Équipe d'ingénierie de l'Internet (IETF)
Request for Comments: 5664
Catégorie : Sur la voie de la normalisation
ISSN: 2070-1721
Traduction Claude Brière de L'Isle

B. Halevy,Panasas
B. Welch, Panasas
J. Zelenka, Panasas
janvier 2010

Opérations du système de fichier réseau parallèle (pNFS) fondées sur l'objet

Résumé
Le système de fichiers réseau en parallèle (pNFS, *Parallel NFS*) étend la version 4 du système de fichiers réseau (NFSv4, *Network File System 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 augmenter les performances et le parallélisme, mais exige des fonctions supplémentaires du client pour l'accès aux données, dont certaines dépendent de la classe de mémorisation utilisée, autrement dit le type de disposition. Les principales opérations de pNFS et les types de données dans NFSv4.1 spécifient une couche indépendante du type de disposition ; les informations spécifiques du type de disposition sont convoyées en utilisant des structures de données opaques dont la structure interne est définie par la spécification du type de disposition particulier. Le présent document spécifie le type de disposition pNFS fondé sur l'objet NFSv4.1 comme accompagnement de la spécification principale de NFSv4.1.

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/rfc5664

Table des matières

1. Introduction

Dans pNFS, le serveur de fichiers retourne des structures de disposition typées qui décrivent où est situé un fichier de données. Il y a des dispositions différentes pour les différents systèmes de mémorisation et les méthodes pour arranger les données sur les appareils de mémorisation. Le présent document décrit les dispositions utilisées avec les appareils de mémorisation fondés sur l'objet (OSD, *Object-based Storage Device*) qui sont accédés conformément aux normes d'OSD de protocole de mémorisation (ANSI INCITS 400-2004) [OSD].

Un "objet" est un conteneur pour des données et attributs, et les fichiers sont mémorisés dans un ou plusieurs objets. Le protocole OSD spécifie plusieurs opérations sur les objets, incluant READ, WRITE, FLUSH, GET ATTRIBUTES, SET ATTRIBUTES, CREATE, et DELETE. Cependant, en utilisant la disposition fondée sur l'objet le client utilise seulement les commandes READ, WRITE, GET ATTRIBUTES, et FLUSH. Les autres commandes sont seulement utilisées par le serveur pNFS. Une disposition fondée sur l'objet pour pNFS inclut des identifiants d'objet, des capacités qui permettent aux clients de lire ou écrire ces objets, et divers paramètres qui contrôlent comment des données de fichier sont réparties à travers leurs objets composants. Le protocole OSD a un schéma de sécurité fondé sur la capacité qui permet au serveur pNFS de contrôler quelles opérations et quels objets peuvent être utilisés par les clients. Ce schéma est décrit plus en détails dans la section des "Considérations sur la sécurité" (Section 13).

1.1 Langage des exigences

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].

2. Description en XDR du protocole de disposition fondée sur l'objet

Le présent document contient la description de représentation de données externes (XDR, *External Data Representation*) [RFC4506] du protocole de disposition d'objets 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 mettre ce document dans le script coquille suivant pour produire la description XDR lisible par la machine du protocole de disposition d'objets NFSv4.1 :

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

C'est-à-dire que 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 > pnfs_osd_prot.x
```

L'effet du script est de supprimer les espaces en tête de chaque ligne, plus une séquence sentinelle de "///".

L'en-tête de fichier XDR incorporé suit. Les descriptions XDR suivantes, avec la séquence sentinelle sont incorporées dans tout le document.

Noter que le code XDR contenu dans le présent document dépend des types provenant du fichier NFSv4.1 nfs4_prot.x [RFC5662]. 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.

2.1 Notice de licence des composants de code

La description XDR, marquée par des lignes qui commencent par la séquence "///", ainsi que les scripts pour extraire la description XDR sont des composants de code comme décrit à la Section 4 des "Dispositions légales relatives aux documents de l'IETF" [LEGAL]. Ces composants de code sont soumis à licence conformément aux termes de la Section 4 des "Dispositions légales relatives aux documents de l'IETF".

La redistribution et l'utilisation en forme source et binaire, avec ou sans modification, sont permises pourvu que les conditions suivantes soient satisfaites :

- Les redistributions de code source doivent conserver la notice de droits de reproduction ci-dessus, cette liste de conditions et le déclinatoire de responsabilité qui suit.
- Les redistributions en forme binaire doivent reproduire la notice de droits de reproduction ci-dessus, cette liste de conditions et le déclinatoire de responsabilité qui suit dans la documentation et/ou autres matériaux fournis avec la distribution.
- Ni le nom de la Internet Society, de l'IETF ou de l'IETF Trust, ni les noms des contributeurs spécifiques, ne peuvent être utilisés pour avaliser ou promouvoir des produits dérivés de ce logiciel sans permission écrite spécifique préalable.

Ce logiciel est fourni par les détenteurs du droit de reproduction et les contributeurs "TEL QUEL" et toutes garanties expresses ou implicites, incluant, sans s'y limiter, les garanties implicites de commercialisabilité et de convenance à un objet particulier sont déclinées. En aucun cas les détenteurs du droit de reproduction et les contributeurs ne seront responsables de dommages directs, indirects, incidents, spéciaux, exemplaires, ou consécutifs (incluant, sans s'y limiter, la fourniture de biens ou services de substitution, perte d'usage, données, ou profits, ou interruption d'affaires) cependant causés, et aucune théorie de responsabilité, contractuelle, stricte, ou pour tort (incluant la négligence ou autrement) survenant d'une façon quelconque de l'utilisation de ce logiciel, même si on est avisé de la possibilité de tels dommages.
Ce code a été déduit de la RFC 5664. Prière de reproduire cette note si possible.

