux documents de l'IETF" [LEGAL]. Ces composants de code sont soumis à licence selon les termes de la Section 4 des "Dispositions légales relatives aux documents de l'IETF".

1.4 Description XDR

Le présent document contient la description XDR ([RFC4506]) du protocole de disposition de bloc NFSv4.1. La description XDR est incorporée dans ce document d'une façon qui rend simple pour le lecteur de l'extraire sous une forme prête à compiler. Le lecteur peut charger ce document dans le script suivant pour produire la description XDR lisible par la machine de la disposition de bloc NFSv4.1 :

```
#!/bin/sh
grep '^ *///' $* | sed 's?^ */// ??' | sed 's?^  *///$??'
```

C'est-à-dire, si le script ci-dessus est mémorisé dans un fichier appelé "extract.sh", et si ce document est dans un fichier appelé "spec.txt", alors le lecteur peut faire :

```
sh extract.sh < spec.txt > nfs4_block_layout_spec.x
```

L'effet du script est de supprimer à la fois les espaces en tête et une séquence sentinelle de "///" de chaque ligne correspondante.

L'en-tête de fichier XDR incorporé suit, avec les éléments incorporés suivants à travers le document :

```
/// /*
///  * Ce code est déduit de la RFC 5663.
///  * Prière de reproduire cettte note si possible.
///  */
```



```
///
/// /*
///  *      nfs4_block_layout_prot.x
///  */
///
/// %#include "nfsv41.h"
///
```

Le code XDR contenu dans ce document dépend des types provenant du fichier nfsv41.x. Cela inclut les types nfs qui se terminent par un 4, comme offset4, length4, etc., ainsi que des types plus génériques comme uint32_t et uint64_t.

2. Description de la disposition de bloc

2.1 Fondements et architecture

L'abstraction de mémorisation fondamentale prise en charge par la mémorisation de bloc/volume est un volume de mémorisation consistant en une série séquentielle de blocs de taille fixe. Cela peut être vu comme un disque logique ; cela peut être réalisé par le système de mémorisation comme un disque physique, une portion d'un disque physique, ou quelque chose de plus complexe (par exemple, un enchaînement, une suppression, RAID, et leurs combinaisons) impliquant plusieurs disques physiques ou des portions de ceux-ci.

Une disposition de pNFS pour cette classe de mémorisation de bloc/volume est responsable de la transposition d'un fichier NFS (ou portion d'un fichier) en les volumes de blocs de mémorisation qui contiennent le fichier. Les blocs sont exprimés comme des étendues avec des décalages de 64 bits et des longueurs utilisant les types NFSv4 offset4 et length4 existants. Les clients doivent être capables d'effectuer une entrée/sortie sur les étendues de bloc sans affecter de zone supplémentaire de mémorisation (particulièrement important pour l'écrit) ; donc, les étendues DOIVENT être alignées sur des limites de 512 octets, et les étendues écrites DOIVENT être alignées sur la taille de bloc utilisée par le serveur NFSv4 pour gérer le système de fichiers réel (4 kilo octets et 8 kilo octets sont les tailles de bloc courantes). Cette taille de bloc est disponible comme attribut NFSv4.1 layout_blksize [RFC5661]. Les étendues lisibles DEVRAIENT être alignées sur la taille de bloc utilisée par le serveur NFSv4, mais afin de prendre en charge les systèmes de fichiers traditionnels avec des fragments, l'alignement sur des limites de 512 octets est acceptable.

L'opération pNFS pour demander une disposition (LAYOUTGET) inclut l'argument "layoutiomode4 loga_iomode", qui indique si la disposition demandée est pour un usage en lecture seule ou un usage en lecture écriture. Une disposition en lecteur seule peut contenir des trous qui sont lus comme zéro, tandis qu'une disposition en lecture écriture va contenir la mémorisation allouée, mais non initialisée dans ces trous (lus comme zéro, peuvent être écrits par le client). Le présent document prend aussi en charge la participation des clients dans la copie sur écriture (par exemple, pour les systèmes de fichiers avec des photographies) en fournissant une mémorisation à la fois en lecture seule et non initialisée pour la même gamme dans une disposition. Les lectures sont initialement effectuées sur la mémorisation en lecture seule, avec l'écrit qui va à la mémorisation non initialisée. Après la première écriture qui initialise la mémorisation non initialisée, toutes les lectures sont effectuées sur cette mémorisation inscriptible maintenant initialisée, et la mémorisation en lecture seule correspondante n'est plus utilisée.

La solution de disposition de bloc/volume étend les responsabilité de sécurité des clients pNFS, et il y a un certain nombre d'environnements où les propriétés de sécurité de mise en œuvre obligatoires pour NFS ne peuvent pas être satisfaites. Les responsabilités de sécurité supplémentaire du client suivent, et une discussion complète est présente à la Section 3, "Considérations sur la sécurité".

- Normalement, les dispositifs de disques de réseau de zone de mémorisation (SAN, *storage area network*) et les protocole de SAN fournissent des mécanismes de contrôle d'accès (par exemple, la transposition et/ou masquage du numéro d'unité logique (LUN, *Logical Unit Number*)) qui opèrent à la granularité des hôtes individuels, et non des blocs individuels. Pour cette raison, la protection fondée sur le bloc doit être fournie par le logiciel client.

- De même, les dispositifs de disques SAN et les protocoles SAN ne sont normalement pas capables de valider les verrous NFS qui s'appliquent aux régions de fichiers. Par exemple, si un fichier est couvert par un verrou de lecture seule obligatoire, le serveur peut s'assurer que seulement des dispositions lisibles pour le fichier sont accordées aux clients pNFS. Cependant, il appartient à chaque client pNFS de s'assurer que la disposition lisible est utilisé seulement pour servir des demandes de lecture, et ne pas permettre d'écrire sur les parties existantes du fichier.

Comme les systèmes de mémorisation de bloc/volume ne sont généralement pas capables d'appliquer une telle sécurité fondée sur le fichier, dans les environnements où les clients pNFS ne peuvent pas être de confiance pour appliquer de telles politiques, les dispositions de mémorisation de bloc/volume pNFS NE DEVRAIENT PAS être utilisées.

2.2 GETDEVICELIST et GETDEVICEINFO

2.2.1 Identification de volume

Les systèmes de mémorisation comme les dispositifs de mémorisation peuvent avoir plusieurs accès de réseau physique qui n'ont pas besoin d'être connectés à un réseau commun, résultant en un client pNFS qui a des accès simultanés multi-chemins aux mêmes volumes de mémorisation via des accès différents sur différents réseaux.

Les réseaux peuvent même n'être pas de la même technologie -- par exemple, l'accès au même volume via iSCSI et canal fibre est possible, donc les adresses réseau sont difficiles à utiliser pour l'identification de volume. Pour cette raison, cette disposition de bloc pNFS identifie des volumes de mémorisation par contenu, par exemple en fournissant les moyens de confronter les étiquettes (des portions uniques d'étiquettes) utilisées par les gestionnaires de volume. L'identification de volume est effectuée en confrontant une ou plusieurs séquences d'octets opaques à des parties spécifiques des données mémorisées. Tout système de bloc pNFS utilisant cette disposition DOIT prendre en charge un moyen d'identification de volume unique fondé sur le contenu qui peut être employé via la structure des données donnée ici.

```
/// struct pnfs_block_sig_component4 {          /* composant de signature de disque */
///     int64_t bsc_sig_offset;                 /* décalage d'octets du composant sur le volume*/
///     opaque  bsc_contents<>;                 /* contenu de ce composant de la signature */
/// };
///
```

Noter que le champ opaque "bsc_contents" dans la structure "pnfs_block_sig_component4" NE DOIT PAS être interprété comme une chaîne terminée par zéro, car il peut contenir des valeurs incorporées de zéros. Il n'y a pas de restriction sur l'alignement (par exemple, il n'est exigé ni de bsc_sig_offset ni de la longueur qu'ils soient des multiples de 4). Le bsc_sig_offset est une quantité signée, qui, quand elle est positive, représente un décalage d'octet depuis le début du volume, et quand elle est négative représente un décalage d'octet depuis la fin du volume.