```
/// /*
///  * pnfs_osd_prot.x
///  */
///
/// %#include <nfs4_prot.x>
///
```

3. Définitions de types de données de base

Les paragraphes qui suivent définissent les types de données de base et les constantes utilisés par le protocole de disposition fondée sur l'objet.

3.1 pnfs_osd_objid4

Un objet est identifié par un numéro, un peu comme un numéro inode. Le modèle de mémorisation d'objet a un schéma à deux niveaux, où les objets dans un appareil de mémorisation d'objet sont groupés en partitions.

```
/// struct pnfs_osd_objid4 {
///     deviceid4       oid_device_id;
///     uint64_t        oid_partition_id;
///     uint64_t        oid_object_id;
/// };
///
```

Le type pnfs_osd_objid4 est utilisé pour identifier un objet au sein d'une partition sur un appareil de mémorisation d'objet spécifié. "oid_device_id" choisit l'appareil de mémorisation d'objet dans l'ensemble d'appareils de mémorisation disponibles. L'appareil est identifié par le type deviceid4, qui est un index dans les informations d'adressage sur cet appareil retournées par les opérations GETDEVICELIST et GETDEVICEINFO. Le type de données deviceid4 est défini dans NFSv4.1 [RFC5661]. Dans un OSD, une partition est identifiée avec un nombre de 64 bits, "oid_partition_id". Dans une partition, un objet est identifié par un nombre de 64 bits, "oid_object_id". La création et la gestion des partitions sort du domaine d'application du présent document, et est une facilité fournie par le système de fichiers de mémorisation fondée sur l'objet.

3.2 pnfs_osd_version4

```
/// enum pnfs_osd_version4 {
///     PNFS_OSD_MISSING    = 0,
///     PNFS_OSD_VERSION_1  = 1,
///     PNFS_OSD_VERSION_2  = 2
/// };
///
```

pnfs_osd_version4 est utilisé pour indiquer la version du protocole OSD ou si un objet est manquant (c'est-à-dire, indisponible). Certains algorithmes de RAID qui prennent en charge la disposition fondée sur l'objet codent des informations redondantes et peuvent compenser des composants manquants, mais l'algorithme de placement de données a besoin de savoir quelles parties sont manquantes.

Pour l'instant, la norme OSD en est à la version 1.0, et on prévoit une version 2.0 de la norme (SNIA T10/1729-D OSD-2]). La seconde génération du protocole OSD propose des caractéristiques supplémentaires pour prendre en charge des capacités plus robustes de récupération d'erreur, de photographies, et de gammes d'octets. Donc, la version OSD est explicitement invoquée dans les informations retournées dans la disposition. (Ces informations peuvent aussi être déduites en regardant le type de capacités dans le champ format, qui est le premier octet. La valeur de format est 0x1 pour une capacité OSD v1. Cependant, il semble plus robuste d'invoquer explicitement la version.)

3.3 pnfs_osd_object_cred4

```
/// enum pnfs_osd_cap_key_sec4 {
///     PNFS_OSD_CAP_KEY_SEC_NONE = 0,
///     PNFS_OSD_CAP_KEY_SEC_SSV  = 1
/// };
///
/// struct pnfs_osd_object_cred4 {
///     pnfs_osd_objid4             oc_object_id;
///     pnfs_osd_version4           oc_osd_version;
///     pnfs_osd_cap_key_sec4  oc_cap_key_sec;
```