Les décalages négatifs sont permis afin de simplifier la mise en œuvre de client sur les systèmes où l'étiquette d'appareil est trouvée à un décalage fixe depuis la fin du volume. Si le serveur utilise des décalages négatifs pour décrire la signature, le client et le serveur NE DOIVENT PAS voir alors des tailles de volume différentes. Les décalages négatifs NE DEVRAIENT PAS être utilisés dans des systèmes qui redéfinissent dynamiquement la taille des volumes sauf à veiller à s'assurer que l'étiquette d'appareil est toujours présente au décalage depuis la fin du volume tel que vu par le client.

Une signature est un dispositif de jusqu'à "PNFS_BLOCK_MAX_SIG_COMP" (défini ci-dessous) composants de signature. Le client NE DOIT PAS supposer que tous les composants de signature sont co-situés au sein d'un seul secteur sur un appareil de bloc.

Le pilote de disposition de bloc de client pNFS utilise cette identification de volume pour transposer les identifiants d'appareil pnfs_block_volume_type4 PNFS_BLOCK_VOLUME_SIMPLE en sa vue locale d'un LUN.

2.2.2 Topologie de volume

La topologie de volume du serveur de bloc pNFS est exprimée par une combinaison arbitraire de types de volume de base énumérés dans les structures de données suivantes. Les composants individuels de la topologie sont contenus dans une matrice et les composants peuvent se référer aux autres composants en utilisant des indices matriciels.

```
/// enum pnfs_block_volume_type4 {
///     PNFS_BLOCK_VOLUME_SIMPLE = 0,           /* volume se transpose en une seule LU */
///     PNFS_BLOCK_VOLUME_SLICE  = 1,           /* volume est une tranche d'un autre volume */
///     PNFS_BLOCK_VOLUME_CONCAT = 2,           /* volume est un enchaînemen de plusieurs volumes */
///     PNFS_BLOCK_VOLUME_STRIPE = 3            /* volume est étalé sur plusieurs volumes */
/// };
///
/// const PNFS_BLOCK_MAX_SIG_COMP = 16;         /* maximum de composants par signature */
/// struct pnfs_block_simple_volume_info4 {
///     pnfs_block_sig_component4 bsv_ds<PNFS_BLOCK_MAX_SIG_COMP>;
///                                             /* signature du disque */
///
/// };
///
/// struct pnfs_block_slice_volume_info4 {
///     offset4  bsv_start;                     /* décalage depuis le début de la tranche en octets */
///     length4  bsv_length;                    /* longueur de la tranche en octets */
///     uint32_t bsv_volume;                    /* indice matriciel du volume étalé */
/// };
///
/// struct pnfs_block_concat_volume_info4 {
```

```
///     uint32_t  bcv_volumes<>;                    /* indices matriciels des volumes enchaînés */
/// };
///
/// struct pnfs_block_stripe_volume_info4 {
///     length4  bsv_stripe_unit;                   /* taille de surface en octets */
///     uint32_t bsv_volumes<>;                     /* indices matriciels des volumes étalés -- DOIT être la même taille */
/// };
///
/// union pnfs_block_volume4 switch (pnfs_block_volume_type4 type) {
///     case PNFS_BLOCK_VOLUME_SIMPLE:
///         pnfs_block_simple_volume_info4 bv_simple_info;
///     case PNFS_BLOCK_VOLUME_SLICE:
///         pnfs_block_slice_volume_info4 bv_slice_info;
///     case PNFS_BLOCK_VOLUME_CONCAT:
///         pnfs_block_concat_volume_info4 bv_concat_info;
///     case PNFS_BLOCK_VOLUME_STRIPE:
///         pnfs_block_stripe_volume_info4 bv_stripe_info;
/// };
///
/// /* type spécifique de disposition de bloc pour da_addr_body */
/// struct pnfs_block_deviceaddr4 {
///     pnfs_block_volume4 bda_volumes<>;           /* matrice de volumes */
/// };
///
```

La structure de données "pnfs_block_deviceaddr4" permet de coder des structures de volume incorporées de complexité arbitraire. Les types d'agrégations qui sont permis sont des surfaces, des enchaînements, et des tranches. Noter que la topologie de volume exprimée dans la structure de données pnfs_block_deviceaddr4 va toujours se résoudre en un ensemble PNFS_BLOCK_VOLUME_SIMPLE de pnfs_block_volume_type4. La matrice de volumes est ordonnée de façon que la racine de la hiérarchie de volumes soit le dernier élément de la matrice. Les volumes d'enchaînement, de tranche, et de surface DOIVENT se référer aux volumes définis par les éléments d'indice inférieur de la matrice.

La structure de données "pnfs_block_device_addr4" est retournée par le serveur comme champ opaque spécifique du protocole de mémorisation da_addr_body dans la structure "device_addr4" par une opération GETDEVICEINFO réussie [RFC5661].

Comme noté ci-dessus, toutes les structures device_addr4 se résolvent finalement en un ensemble de volumes de type PNFS_BLOCK_VOLUME_SIMPLE. Ces volumes sont chacun identifiés de façon univoque par un ensemble de composants de signature. Des hiérarchies de volumes compliquées peuvent être composées de douzaines de volumes, chacun avec plusieurs composants de signature ; donc, l'adresse de l'appareil peut exiger plusieurs kilo octets. Le client DEVRAIT être prêt à allouer une grande mémoire tampon pour contenir le résultat. Dans le cas où le serveur retourne NFS4ERR_TOOSMALL, le client DEVRAIT allouer une mémoire tampon d'au moins gdir_mincount_bytes pour contenir le résultat attendu et réessayer la demande GETDEVICEINFO.

2.2.3 GETDEVICELIST et GETDEVICEINFO deviceid4

En réponse à une demande GETDEVICELIST, le serveur va normalement retourner un seul "deviceid4" dans le dispositif gdlr_deviceid_list. C'est parce que le deviceid4, quand il est passé à GETDEVICEINFO va retourner une "device_addr4", qui code toute la hiérarchie de volume. Dans le cas de systèmes de fichiers en copie sur écriture, le dispositif "gdlr_deviceid_list" peut contenir deux deviceid4, un se référant à la hiérarchie de volume en lecture seule, et un se référant à la hiérarchie de volume inscriptible. Il n'y a pas d'ordre exigé des identifiants lisibles et inscriptibles dans le dispositif car les volumes sont identifiés de façon univoque par leur deviceid4, et sont référencés par les dispositions en utilisant le deviceid4. Un autre exemple de serveur qui retourne plusieurs éléments d'appareil se produit quand la bride de fichier représente la racine d'un espace de noms s'étendant sur plusieurs systèmes de fichiers physiques sur le serveur, chacun avec une hiérarchie de volume différente. Dans cet exemple, une mise en œuvre de serveur peut retourner une liste d'identifiants d'appareil utilisés par chaque système de fichiers physiques, ou peut retourner une liste vide.

Chaque deviceid4 retourné par une opération GETDEVICELIST réussie est un identifiant abrégé utilisé pour référencer la topologie de volume globale. Ces identifiants d'appareil, ainsi que les identifiants d'appareil retournés dans les extensions d'une opération LAYOUTGET, peuvent être utilisés comme entrée de l'opération GETDEVICEINFO. Décoder le résultat

"pnfs_block_deviceaddr4" dans une ordre plat de blocs de données transposés en volumes PNFS_BLOCK_VOLUME_SIMPLE. Combiné avec la transposition en un client LUN décrite au paragraphe 2.2.1 "Identification de volume", un décalage de volume logique peut être transposé en un bloc sur un client LUN pNFS [RFC5661].

2.3 Structures de données : listes de Extents et Extent

Une disposition de blocs pNFS est une liste d'extensions au sein d'un dispositif plat de blocs de données dans un volume logique. Les détails de la topologie de volume peuvent être déterminés en utilisant l'opération GETDEVICEINFO (voir la discussion de l'identification de volume, au paragraphe 2.2). La disposition de bloc décrit les extensions de bloc individuel sur le volume qui constitue le fichier. Les décalages et la longueur contenus dans une extension sont spécifiés en unités d'octets.

```
/// enum pnfs_block_extent_state4 {
///     PNFS_BLOCK_READ_WRITE_DATA = 0,     /* les données situées par cette extension sont valides en lecture et
                                               en écriture. */
///     PNFS_BLOCK_READ_DATA = 1,           /* les données seulement ; elle ne peuvent pas être écrites. */
///     PNFS_BLOCK_INVALID_DATA = 2,        /* la localisation est valide ; les données sont invalides. C'est une
                                               nouvelle extension (pré) allouée. Il y a un espace physique sur le
                                               volume. */
///     PNFS_BLOCK_NONE_DATA  = 3           /* la localisation est invalide. Il y a un trou dans le fichier. Il n'y a
                                               pas d'espace physique dans le volume. */
/// };
///
/// struct pnfs_block_extent4 {
///     deviceid4    bex_vol_id;            /* identifiant de volume logique sur lequel l'extension de fichier
                                               est mémorisée. */
///     offset4      bex_file_offset;       /* décalage d'octets depuis le début du fichier */
///     length4      bex_length;            /* taille en octets de l'extension */
///     offset4      bex_storage_offset;    /* décalage en octets depuis le début du volume */
///     pnfs_block_extent_state4 bex_state; /* état de cette extension */
/// };
///
/// /* type de disposition de bloc spécifique pour loc_body */
/// struct pnfs_block_layout4 {
///     pnfs_block_extent4 blo_extents<>;   /* extensions qui constituent cette disposition. */
/// };
///
```

La disposition de bloc consiste en une liste d'extensions qui transposent les régions logiques du fichier en localisations physiques sur un volume. Le champ "bex_storage_offset" au sein de chaque extension identifie une localisation sur le volume logique spécifié par le champ "bex_vol_id" dans l'extension. Le bex_vol_id lui-même est abrégé pour toute la topologie du volume logique sur lequel le fichier est mémorisé. Le client est responsable de la traduction de ce décalage logique en un décalage sur l'unité logique SAN sous-jacente appropriée. Dans la plupart des cas, toutes les extension dans une disposition vont résider sur le même volume et donc avoir le même bex_vol_id. Dans le cas de systèmes de fichiers de copie sur écriture, l'extension PNFS_BLOCK_READ_DATA peut avoir un bex_vol_id différent des extensions inscriptibles.

Chaque extension transpose une région logique du fichier sur une portion du volume logique spécifié. Les champs bex_file_offset, bex_length, et bex_state pour une extension retournée du serveur sont valides pour toutes les extensions. À l'opposé, l'interprétation du champ bex_storage_offset dépend de la valeur de bex_state comme suit (en ordre croissant) :

- PNFS_BLOCK_READ_WRITE_DATA signifie que bex_storage_offset est valide, et pointe sur des données valides/initialisées qui peuvent être lues et écrites.

- PNFS_BLOCK_READ_DATA signifie que bex_storage_offset est valide, et pointe sur des données valides/initialisées qui peuvent seulement être lues. Les opérations d'écriture sont interdites ; le client peut devoir demander une disposition de lecture-écriture.

- PNFS_BLOCK_INVALID_DATA signifie que bex_storage_offset est valide, mais pointe sur des données invalides

non initialisées. Ces données ne doivent pas être physiquement lues à partir du disque avant d'avoir été initialisées. Une demande de lecture pour une extension PNFS_BLOCK_INVALID_DATA doit remplir la mémoire tampon de l'utilisateur avec des zéros, sauf si l'extension est couverte par une extension PNFS_BLOCK_READ_DATA d'un système de fichier en copie sur écriture. Les demandes d'écriture doivent écrire tous les blocs de taille de serveur sur le disque ; les octets non initialisés par l'utilisateur doivent être réglés à zéro. Toute écriture en mémorisation dans une extension PNFS_BLOCK_INVALID_DATA change la portion écrite de l'extension en PNFS_BLOCK_READ_WRITE_DATA ; le client pNFS est chargé de rapporter ce changement via LAYOUTCOMMIT.

- PNFS_BLOCK_NONE_DATA signifie que bex_storage_offset n'est pas valide, et que cette extension ne peut pas être utilisée pour satisfaire des demandes d'écriture. Les demandes de lecture peuvent être satisfaites par un remplissage de zéros comme pour PNFS_BLOCK_INVALID_DATA. Les extensions PNFS_BLOCK_NONE_DATA peuvent être retournées par des demandes pour des extensions lisibles ; elles ne sont jamais retournées si la demande était pour une extension inscriptible.

Une liste d'extensions contient toutes les extensions pertinentes en ordre croissant du bex_file_offset de chaque extension ; tous les liens sont rompus en ordre croissant de l'état d'extension (bex_state).

2.3.1 Demande de disposition et listes de Extent

Chaque demande d'une disposition spécifie au moins trois paramètres : décalage de fichier, taille désirée, et taille minimum. Si l'état d'une demande indique le succès, la liste d'extensions retournée doit satisfaire les critères suivants :

- Une demande pour une disposition lisible (mais pas inscriptible) retourne seulement des extensions PNFS_BLOCK_READ_DATA ou PNFS_BLOCK_NONE_DATA (mais pas PNFS_BLOCK_INVALID_DATA ou PNFS_BLOCK_READ_WRITE_DATA).

- Une demande pour une disposition inscriptible retourne des extensions PNFS_BLOCK_READ_WRITE_DATA ou PNFS_BLOCK_INVALID_DATA (mais pas des extensions PNFS_BLOCK_NONE_DATA). Elle peut aussi ne retourner des extensions PNFS_BLOCK_READ_DATA que quand les gammes de décalage dans ces extensions sont aussi couvertes par des extensions PNFS_BLOCK_INVALID_DATA pour permettre les écritures.

- La première extension dans la liste DOIT contenir le décalage demandé par rapport au début.

- La taille totale des extensions au sein de la gamme demandée DOIT couvrir au moins la taille minimum. Une exception est permise : la taille totale PEUT être plus petite si seulement des extensions lisibles étaient demandées et si le EOF est rencontré.

- Les extensions dans la liste des extensions DOIVENT être logiquement contiguës pour la disposition en lecture seule. Pour une disposition en lecture-écriture, l'ensemble de contenus inscriptibles (c'est-à-dire, en excluant les extensions PNFS_BLOCK_READ_DATA) DOIT être logiquement contigu. Chaque extension PNFS_BLOCK_READ_DATA dans une disposition en lecture-écriture DOIT être couverte par une ou plusieurs extensions PNFS_BLOCK_INVALID_DATA. Ce chevauchement des extensions PNFS_BLOCK_READ_DATA et PNFS_BLOCK_INVALID_DATA est le seul chevauchement d'extensions permis.