```
///    opaque                  oc_capability_key<>;
///    opaque                  oc_capability<>;
/// };
///
```

La structure pnfs_osd_object_cred4 est utilisée pour identifier chaque composant constituant le fichier. Le "oc_object_id" identifie l'objet composant, le "oc_osd_version" représente la version du protocole osd, ou si ce composant est indisponible, et les "oc_capability" et "oc_capability_key", avec le "oda_systemid" provenant de pnfs_osd_deviceaddr4, fournissent les accréditifs de sécurité OSD nécessaires pour accéder à cet objet. La valeur "oc_cap_key_sec" note la méthode utilisée pour sécuriser la oc_capability_key (voir les détails à la Section 13.1).

Pour se conformer aux exigences de sécurité de OSD, la clé de capacité DEVRAIT être transférée de façon sûre pour empêcher l'espionnage (voir la Section 13). Donc, un client DEVRAIT soit produire les opérations LAYOUTGET ou GETDEVICEINFO via RPCSEC_GSS avec le service de confidentialité, soit établir préalablement un vérificateur d'état de secret (SSV, *secret state verifier*) pour les sessions via l'opération NFSv4.1 SET_SSV. Le type pnfs_osd_cap_key_sec4 est utilisé pour identifier la méthode du serveur pour sécuriser la clé de capacité.

- PNFS_OSD_CAP_KEY_SEC_NONE note que la oc_capability_key n'est pas chiffrée, et dans ce cas le client DEVRAIT produire les opérations LAYOUTGET ou GETDEVICEINFO avec RPCSEC_GSS et le service de confidentialité ou le transport NFSv4.1 devrait être sécurisé en utilisant des méthodes externes à NFSv4.1 comme l'utilisation de IPsec [RFC4301] pour transporter le protocole NFSV4.1.

- PNFS_OSD_CAP_KEY_SEC_SSV note que les contenus oc_capability_key sont chiffrés en utilisant le contexte SSV GSS et la clé de capacité comme entrées à la fonction GSS_Wrap() (voir GSS-API [RFC2743]) avec le fanion conf_req_flag réglé à VRAI. Le client DOIT utiliser la clé secrète SSV au titre du contexte GSS du client pour déchiffrer la clé de capacité en utilisant la valeur du champ oc_capability_key comme message d'entrée à la fonction GSS_unwrap(). Noter que pour empêcher l'espionnage de la clé SSV, le client DEVRAIT produire SET_SSV via RPCSEC_GSS avec le service de confidentialité.

La méthode choisie réelle dépend de si le client a établi une clé SSV avec le serveur et si il a produit l'opération avec la méthode de confidentialité RPCSEC_GSS. Naturellement, si le client n'a pas établi une clé SSV via SET_SSV, le serveur DOIT utiliser la méthode PNFS_OSD_CAP_KEY_SEC_NONE. Autrement, si l'opération n'a pas été produite avec la méthode de confidentialité RPCSEC_GSS, le serveur DEVRAIT sécuriser la oc_capability_key avec la méthode PNFS_OSD_CAP_KEY_SEC_SSV. Le serveur PEUT aussi utiliser la méthode PNFS_OSD_CAP_KEY_SEC_SSV quand l'opération a été produite avec la méthode de confidentialité RPCSEC_GSS.

3.4 pnfs_osd_raid_algorithm4

```
/// enum pnfs_osd_raid_algorithm4 {
///    PNFS_OSD_RAID_0    = 1,
///    PNFS_OSD_RAID_4    = 2,
///    PNFS_OSD_RAID_5    = 3,
///    PNFS_OSD_RAID_PQ   = 4    /* Reed-Solomon P+Q */
/// };
///
```

pnfs_osd_raid_algorithm4 représente l'algorithme de redondance des données utilisé pour protéger le contenu du fichier. Voir les détails au paragraphe 5.4.

4. Adressage et découverte d'appareil de mémorisation d'objet

Les opérations de données sur un OSD exigent que le client connaisse "l'adresse" de chaque objet racine de l'OSD. L'objet racine est synonyme de l'unité logique de l'interface système de petit ordinateur (SCSI, *Small Computer System Interface*). Le client spécifie les unités logiques de SCSI à sa pile de protocole SCSI en utilisant une représentation locale pour le client. Parce que ces représentations sont locales, GETDEVICEINFO doit retourner les information qui peuvent être utilisées par le client pour choisir la représentation locale correcte.

Dans le monde du bloc, un décalage d'ensemble (numéro de bloc logique ou piste/secteur) contient une étiquette de disque.

Cette étiquette identifie le disque de façon univoque. À l'opposé, un OSD a un ensemble standard d'attributs sur son objet racine. Pour les besoins d'identification de l'appareil, l'identifiant de système OSD (numéro d'attribut d'informations racines 3) et le nom d'OSD (numéro d'attribut d'informations racines 9) sont utilisés comme étiquette. Cela apparaît dans le type pnfs_osd_deviceaddr4 ci-dessous dans les champs "oda_systemid" et "oda_osdname".

Dans certaines situations, la découverte de la cible SCSI peut devoir être fondée sur des informations contenues dans la réponse GETDEVICEINFO. Un exemple en est les cibles de SCSI Internet (iSCSI) qui ne sont pas connues du client tant qu'une disposition n'a pas été demandée. Les informations fournies dans les champs "oda_targetid", "oda_targetaddr", et "oda_lun" dans le type pnfs_osd_deviceaddr4 décrit ci-dessous (paragraphe 4.2) permettent au client de sonder un appareil spécifique connaissant son adresse réseau et facultativement son nom iSCSI (voir iSCSI [RFC3720]) ou quand l'adresse réseau de l'appareil est omise, lui permettent de découvrir l'appareil de mémorisation d'objet en utilisant le nom d'appareil ou l'identifiant d'appareil SCSI fourni (voir [SPC-3].)

Le oda_systemid est implicitement utilisé par le client, en utilisant la clé de signature d'accréditif d'objet pour signer chaque demande avec la valeur de vérification d'intégrité de la demande. Cette méthode protège le client d'un accès non intentionnel à un appareil si la transposition de l'adresse d'appareil a changé (ou été révoquée). Le serveur calcule la clé de capacité en utilisant sa propre vue de l'identifiant de système associé aux identifiant d'appareil respectifs présents dans l'accréditif. Si la vue du client de la transposition d'identifiant d'appareil est périmée, le client va utiliser le mauvais systemid (qui doit être unique à l'échelle du système) et la demande d'entrée/sortie à l'OSD va échouer à la vérification d'intégrité.

Pour récupérer de cette condition, le client devrait rapporter l'erreur et retourner la disposition en utilisant LAYOUTRETURN, et invalider toutes les transpositions d'adresse d'appareil associées à cette disposition. Le client peut alors si il le souhaite demander une nouvelle disposition en utilisant LAYOUTGET et résoudre les identifiants d'appareil référencés en utilisant GETDEVICEINFO ou GETDEVICELIST.

Le serveur DOIT fournir le oda_systemid et DEVRAIT aussi fournir le oda_osdname. Quand le nom d'OSD est présent, le client DEVRAIT obtenir les attributs d'informations racines chaque fois qu'il établit une communication avec l'OSD et vérifier que le nom d'OSD qu'il a obtenu de l'OSD correspond à celui envoyé par le serveur de métadonnées. Pour ce faire, le client utilise les accréditifs root_obj_cred.

4.1 pnfs_osd_targetid_type4

L'énumération suivante spécifie la manière dont une cible SCSI peut être spécifiée. La cible peut être spécifiée comme un nom SCSI, ou comme un identifiant SCSI.

```
/// enum pnfs_osd_targetid_type4 {
///     OBJ_TARGET_ANON          = 1,
///     OBJ_TARGET_SCSI_NAME     = 2,
///     OBJ_TARGET_SCSI_DEVICE_ID   = 3
/// };
///
```

4.2 pnfs_osd_deviceaddr4

La spécification pour une adresse d'appareil d'objet est comme suit :

```
/// union pnfs_osd_targetid4 switch (pnfs_osd_targetid_type4 oti_type) {
///     case OBJ_TARGET_SCSI_NAME:
///         string          oti_scsi_name<>;
///
///     case OBJ_TARGET_SCSI_DEVICE_ID:
///         opaque          oti_scsi_device_id<>;
///
///     default:
///         void;
/// };
///
/// union pnfs_osd_targetaddr4 switch (bool ota_disponible) {
```

```
///     case TRUE:
///         netaddr4          ota_netaddr;
///     case FALSE:
///         void;
/// };
///
/// struct pnfs_osd_deviceaddr4 {
///     pnfs_osd_targetid4       oda_targetid;
///     pnfs_osd_targetaddr4     oda_targetaddr;
///     opaque                   oda_lun[8];
///     opaque                   oda_systemid<>;
///     pnfs_osd_object_cred4    oda_root_obj_cred;
///     opaque                   oda_osdname<>;
/// };
///
```

4.2.1 Identifiant de cible SCSI

Quand "oda_targetid" est spécifié comme un OBJ_TARGET_SCSI_NAME, la chaîne "oti_scsi_name" DOIT être formatée comme un "nom iSCSI" comme spécifié dans iSCSI [RFC3720] et [RFC3980]. Noter que la spécification du format oti_scsi_name sort du domaine d'application du présent document. L'analyse de la chaîne est fondée sur le préfixe de chaîne, par exemple, "iqn.", "eui.", ou "naa." et plus de formats POURRONT être spécifiés à l'avenir en accord avec les propriétés des noms iSCSI.

Actuellement, le nom iSCSI assure la dénomination de l'appareil cible en utilisant une chaîne formatée comme un nom qualifié iSCSI (IQN, *iSCSI Qualified Name*) ou un identifiant unique étendu (EUI, *Extended Unique Identifier*) [EUI-64]. Ceux-ci sont normalement utilisés pour identifier des appareils iSCSI ou du protocole d'acheminement sécurisé (SRP, *Secure Routing Protocol*) [SRP]. Le format de chaîne d'autorité d'adresse réseau (NAA, *Network Address Authority*) (voir la [RFC3980]) assure la dénomination de l'appareil en utilisant des identifiants uniques au monde, comme défini dans la spécification de tramage et signalisation de canal fibre (FC-FS, *Fibre Channel Framing and Signaling*) [FC-FS-2]. Ils sont normalement utilisés pour identifier des appareils de canal fibre ou de SCSI rattachés en série (SAS, *Serial Attached SCSI*) [SAS-1.1]. En particulier, de tels appareils sont à double rattachement sur canal fibre ou SAS et sur iSCSI.

Quand "oda_targetid" est spécifié comme un OBJ_TARGET_SCSI_DEVICE_ID, le champ opaque "oti_scsi_device_id" DOIT être formaté comme identifiant d'appareil SCSI comme défini dans la page 83h VPD de [SPC-3] (paragraphe 7.6.3, "Page VDP d'identification d'appareil"). Si l'identifiant d'appareil est identique à l'identifiant de système OSD, comme donné par oda_systemid, le serveur DEVRAIT fournir valeur opaque oti_scsi_device_id de longueur zéro. Noter que comme pour le "oti_scsi_name", la spécification du contenu opaque oti_scsi_device_id sort du domaine d'application du présent document et que plus de formats POURRONT être spécifiés à l'avenir en accord avec SPC-3.

Le pnfs_osd_targetid_type4 OBJ_TARGET_ANON PEUT être utilisé pour ne pas fournir d'identification de cible. Dans ce cas, seuls l'identifiant de système OSD, et facultativement l'adresse réseau fournie, sont utilisés pour localiser l'appareil.

4.2.2 Adresse réseau d'appareil