- Les extensions DOIVENT être ordonnées dans la liste par décalage par rapport au début, avec les extensions PNFS_BLOCK_READ_DATA qui précèdent les extensions PNFS_BLOCK_INVALID_DATA dans le cas de bex_file_offsets égaux.

Si la taille minimum demandée, loga_minlength, est zéro, c'est l'indication au serveur de métadonnées que le client désire toute disposition au décalage loga_offset ou moins que le serveur de métadonnées a "directement disponible". Directement est subjectif, et dépend du type de disposition et de la mise en œuvre de serveur pNFS. Pour les serveurs de disposition de bloc, directement disponible DEVRAIT être interprété de telle façon que les dispositions lisibles soient toujours disponibles, même si certaines extensions sont dans l'état PNFS_BLOCK_NONE_DATA. Quand on traite des demandes pour des dispositions inscriptibles, une disposition est directement disponible si des extensions peuvent être retournées dans l'état PNFS_BLOCK_READ_WRITE_DATA.

2.3.2 Engagements de dispositin

```
/// /* type spécifique de disposition de bloc pour lou_body */
/// struct pnfs_block_layoutupdate4 {
///     pnfs_block_extent4 blu_commit_list<>;
///                             /* liste des extensions qui contiennent maintenant des données valides. */
/// };
///
```

La structure "pnfs_block_layoutupdate4" est utilisée par le client comme argument spécifique du protocole de bloc dans l'opération LAYOUTCOMMIT. Le champ "blu_commit_list" est une liste d'extensions qui couvre les régions de la disposition de fichier qui était précédemment dans l'état PNFS_BLOCK_INVALID_DATA, mais a été écrite par le client et devrait être maintenant considérée être dans l'état PNFS_BLOCK_READ_WRITE_DATA. Le champ bex_state de chaque extension dans la blu_commit_list DOIT être réglé à PNFS_BLOCK_READ_WRITE_DATA. Les extensions dans la liste d'engagements DOIVENT être disjointes et DOIVENT être triées par bex_file_offset. Le champ bex_storage_offset est inutilisé. Les mises en œuvre devraient savoir que un serveur peut être incapable d'engager des régions à une granularité inférieure au bloc de système de fichier (normalement 4 kB ou 8 kB). Comme noté ci-dessus, la taille de bloc que le serveur utilise est disponible comme un attribut NFSv4, et toutes les extensions incluses dans la "blu_commit_list" DOIVENT être alignées sur cette granularité et ont une taille qui est un multiple de cette granularité. Si le client croit que ses actions ont déplacé la fin de fichier au milieu d'un bloc engagé, il DOIT écrire des zéros depuis la fin de fichier jusqu'à la fin de ce bloc avant d'engager le bloc. Manquer à le faire peut résulter en détritus (données non initialisées) apparaissant dans cette zone si le fichier est ensuite étendu en déplaçant la fin de fichier.

2.3.3 Retours de disposition

L'opération LAYOUTRETURN est faite sans aucune données spécifiques de disposition de bloc. Quand l'opération LAYOUTRETURN spécifie un type LAYOUTRETURN4_FILE_return, alors la structure de données layoutreturn_file4 spécifie la région de la disposition de fichier qui n'est plus nécessaire au client. Le champ opaque "lrf_body" de la structure de données "layoutreturn_file4" DOIT avoir une longueur de zéro. Une opération LAYOUTRETURN représente une libération explicite des ressources par le client, généralement faite dans le but d'éviter à l'avenir des opérations CB_LAYOUTRECALL inutiles. Le client peut retourner des régions disjointes du fichier en utilisant plusieurs opérations LAYOUTRETURN au sein d'une seule opération COMPOUND.

Noter que la disposition de bloc/volume prend en charge la révocation unilatérale de disposition. Quand une disposition est révoquée unilatéralement par le serveur, généralement due à l'expiration du temps de prêt du client, ou du rappel d'une délégation, ou du client qui manque à retourner une disposition en temps utile, il est important pour que tout soit correct que toutes les entrées/sorties en cours que le client a produites avant la révocation de la disposition soient rejetées de la mémorisation. Pour le protocole de bloc/volume, ceci est possible en excluant un client avec un temporisateur de disposition expiré de la mémorisation physique. Noter, cependant, que la granularité de cette opération peut seulement être au niveau de l'unité logique de l'hôte. Donc, si une des dispositions d'un client est révoquée unilatéralement par le serveur, cela va effectivement rendre inutiles *toutes* les dispositions du client pour les fichiers situés sur les unités de mémorisation composant le volume logique. Cela peut rendre inutiles les dispositions du client pour des fichiers dans d'autres systèmes de fichiers.

2.3.4 Traitement de copie sur écriture par le client

La copie sur écriture est un mécanisme utilisé pour prendre en charge des photographies de fichier et/ou de système de fichiers. Quand il écrit sur des régions non alignées, ou des régions plus petites qu'un bloc de système de fichier, le rédacteur doit copier les portions des données du fichier original à une nouvelle localisation sur le disque. Ce comportement peut être mis en œuvre sur le client ou sur le serveur. Les paragraphes qui suivent décrivent comment un client de disposition de bloc pNFS met en œuvre l'accès à un fichier qui exige une sémantique de copie sur écriture.

Distinguer les types d'extension PNFS_BLOCK_READ_WRITE_DATA et PNFS_BLOCK_READ_DATA en combinaison avec le chevauchement permis des extensions PNFS_BLOCK_READ_DATA avec les extensions PNFS_BLOCK_INVALID_DATA permet que le traitement de copie sur écriture soit fait par les clients pNFS. Dans le NFS classique, cette opération serait faite par le serveur. Comme pNFS permet aux clients de faire un accès direct au bloc, il est utile pour les clients de participer aux opérations de copie sur écriture. Tous les clients de bloc/volume pNFS DOIVENT prendre en charge ce traitement de copie sur écriture.

Quand un client souhaite écrire des données couvertes par une extension PNFS_BLOCK_READ_DATA, il DOIT avoir

demandé une disposition inscriptible au serveur ; cette disposition va contenir des extensions PNFS_BLOCK_INVALID_DATA pour couvrir toutes les gammes de données des extensions PNFS_BLOCK_READ_DATA de cette disposition. Plus précisément, pour toute gamme bex_file_offset couverte par une ou plusieurs extensions PNFS_BLOCK_READ_DATA dans une disposition inscriptible, le serveur DOIT inclure une ou plusieurs extensions PNFS_BLOCK_INVALID_DATA dans la disposition qui couvre la même gamme bex_file_offset. Quand il effectue une écriture sur une telle zone d'une disposition, le client DOIT effectivement copier les données provenant de l'extension PNFS_BLOCK_READ_DATA pour tous les blocs partiels de bex_file_offset et de gamme, fusionner les changements à écrire, et écrire le résultat dans l'extension PNFS_BLOCK_INVALID_DATA pour les blocs de ce bex_file_offset et gamme. C'est-à-dire, si des blocs entiers de données sont à écraser par une opération, il n'est pas besoin d'aller chercher les blocs PNFS_BLOCK_READ_DATA correspondants, mais tout bloc partiel écrit doit être fusionné avec les données apportées via les extensions PNFS_BLOCK_READ_DATA avant de mémoriser le résultat via des extensions PNFS_BLOCK_INVALID_DATA. Pour les besoins de cette discussion, "blocs entiers" et "blocs partiels" se rèfèrent à la taille de bloc du système de fichiers du serveur. La mémorisation des données dans une extension PNFS_BLOCK_INVALID_DATA convertit la portion écrite de l'extension PNFS_BLOCK_INVALID_DATA en une extension PNFS_BLOCK_READ_WRITE_DATA ; toutes les lectures suivantes DOIVENT être effectuées à partir de cette extension ; la portion correspondante de l'extension PNFS_BLOCK_READ_DATA NE DOIT PAS être utilisée après la mémorisation des données dans une extension PNFS_BLOCK_INVALID_DATA. Si un client écrit seulement une portion d'une extension, l'extension peut être partagée aux frontières d'alignement de bloc.

Quand un client souhaite écrire des données dans une extension PNFS_BLOCK_INVALID_DATA qui n'est pas couverte par une extension PNFS_BLOCK_READ_DATA, il DOIT traiter cette écriture comme une écriture sur un fichier non impliqué par une sémantique de copie sur écriture. Donc, les données doivent être écrites dans des incréments d'au moins la taille de bloc, alignés sur des multiples de décalages de taille de bloc, et les portions non écrites de blocs doivent être remplies de zéros.

Dans l'opération LAYOUTCOMMIT qui renvoie normalement des informations de disposition mises à jour au serveur, pour les données inscriptibles, certaines extensions PNFS_BLOCK_INVALID_DATA peuvent être engagées comme des extensions PNFS_BLOCK_READ_WRITE_DATA, ce qui signifie que la mémorisation aux valeurs correspondantes de bex_storage_offset y a été mémorisée et qu'elles sont maintenant considérées comme des données valides en lecture. Les extensions PNFS_BLOCK_READ_DATA ne sont pas engagées au serveur. Pour les extensions que le client reçoit via LAYOUTGET comme PNFS_BLOCK_INVALID_DATA et retourne via LAYOUTCOMMIT comme des PNFS_BLOCK_READ_WRITE_DATA, le serveur va comprendre que la transposition PNFS_BLOCK_READ_DATA pour cette extension n'est plus valide ou nécessaire pour ce fichier.

2.3.5 Les extensions sont des permissions

Les extensions de disposition retournées aux clients pNFS accordent la permission de lire ou écrire ; les PNFS_BLOCK_READ_DATA et PNFS_BLOCK_NONE_DATA sont en lecture seule (PNFS_BLOCK_NONE_DATA lit des zéros) PNFS_BLOCK_READ_WRITE_DATA et PNFS_BLOCK_INVALID_DATA sont en lecture/écriture, (PNFS_BLOCK_INVALID_DATA lit des zéros, toute écriture le convertit en PNFS_BLOCK_READ_WRITE_DATA). C'est le seul moyen qu'a un client d'obtenir la permission d'effectuer des entrées/sorties directes sur les appareils de mémorisation ; un client pNFS NE DOIT PAS effectuer d'opérations d'entrées/sorties directes qui ne sont pas permises par une extension détenue par le client. L'adhésion du client à cette règle place le serveur pNFS au contrôle d'opérations d'appareils de mémorisation potentiellement conflictuelles, permettant au serveur de déterminer ce qui constitue le conflit et comment éviter des conflits en accordant et en rappelant les extensions aux clients.

Il n'est pas exigé des appareils de mémorisation de classe de bloc/volume qu'ils effectuent des opérations de lecture et écriture de façon atomique. Des opérations concurrentes de lecture et écriture qui se chevauchent sur les mêmes données peuvent causer le retour d'une lecture d'une mixture de données d'avant l'écriture et d'après l'écriture. Un chevauchement d'opérations d'écriture peut être pire, car le résultat pourrait être une mixture de données provenant des deux opérations d'écriture ; la corruption des données peut arriver si la mémorisation sous-jacente est étalée et si les opérations se terminent dans des ordres différents sur les différentes surfaces. Quand il y a plusieurs clients qui souhaitent accéder aux mêmes données, un serveur pNFS peut éviter ces conflits en mettant en œuvre une politique de contrôle de concurrence d'un seul rédacteur OUX plusieurs lecteurs. Cette politique DOIT être mise en œuvre quand les appareils de mémorisation ne fournissent pas l'atomicité pour les opérations concurrente de lecture/écriture et d'écriture/écriture sur les mêmes données.

Si un client fait une demande de disposition qui entre en conflit avec une délégation existante de disposition, la demande va être rejetée avec l'erreur NFS4ERR_LAYOUTTRYLATER. Ce client est alors supposé réessayer la demande après un court intervalle. Durant cet intervalle, le serveur DEVRAIT rappeler la portion en conflit de la délégation de disposition du client qui la détient actuellement. Cette approche de rejet et ré-essai n'empêche pas la famine du client quand il y a rétention

de la disposition d'un fichier particulier. Pour cette raison, un serveur pNFS DEVRAIT mettre en œuvre un mécanisme pour empêcher la famine. Une possibilité est que le serveur puisse tenir une file d'attente des demandes de disposition rejetées. Chauqe nouvelle demande de disposition peut être vérifiée pour voir si elle est en conflit avec une demande rejetée précédente, et si il en est ainsi, la demande plus récente peut être rejetée. Une fois que le client demandeur original réessaye sa demande, son entrée dans la file d'attente de demandes rejetées peut être éliminée, ou l'entrée dans la file d'attente de demandes rejetées peut être retirée quand elle atteint un certain âge.

NFSv4 prend en charge les verrous obligatoires et les réservations partagées. Ce sont des mécanismes que les clients peuvent utiliser pour restreindre l'ensemble des opérations d'entrée/sortie qui sont permises aux autres clients. Comme toutes les opérations d'entrée/sortie arrivent finalement au serveur NFSv4 pour traitement, le serveur est en position d'appliquer ces restrictions. Cependant, avec les dispositions pNFS, les entrées/sortie vont être produites à partir de clients qui détiennent directement les dispositions aux appareils de mémorisation qui hébergent les données. Ces appareils n'ont pas connaissance des fichiers, des verrous obligatoires, ou des réservations partagées, et ne sont pas en position d'appliquer ces restrictions. Pour cette raison le serveur NFSv4 NE DOIT PAS accorder des dispositions qui entrent en conflit avec des verrous obligatoires ou des réservations partagées. De plus, si une demande de verrou obligatoire en conflit ou une demande ouverte en conflit arrive au serveur, le serveur DOIT rappeler la partie de la disposition en conflit avec la demande avant d'accorder la demande.

2.3.6 Traitement de fin de fichier

La localisation de fin de fichier peut être changée de deux façons : implicitement par suite d'un WRITE ou LAYOUTCOMMIT au delà de la fin de fichier actuelle, ou explicitement par suite d'une demande SETATTR. Normalement, quand un fichier est tronqué par un client NFSv4 via l'invocation de SETATTR, le serveur libère tous les blocs de disque qui appartiennent au fichier et sont au-delà du nouvel octet de fin de fichier, et DOIT écrire des zéros à la portion du nouveau bloc de fin de fichier au delà du nouvel octet de fin de fichier. Ces actions rendent sémantiquement invalides toutes les dispositions pNFS qui se réfèrent aux blocs qui sont libérés ou écrits. Donc, le serveur DOIT rappeler des clients les portions de toute disposition pNFS qui se réfère aux blocs qui vont être libérés ou écrits par le serveur avant de traiter la demande tronquée. Ces rappels peuvent prendre du temps à se faire; comme expliqué dans la [RFC5661], si le serveur ne peut pas répondre à la demande SETATTR du client dans un délai raisonnable, il DEVRAIT répondre au client avec l'erreur NFS4ERR_DELAY.