Le champ facultatif "oda_targetaddr" PEUT être fourni par le serveur comme indication pour accélérer la découverte d'appareil sur, par exemple, le protocole de transport iSCSI. L'adresse réseau est donnée avec le type netaddr4, qui spécifie un point d'extrémité fondé sur TCP/IP (comme spécifié dans NFSv4.1 [RFC5661]). Quand elle est donnée, le client DEVRAIT l'utiliser pour sonder si l'appareil SCSI est à l'adresse réseau donnée. Le client PEUT encore utiliser d'autres mécanismes de découverte comme le service de nom de mémorisation sur Internet (iSNS, *Internet Storage Name Service*) [RFC4171] pour localiser l'appareil en utilisant oda_targetid. En particulier, un tel service de noms externe DEVRAIT être utilisé quand les appareils peuvent être rattachés au réseau en utilisant plusieurs connexions, et/ou plusieurs tissus de mémorisation (par exemple, canal fibre et iSCSI).

Le champ "oda_lun" identifie le numéro d'unité logique OSD de 64 bits, formaté conformément à [SAM-3]. Le client utilise le numéro d'unité logique pour communiquer avec l'unité logique OSD spécifique. Son utilisation est définie en détails par le protocole de transport SCSI, par exemple, iSCSI [RFC3720].

5. Disposition fondée sur l'objet

Le type layout4 est défini dans NFSv4.1 [RFC5661] comme suit :

```
enum layouttype4 {
     LAYOUT4_NFSV4_1_FILES     = 1,
     LAYOUT4_OSD2_OBJECTS      = 2,
     LAYOUT4_BLOCK_VOLUME  = 3,
 };

struct layout_content4 {
     layouttype4         loc_type;
     opaque              loc_body<>;
 };

struct layout4 {
     offset4             lo_offset;
     length4             lo_length;
     layoutiomode4       lo_iomode;
     layout_content4     lo_content;
 };
```

Le présent document définit la structure associée à la valeur de layouttype4, LAYOUT4_OSD2_OBJECTS. NFSv4.1 [RFC5661] spécifie la structure loc_body comme un type XDR "opaque". La disposition opaque n'est pas interprétée par les couches de client génériques pNFS, mais évidemment doit être interprétée par le pilote de disposition de mémorisation d'objet. Cette section définit la structure de cette valeur opaque, pnfs_osd_layout4.

5.1 pnfs_osd_data_map4

```
/// struct pnfs_osd_data_map4 {
///     uint32_t                odm_num_comps;
///     length4                 odm_stripe_unit;
///     uint32_t                odm_group_width;
///     uint32_t                odm_group_depth;
///     uint32_t                odm_mirror_cnt;
///     pnfs_osd_raid_algorithm4    odm_raid_algorithm;
/// };
 ///
```

La structure pnfs_osd_data_map4 donne les paramètres de l'algorithme qui transpose le contenu d'un fichier sur les objets composants. Au lieu de limiter le système à un simple schéma de surface où la perte d'un seul objet composant résulte en la perte de données, les paramètres de transposition prennent en charge le reflet et des schémas plus compliqués qui protègent contre la perte d'un objet composant.

"odm_num_comps" est le nombre d'objets composants sur lequel le fichier est étalé. Le serveur PEUT augmenter le fichier en ajoutant plus de composants à la surface pendant que les clients conservent des dispositions valides jusqu'à ce que le fichier ait atteint sa largeur finale. La longueur du fichier dans ce cas DOIT être limitée au nombre d'octets dans une surface complète.

La "odm_stripe_unit" est le nombre d'octets placés dans un composant avant d'avancer au prochain sur la liste des composants. Le nombre d'octets dans une surface complète est odm_stripe_unit fois le nombre de composants. Dans certains schémas RAID, une surface inclut des informations redondantes (c'est-à-dire, de parité) qui permettent au système de récupérer de pertes ou dommages à l'objet composant.

Les paramètres "odm_group_width" et "odm_group_depth" permettent un schéma de surface incorporé (voir les détails au paragraphe 5.3.2). Si il n'y a pas d'incorporation, alors odm_group_width et odm_group_depth DOIVENT être à zéro. La taille du dispositif de composants DOIT être un multiple de odm_group_width.

"odm_mirror_cnt" est utilisé pour dupliquer un fichier en dupliquant ses objets composants. Si il n'y a pas de reflet, alors

odm_mirror_cnt DOIT être 0. Si odm_mirror_cnt est supérieur à zéro, alors la taille du dispositif composant DOIT être un multiple de (odm_mirror_cnt+1) ; voir les détails au paragraphe 5.3.

5.2 pnfs_osd_layout4