Les blocs dans l'état PNFS_BLOCK_INVALID_DATA qui se trouvent au delà du nouveau bloc de fin de fichier présentent un cas particulier. Le serveur a réservé ces blocs pour qu'ils soient utilisés par un client pNFS avec une disposition inscriptible pour le fichier, mais le client a déjà engagé les blocs, et ils ne font pas encore partie de la transposition du fichier sur le disque. Le serveur PEUT libérer ces blocs quand il traite la demande SETATTR. Si il en est ainsi, le serveur DOIT rappeler toutes les dispositions des clients pNFS qui se réfèrent aux blocs avant de traiter la troncature. Si le serveur ne libère pas les blocs PNFS_BLOCK_INVALID_DATA lorsque il traite la demande SETATTR, il n'a pas besoin de rappeler les dispositions qui se réfèrent seulement aux blocs PNFS_BLOCK_INVALID_DATA.

Quand un fichier est étendu implicitement par un WRITE ou LAYOUTCOMMIT au-delà de la fin de fichier courante, ou étendu explicitement par une demande SETATTR, le serveur n'a besoin de rappeler de portions d'aucune disposition pNFS.

2.3.7 Indications de disposition

L'opération SETATTR prend en charge un attribut Indication de disposition [RFC5661]. Quand le client établit une indication de disposition (type de données layouthint4) avec un type de disposition de LAYOUT4_BLOCK_VOLUME (champ loh_type) le champ loh_body contient une valeur de type de données de pnfs_block_layouthint4.

```
/// /* type spécifique de disposition de bloc pour loh_body */
/// struct pnfs_block_layouthint4 {
///     uint64_t blh_maximum_io_time;                /* temps maximum d'entrée/sortie en secondes */
/// };
///
```

Le client de disposition de bloc utilise la structure de données d'indication de disposition pour communiquer au serveur le temps maximum que peut prendre une entrée/sortie pour s'exécuter sur le client. Les clients qui utilisent des dispositions de bloc DOIVENT établir l'attribut Indication de disposition avant d'utiliser les opérations LAYOUTGET.

2.3.8 Barrage de client

Le protocole de bloc pNFS doit traiter des situations dans lesquelles une défaillance de système, normalement un problème de connexité réseau, exige que le serveur révoque unilatéralement les extensions d'un client afin de transférer les extensions à un autre client. La mise en œuvre de serveur pNFS DOIT s'assurer que quand des ressources sont transférées à un autre client, elles ne sont plus utilisées par le client qui les possédait à l'origine, et il doit s'en assurer contre toutes les combinaisons possible de partitions et délais parmi tous les participants au protocole (serveur, mémorisation et client). Deux approches pour garantir cet isolement sont possibles et sont discutées ci-dessous.

Un choix de mise en œuvre pour établir une barrière entre le client de bloc et la mémorisation de bloc est l'utilisation du masquage de LUN ou la transposition aux systèmes de mémorisation ou au réseau de zone de mémorisation pour désactiver l'accès par le client à isoler. Cela exige l'accès du serveur à une interface de gestion pour le système de mémorisation et l'autorisation d'effectuer le masquage de LUN et les opérations de gestion. Par exemple, la spécification d'initiative de gestion de mémorisation (SMI-S, *Storage Management Initiative Specification*) [SMIS] fournit un moyen pour découvrir et masquer les LUN, incluant un moyen d'associer les clients avec les noms mondiaux ou noms d'initiateur nécessaires à masquer.

En l'absence de prise en charge du masquage de LUN, le serveur doit s'appuyer sur les clients pour mettre en œuvre un mécanisme de barrage d'entrée/sortie de prêt temporaire. Parce que les clients ne savent pas si le serveur utilise le masquage de LUN, dans tous les cas, le client DOIT mettre en œuvre le barrage de prêt temporaire. Dans le barrage de prêt temporaire, on définit deux périodes, la première, "lease_time" est la durée d'un prêt comme défini par l'attribut lease_time du serveur [RFC5661], et le second, "blh_maximum_io_time" est la durée maximum que peuvent prendre les entrées/sorties d'un client pour s'achever ou échouer sur le système de mémorisation ; cette valeur est souvent 30 secondes ou 60 secondes, mais peut être plus longue dans certains environnements. Si la durée maximum d'entrée/sortie de client ne peut pas être limitée, le client DOIT utiliser une valeur toute de uns comme blh_maximum_io_time.

Après l'établissement d'un nouvel identifiant de client, le client DOIT utiliser SETATTR avec une indication de disposition de type LAYOUT4_BLOCK_VOLUME pour informer le serveur de son temps maximum d'entrée/sortie avant de produire la première opération LAYOUTGET. Bien que l'indication de temps maximum d'entrée/sortie soit un attribut par fichier, c'est en fait une caractéristique par client. Donc, le serveur DOIT conserver la dernière indication de temps maximum d'entrée/sortie envoyée séparément pour chaque client. Chaque fois que le temps maximum d'entrée/sortie change, le serveur DOIT l'appliquer à tous les fichiers pour lesquels le client a une disposition. Si le client ne spécifie pas cet attribut sur un fichier pour lequel une disposition de bloc est demandée, le serveur DEVRAIT utiliser la plus récente valeur fournie par le même client pour un fichier ; si ce client n'a pas fourni une valeur pour cet attribut, le serveur DEVRAIT rejeter la demande de disposition avec l'erreur NFS4ERR_LAYOUTUNAVAILABLE. Le client NE DEVRAIT PAS envoyer un SETATTR de l'indication de disposition avec chaque LAYOUTGET. Un serveur qui met en œuvre le barrage via le masquage de LUN DEVRAIT accepter toute valeur maximum de temps d'entrée/sortie provenant d'un client. Un serveur qui ne met pas en œuvre le barrage peut retourner une erreur NFS4ERR_INVAL à l'opération SETATTR. Un tel serveur DEVRAIT retourner NFS4ERR_INVAL quand un client envoie un temps maximum d'entrée/sortie sans limite (tout de uns) ou quand le temps maximum d'entrée/sortie est significativement supérieur à celui des autres clients qui utilisent les dispositions de bloc avec pNFS.

Quand un client reçoit l'erreur NFS4ERR_INVAL en réponse à l'opération SETATTR pour une indication de disposition, le client NE DOIT PAS utiliser l'opération LAYOUTGET. Après avoir répondu avec NFS4ERR_INVAL au SETATTR pour l'indication de disposition, le serveur DOIT retourner l'errEneur NFS4ERR_LAYOUTUNAVAILABLE à toutes les opérations LAYOUTGET suivantes provenant de ce client. Donc, le serveur, en retournant NFS4ERR_INVAL ou NFS4_OK détermine si un client avec un grand temps maximum d'entrée/sortie non limité peut ou non utiliser pNFS.