```
/// struct pnfs_osd_layout4 {
///     pnfs_osd_data_map4      olo_map;
///     uint32_t                olo_comps_index;
///     pnfs_osd_object_cred4   olo_components<>;
/// };
///
```

La structure pnfs_osd_layout4 spécifie une disposition sur un ensemble d'objets composants. Le champ "olo_components" est un dispositif d'identifiants d'objet et d'accréditifs de sécurité qui accorde l'accès à chaque objet. L'organisation des données est définie par le type pnfs_osd_data_map4 qui spécifie comment les données du fichier sont transposées en les objets composants (c'est-à-dire, le schéma de surfaçage). L'algorithme de placement des données qui transpose les données de fichier en objets composants suppose que chaque objet composant se produit exactement une fois dans le dispositif de composants. Donc, les objets composants DOIVENT apparaître seulement une fois dans le dispositif olo_components. Le dispositif de composants peut représenter tous les objets composant le fichier, dans ce cas, "olo_comps_index" est réglé à zéro et le nombre d'entrées dans le dispositif olo_ccomponents est égal à olo_map.odm_num_comps. Le serveur PEUT retourner moins de composants que odm_num_comps, pourvu que les componsats retournés soient suffisants pour accéder à tout octet dans la gamme de données de la disposition (par exemple, une sous-surface des composants de "odm_group_width"). Dans ce cas, olo_comps_index représente la position du dispositif de composants retourné au sein du dispositif complet de composants que comporte le fichier.

Noter que la disposition dépend de la taille du fichier, que le client apprend des paramètres génériques de retour de LAYOUTGET, en faisant des commandes GETATTR au serveur de métadonnées. Le client utilise la taille de fichier pour décider si il devrait boucher les trous avec des zéros ou retourner une lecture abrégée. Des schémas de surfaçage peuvent causer des cas où les objets composants sont plus courts que d'autres composants parce que un trou se trouve correspondre à la dernière partie de l'objet composant.

5.3 Schémas de transposition de données

Ce paragraphe décrit en détail les différents schémas de transposition de données. La disposition d'objet utilise toujours une disposition "dense" comme décrit dans NFSv4.1 [RFC5661]. Cela signifie que la seconde unité de surface du fichier commence au décalage 0 du second composant, plutôt qu'à un décalage de stripe_unit octets. Après qu'une surface complète a été écrite, l'unité de surface suivante est ajoutée au premier objet composant dans la liste sans aucun trou dans les objets composants.

5.3.1 Surface simple

La transposition du décalage logique au sein d'un fichier (L) en l'objet composant C et le décalage spécifique de l'objet O sont définis par les équations suivantes :

L = décalage logique dans le fichier
W = nombre total de composants
S = W * stripe_unit *(unité de surfaçage)*
N = L / S
C = (L-(N*S)) / stripe_unit
O = (N*stripe_unit)+(L%stripe_unit)

Dans ces équations, S est le nombre d'octets dans une surface pleine, et N est le nombre de surfaces. C est un indice dans la matrice des composants, donc il choisit un appareil particulier de mémorisation d'objet. N et C comptent tous deux à partir de zéro. O est le décalage au sein de l'objet qui correspond au décalage du fichier. Noter que ces calculs ne traitent pas le même objet apparaissant plusieurs fois dans la matrice olo_components.

Par exemple, considérons un objet réparti sur quatre appareils, <D0 D1 D2 D3>. La stripe_unit est 4096 octets. La largeur de surface S est donc 4 * 4096 = 16384.

Décalage 0 :
N = 0 / 16 384 = 0
C = 0-0 / 4 096 = 0 (D0)
O = 0 * 4 096 + (0 % 4 096) = 0

Décalage 4 096 :
N = 4 096 / 16384 = 0
C = (4 096-(0 * 16 384)) / 4 096 = 1 (D1)
O = (0 * 4 096)+(4 096 % 4 096) = 0

Décalage 9 000 :
N = 9 000 / 16 384 = 0
C = (9 000-(0 * 16 384)) / 4 096 = 2 (D2)
O = (0 * 4 096)+(9 000 % 4 096) = 808

Décalage 132 000 :
N = 132 000 / 16 384 = 8
C = (132 000 - (8 * 16 384)) / 4 096 = 0 (D0)
O = (8 * 4 096) + (132 000 % 4 096) = 33 696

5.3.2 Surface incorporée

Les paramètres odm_group_width et odm_group_depth permettent un schéma de surfaçage incorporé. odm_group_width définit la largeur d'une surface de données et odm_group_depth définit combien de surfaces sont écrites avant d'avancer au prochain groupe de composants dans la liste des objets composants pour le fichier. Le calcul utilisé pour transposer du décalage d'un fichier à un objet composant et son décalage au sein de cet objet est montré ci-dessous. Les calculs transposent du décalage logique L à l'indice de composant C et au décalage relatif O au sein de cet objet composant.

L = décalage logique dans le fichier
W = nombre total de composants
S = stripe_unit * group_depth * W
T = stripe_unit * group_depth * group_width
U = stripe_unit * group_width
M = L / S
G = (L - (M * S)) / T
H = (L - (M * S)) % T
N = H / U
C = (H - (N * U)) / stripe_unit + G * group_width
O = L % stripe_unit + N * stripe_unit + M * group_depth * stripe_unit

Dans ces équations, S est le nombre d'octets sur tous les objets composants avant que le schéma se répète. T est le nombre d'octets sur un groupe d'objets composants avant d'avancer au prochain groupe. U est le nombre d'octets dans une surface au sein d'un groupe. M est le nombre de surfaces "majeures" (c'est-à-dire, sur tous les composants) et N est le nombre de surfaces "mineures" (c'est-à-dire, sur le groupe). G compte les groupes depuis le début de la surface majeure, et H est le décalage d'octets au sein du groupe.