En utilisant les valeurs de temps de prêt et de temps maximum d'entrée/sortie, on spécifie le comportement du client et du serveur comme suit :

Quand un client reçoit des informations de disposition via une opération LAYOUTGET, ces dispositions sont valides pour au plus "lease_time" secondes à partir du moment où le serveur les a accordées. Une disposition est renouvelée par toute opération SEQUENCE réussie, ou chaque fois qu'un nouvel identifiant d'état est créé ou mis à jour (voir la section "Renouvellement de prêt" de la [RFC5661]). Si le prêt de disposition n'est pas renouvelé avant l'expiration, le client DOIT cesser d'utiliser la disposition après "lease_time" secondes à partir du moment où il a envoyé la commande LAYOUTGET originale ou envoyé la dernière opération renouvelant le prêt. En d'autres termes, le client ne peut pas produire d'entrée/sortie sur les blocs spécifiés par une disposition expirée. En présence de grands délais de communication entre le client et le serveur, il est même possible que le prêt expire avant que la réponse du serveur arrive au client. Dans une telle situation, le client NE DOIT PAS utiliser les dispositions expirées, et DEVRAIT revenir à l'utilisation des opérations READ et WRITE NFSv4.1 standard. De plus, le client doit être configuré de telle façon que les opérations d'entrée/sortie

s'achèvent au sein de "blh_maximum_io_time" même en présence de pilotes multi-chemins qui vont réessayer les entrées/sorties via plusieurs chemins.

Comme indiqué à la section "Traitement de l'expiration du prêt chez le client" de la [RFC5661], si une opération SEQUENCE réussit, mais si le fanion sr_status_flag a SEQ4_STATUS_EXPIRED_ALL_STATE_REVOKED, SEQ4_STATUS_EXPIRED_SOME_STATE_REVOKED, ou si SEQ4_STATUS_ADMIN_STATE_REVOKED est établi, le client DOIT immédiatement cesser d'utiliser toutes les transpositions d'identifiants de disposition et d'appareil en adresses d'appareil associées au serveur correspondant.

En l'absence de communication bidirectionelle connue entre le client et le serveur sur le canal distant, le serveur doit attendre pendant au moins "lease_time" plus "blh_maximum_io_time" avant de transférer les dispositions du client original à d'autres clients. Le serveur, comme le client, doit avoir une approche prudente, et lancer le temporisateur d'expiration de prêt à partir du moment où il a reçu l'opération qui a renouvelé le prêt pour la dernière fois.

2.4 Questions de récupération de pannes

Une exigence critique en cas de récupération de panne est que le client et le serveur sachent tous deux quand l'autre est défaillant. De plus, il est exigé qu'un client ait une vue cohérente des données à travers les redémarrages du serveur. Ces exigences et une discussion complète des questions de récupération de panne sont couvertes dans la section "Récupération de panne" de la spécification NFSv4.1 [RFC5661]. Le présent document contient du matériel supplémentaire sur la récupération de panne spécifique seulement de la disposition de bloc/volume.

Quand le serveur tombe en panne alors que le client détient une disposition inscriptible, et que le client a écrit des données sur des blocs couverts par la disposition, et si les blocs sont encore dans l'état PNFS_BLOCK_INVALID_DATA, le client a deux options pour la récupération. Si les données écrites sur ces blocs sont encore en mémoire tampon chez le client, il peut simplement réécrire les données via NFSv4, une fois que le serveur est revenu en ligne. Cependant, si les données ne sont plus dans la mémoire tampon du client, il NE DOIT PAS tenter de prendre les données à partir des données du serveur. À la place, il devrait tenter d'engager les blocs en question chez le serveur durant la période de grâce de récupération du serveur, en envoyant un LAYOUTCOMMIT avec le fanion "loca_reclaim" réglé à vrai. Ce processus est décrit en détails au paragraphe 18.42.4 de la [RFC5661].

2.5 Rappel de ressources : CB_RECALL_ANY

Le serveur peut décider qu'il ne peut pas détenir tout l'état pour les dispositions sans épuiser ses ressources. Dans ce cas, il est libre de rappeler les dispositions individuelles en utilisant CB_LAYOUTRECALL pour réduire la charge, ou il peut choisir de demander que le client retourne toutes les dispositions.

La spécification de NFSv4.1 [RFC5661] définit les types suivants :

```
const RCA4_TYPE_MASK_BLK_LAYOUT = 4;

struct CB_RECALL_ANY4args {
     uint32_t      craa_objects_to_keep;
     bitmap4       craa_type_mask;
};
```

Quand le serveur envoie une demande CB_RECALL_ANY à un client en spécifiant le bit RCA4_TYPE_MASK_BLK_LAYOUT dans le gabarit craa_type_mask, le client devrait répondre immédiatement par NFS4_OK, et ensuite retourner en asynchrone les dispositions de fichier complètes jusqu'à ce que le nombre de fichiers avec les dispositions en mémoire tampon chez le client soit inférieur à craa_object_to_keep.

2.6 Erreurs transitoires et permanentes

Le serveur peut répondre à LAYOUTGET avec divers états d'erreur. Ces erreurs peuvent porter des conditions transitoires ou des conditions plus permanentes qui ont peu de chances d'être bientôt résolues.

Les erreurs transitoires, NFS4ERR_RECALLCONFLICT et NFS4ERR_TRYLATER, sont utilisées pour indiquer que le serveur ne peut pas accorder immédiatement la disposition au client. Dans le premier cas, c'est parce que le serveur a récemment produit un CB_LAYOUTRECALL au client demandeur, tandis que dans le cas de NFS4ERR_TRYLATER, le

serveur ne peut pas accorder la demande peut-être à cause de conflits de partage avec d'autres clients. Dans l'un et l'autre cas, une approche raisonnable pour le client est d'attendre plusieurs millisecondes et de réessayer la demande. Le client DEVRAIT garder trace du nombre d'essais, et si l'affaire ne progresse pas, le client DEVRAIT envoyer la demande d'opération READ ou WRITE directement au serveur.

L'erreur NFS4ERR_LAYOUTUNAVAILABLE peut être retournée par le serveur si les dispositions ne sont pas supportées pour le fichier demandé ou son système de fichiers contenant. Le serveur peut aussi retourner ce code d'erreur si le serveur est engagé dans un processus de migration du fichier à partir d'une mémorisation secondaire, ou pour toute autre raison qui fait que le serveur est incapable de fournir la disposition. Suite à la réception d'un NFS4ERR_LAYOUTUNAVAILABLE, le client DEVRAIT envoyer les demandes READ et WRITE futures directement au serveur. On s'attend à ce qu'un client ne mette pas pour toujours l'état de disposition indisponible du fichier en mémoire tampon, en particulier si le fichier est clos, et donc finalement, le client PEUT réitérer l'opération LAYOUTGET.

3. Considérations sur la sécurité

Normalement, les dispositifs de disques et les protocoles SAN fournissent des mécanismes de contrôle d'accès (par exemple, la transposition et/ou le masquage de LUN) qui opèrent à la granularité des hôtes individuels. La fonction de ces mécanismes rend possible au serveur de "clôturer" les machines clientes individuelles à l'égard de certains disques physiques -- c'est-à-dire, d'empêcher des machines clientes individuelles de lire ou écrire sur certains disques physiques. Des méthodes de contrôle d'accès d'une granularité plus fine ne sont généralement pas disponibles. Pour cette raison, certaines responsabilités de sécurité sont déléguées aux clients pNFS pour les dispositions de bloc/volume. Les systèmes de mémorisation de bloc/volume contrôlent généralement l'accès à la granularité de volume et donc les clients pNFS ne doivent être de confiance que pour effectuer des accès permis par les extensions de disposition qu'ils détiennent actuellement (par exemple, et non l'accès à la mémorisation pour des fichiers sur lesquels une extension de disposition n'est pas détenue). En général, le serveur ne va pas être capable d'empêcher un client qui détient une disposition pour un fichier d'accéder à des parties du disque physique non couvertes par la disposition. De même, le serveur ne va pas être capable d'empêcher un client d'accéder à des blocs couverts par une disposition qu'il a déjà retourné. Ce niveau de protection fondé sur le bloc doit être fourni par le logiciel client.