Par exemple, considérons un objet étalé sur 100 appareils avec une group_width *(largeur de groupe)* de 10, une group_depth *(profondeur de groupe)* de 50, et une stripe_unit *(unité de surface)* de 1 MB. Dans ce schéma, 500 MB sont écrits sur les dix premiers composants, et 5000 MB sont écrits avant que le schéma revienne au premier composant dans la matrice.

Décalage 0 :
W = 100
S = 1 MB * 50 * 100 = 5000 MB
T = 1 MB * 50 * 10 = 500 MB
U = 1 MB * 10 = 10 MB
M = 0 / 5000 MB = 0
G = (0 - (0 * 5000 MB)) / 500 MB = 0
H = (0 - (0 * 5000 MB)) % 500 MB = 0
N = 0 / 10 MB = 0

C = (0 - (0 * 10 MB)) / 1 MB + 0 * 10 = 0
O = 0 % 1 MB + 0 * 1 MB + 0 * 50 * 1 MB = 0

Décalage 27 MB :
M = 27 MB / 5000 MB = 0
G = (27 MB - (0 * 5000 MB)) / 500 MB = 0
H = (27 MB - (0 * 5000 MB)) % 500 MB = 27 MB
N = 27 MB / 10 MB = 2
C = (27 MB - (2 * 10 MB)) / 1 MB + 0 * 10 = 7
O = 27 MB % 1 MB + 2 * 1 MB + 0 * 50 * 1 MB = 2 MB

Décalage 7232 MB :
M = 7232 MB / 5000 MB = 1
G = (7232 MB - (1 * 5000 MB)) / 500 MB = 4
H = (7232 MB - (1 * 5000 MB)) % 500 MB = 232 MB
N = 232 MB / 10 MB = 23
C = (232 MB - (23 * 10 MB)) / 1 MB + 4 * 10 = 42
O = 7232 MB % 1 MB + 23 * 1 MB + 1 * 50 * 1 MB = 73 MB

5.3.3 Miroir

Le compte odm_mirror_cnt est utilisé pour dupliquer un fichier en dupliquant ses objets composants. Si il n'y a pas de miroir, odm_mirror_cnt DOIT alors être 0. Si odm_mirror_cnt est supérieur à zéro, la taille de la matrice olo_components DOIT être un multiple de (odm_mirror_cnt+1). Donc, pour un miroir classique sur deux objets, odm_mirror_cnt est un. Noter que le reflet peut être défini sur tout algorithme RAID et schéma de surfaçage (simple ou incorporé). Si odm_group_width est aussi différent de zéro, alors la taille de la matrice olo_components DOIT être un multiple de odm_group_width * (odm_mirror_cnt+1). Les répliques sont adjacentes dans la matrice olo_components, et la valeur C produite par les équations ci-dessus n'est pas un indice direct dans la matrice olo_components. Les équations suivantes déterminent plutôt l'indice de composant de réplique RCi, où i va de 0 à odm_mirror_cnt.

C = indice de composant de surfaçage ou surfaçage à deux niveaux
i va de 0 à odm_mirror_cnt, inclus
RCi = C * (odm_mirror_cnt+1) + i

5.4 Algorithmes RAID

pnfs_osd_raid_algorithm4 détermine l'algorithme et le placement des données redondantes. Ce paragraphe définit les différents algorithmes de redondance. Note : le terme "RAID" (Redundant Array of Independent Disks, matrice redondante à disques indépendants) est utilisé dans le présent document pour représenter une matrice d'objets composants qui mémorisent des données pour un fichier individuel. Les objets sont mémorisés sur des appareils de mémorisation fondée sur l'objet indépendants. Les données du fichier sont codées et étalées sur la matrice d'objets composants en utilisant les algorithmes développés pour les systèmes RAID fondés sur le bloc.

5.4.1 PNFS_OSD_RAID_0

PNFS_OSD_RAID_0 signifie qu'il n'y a pas de données de parité, de sorte que tous les octets dans les objets composants sont des octets de données localisés par les équations ci-dessus pour C et O. Si un objet composant est marqué comme PNFS_OSD_MISSING, le client pNFS DOIT retourner une erreur d'entrée/sortie si il y a une tentative de lecture de ce composant ou, autrement, il peut réessayer le READ sur le serveur pNFS.

5.4.2 PNFS_OSD_RAID_4

PNFS_OSD_RAID_4 sigifie que le dernier objet composant, ou le dernier dans chaque groupe (si odm_group_width est supérieur à zero) contient des informations de parité calculées sur le reste de la surface avec une opération OUX. Si un objet composant est indisponible, le client peut lire le reste des unités de surface dans la surface endommagée et recalculer l'unité de surface manquante en OUXant les autres unités de surface dans la surface. Ou le client peut répéter le READ sur le serveur pNFS qui va probablement effectuer la lecture reconstruite au nom du client.

Quand la parité est présente dans le fichier, il y a alors un calcul supplémentaire pour transposer du décalage de fichier L au décalage qui compte pour la parité incorporée, L'. On calcule d'abord L', et on utilise ensuite L' dans les équations ci-dessous pour C et O.

L = décalage de fichier, non compris la parité
P = nombre d'appareils de parité dans chaque surface
W = group_width, si non zéro, autrement, taille de la matrice olo_components
N = L / (W-P * stripe_unit)
L' = N * (W * stripe_unit) + (L % (W-P * stripe_unit))

5.4.3 PNFS_OSD_RAID_5

PNFS_OSD_RAID_5 signifie que la position des données de parité subit une rotation sur chaque surface ou chaque groupe (si odm_group_width est supérieur à zero). Dans la première surface, le dernier composant contient la parité. Dans la seconde surface, l'avant dernier composant contient la parité, et ainsi de suite. Dans ce schéma, toutes les unités de surface subissent une rotation de sorte que l'entrée/sortie est également répartie entre les objets lorsque le fichier est lu à la suite. La disposition de parité rotative est illustrée ci-dessous, les numéros indiquant l'unité de surface.