Un autre méthode d'utilisation de protocole de bloc/volume est que les appareils de mémorisation exportent des adresses de bloc virtuelles, qui reflètent bien les fichiers auxquels appartiennent les blocs. Ces adresses de bloc virtuelles sont exportées aux clients pNFS via les dispositions. Cela permet à l'appareil de mémorisation de faire les vérifications d'accès appropriées, tout en transposant les adresses de bloc virtuelles en adresses de bloc physiques. Dans des environnements où les exigences de sécurité sont telles que la protection côté client contre l'accès à une mémorisation en dehors des extensions de disposition autorisées n'est pas suffisante, des dispositions de mémorisation de bloc/volume pNFS NE DEVRAIENT PAS être utilisées si l'appareil de mémorisation n'est pas capable de mettre en œuvre les vérifications d'accès appropriées, via l'utilisation d'adresses de bloc virtuelles ou d'autres moyens. À l'opposé, un environnement où la protection côté client peut suffire consiste en clients, serveur et systèmes de mémorisation co-situés dans un centre de données avec un SAN physiquement isolé sous le contrôle d'un seul administrateur de système ou d'un petit groupe d'administrateurs de système.

Cela a aussi des implications pour certaines fonctions NFSv4 en dehors de pNFS. Par exemple, si un fichier est couvert par un verrou obligatoire de lecture seule, le serveur peut s'assurer que seulement des dispositions lisibles pour le fichier sont accordées aux clients pNFS. Cependant, il appartient à chaque client pNFS de s'assurer que la disposition lisible est utilisée seulement pour des demandes de service de lecture, et non pour permettre d'écrire sur les parties existantes du fichier. De même, les appareils de mémorisation de bloc/volume sont incapables de valider les listes de contrôle d'accès NFS et les modes de fichier ouverts, de sorte que le client doit appliquer les politiques avant d'envoyer une demande READ ou WRITE à l'appareil de mémorisation. Comme les systèmes de mémorisation de bloc/volume sont généralement incapables d'appliquer une telle sécurité fondée sur le fichier, dans les environnements où les clients pNFS ne peuvent pas être de confiance pour appliquer de telles politiques, les dispositions pNFS de mémorisation de bloc/volume NE DEVRAIENT PAS être utilisées.

L'accès à la mémorisation de bloc/volume est logiquement à une couche inférieure de la pile d'entrée/sortie que NFSv4, et donc la sécurité de NFSv4 n'est pas directement applicable aux protocoles qui accèdent directement à une telle mémorisation. Selon le protocole, certains des mécanismes de sécurité fournis par NFSv4 (par exemple, chiffrement, intégrité cryptographique) peuvent n'être pas disponibles ou peuvent être fournis via des moyens différents. À la limite, pNFS avec mémorisation de bloc/volume peut être utilisé avec des protocoles d'accès à la mémorisation (par exemple, SCSI en parallèle) qui ne fournissent essentiellement aucune fonction de sécurité. À une autre extrêmité, pNFS peut être utilisé avec des protocoles de mémorisation comme iSCSI qui peuvent fournir une fonction de sécurité significative. Il est de la responsabilité de ceux qui administrent et déploient pNFS avec un protocole d'accès à une mémorisation de

bloc/volume de s'assurer qu'une protection appropriée est fournie à ce protocole (la sécurité physique est un moyen courant pour les protocoles qui ne sont pas fondés sur IP). Dans les environnements où les exigences de sécurité pour le protocole de mémorisation ne peuvent pas être satisfaites, les dispositions pNFS de mémorisation de bloc/volume NE DEVRAIENT PAS être utilisées.

Quand la sécurité est disponible pour un protocole de mémorisation, elle est généralement à une granularité différente et avec une notion différente de l'identité que dans NFSv4 (par exemple, NFSv4 contrôle l'accès d'utilisateur aux fichiers, iSCSI contrôle l'accès de l'initiateur aux volumes). La responsabilité d'appliquer les correspondances appropriées entre ces couches de sécurité incombe au client pNFS. Comme avec les questions du premier paragraphe de cette section, dans les environnements où les exigences de sécurité sont telles que la protection côté client contre l'accès à la mémorisation en dehors de la disposition n'est pas suffisante, les dispositions pNFS de mémorisation de bloc/volume NE DEVRAIENT PAS être utilisées.

4. Conclusions

Le présent document spécifie le type de disposition bloc/volume pour pNFS et les fonctions associées.

5. Considérations relatives à l'IANA

Il n'y a pas de considérations relatives à l'IANA dans ce document. Toutes les considération relatives à l'IANA sur pNFS sont traitées dans la [RFC5661].

6. Remerciements

Le présent document s'appuie largement sur la familiarité des auteurs avec la fonction de transposition et le protocole du système de fichier multi-chemins (MPFS, *Multi-Path File System*) de EMC (précédemment appelé HighRoad) [MPFS]. Le protocole utilisé par MPFS est appelé protocole de transposition de fichier (FMP, *File Mapping Protocol*) ; c'est un protocole d'ajout qui fonctionne en parallèle avec des protocoles de système de fichier comme NFSv3 pour fournir des fonctions semblables à celles de pNFS pour la mémorisation de bloc/volume. Tout en s'appuyant sur FMP, les structures de données et les considérations fonctionnelles dans ce document diffèrent de façon significative, sur la base des leçons tirées et de l'opportunité de tirer parti des caractéristiques de NFSv4 comme les opérations COMPOUND. La conception de la prise en charge de la participation du client pNFS à la copie dans l'écriture est fondée sur le texte et les idées apportées par Craig Everhart.

Andy Adamson, Ben Campbell, Richard Chandler, Benny Halevy, Fredric Isaman, et Mario Wurzl ont tous aidé à la relecture des versions de cette spécification.

7. Références

7.1 Références normatives

[LEGAL] IETF Trust, "Legal Provisions Relating to IETF Documents", http://trustee.ietf.org/docs/IETF-Trust-License-Policy.pdf, novembre 2008.

[RFC2119] S. Bradner, "Mots clés à utiliser dans les RFC pour indiquer les niveaux d'exigence", BCP 14, mars 1997. DOI 10.17487/RFC2119, *(MàJ par* RFC8174)

[RFC4506] M. Eisler, éd., "XDR : norme de représentation des données externes", mai 2006. (STD0067)

[RFC5661] S. Shepler, M. Eisler, D. Noveck, "Système de fichiers réseau (NFS) version 4.1 : protocole", janvier 2010. *(P.S. ; MàJ par* RFC8178, RFC8434 *; remplacée par* RFC8881)

7.2 Références pour information

[MPFS] EMC Corporation, "EMC Celerra Multi-Path File System (MPFS)", EMC Data Sheet, http://www.emc.com/collateral/software/data-sheet/h2006-celerra-mpfs-mpfsi.pdf.

[SMIS] SNIA, "Storage Management Initiative Specification (SMI-S) v1.4", http://www.snia.org/tech_activities/standards/curr_standards/smi/SMI-S_Technical_Position_v1.4.0r4.zip.

Adresse des auteurs

David L. Black
EMC Corporation
176 South Street
Hopkinton, MA 01748
USA
téléphone : +1 (508) 293-7953
mél : black_david@emc.com

Stephen Fridella
Nasuni Inc
313 Speen St
Natick MA 01760

USA
mél : stevef@nasuni.com

Jason Glasgow
Google
5 Cambridge Center
Cambridge, MA 02142
USA
téléphone : +1 (617) 575 1599
mél : jglasgow@aya.yale.edu