```
0 1 2 P
4 5 P 3
8 P 6 7
P 9 a b
```

Pour calculer l'objet composant C, on calcule d'abord le décalage qui compte pour la parité L' et on utilise cela pour calculer C. On fait ensuite tourner C pour obtenir C'. Finalement, on augmente C' de un si les informations de parité viennent à ou avant C' au sein de cette surface. Les équations suivantes illustrent cela en calculant I, qui est l'indice du composant qui contient la parité pour une surface donnée.

L = décalage de fichier, non compris la parité
W = odm_group_width, si non zéro, autrement taille de la matrice olo_components
N = L / (W-1 * stripe_unit)
(Calcule L' comme décrit ci-dessus)
(Calcule C fondé sur L' comme décrit ci-dessus)
C' = (C - (N%W)) % W
I = W - (N%W) - 1
si (C' <= I) {
C'++
}

5.4.4 PNFS_OSD_RAID_PQ

PNFS_OSD_RAID_PQ est un schéma de double parité qui utilise le schéma de codage Reed-Solomon P+Q [TECC]. Dans cette disposition, les deux derniers objets composants contiennent, respectivement, les données P et Q. P est la parité calculée avec OUX, et Q est une équation plus complexe qui n'est pas décrite ici. Les équations données ci-dessus pour la parité incorporée peuvent être utilisées pour transposer un décalage de fichier en l'objet composant correct en réglant le nombre de composants de parité à 2 au lieu de 1 pour RAID4 ou RAID5. Les clients peuvent simplement choisir de lire les données à travers le serveur de métadonnées si deux composants sont manquants ou endommagés.

5.4.5 Usage de RAID et notes de mise en œuvre

Les dispositions de RAID avec des données redondantes dans leurs surfaces exigent une mise en série supplémentaire des mises à jour pour assurer un fonctionnement correct. Autrement, si deux clients écrivent simultanément sur la même gamme logique d'un objet, le résultat pourrait inclure des données différentes dans les même gammes de couples reflétés, ou corrompre les informations de parité. Il est de la responsabilité du serveur de métadonnées d'appliquer des exigences de mise en série comme celle là. Par exemple, le serveur de métadonnées peut le faire en ne permettant pas de dispositions d'écriture en chevauchement dans les objets reflétés.

6. Mise à jour de disposition fondée sur l'objet

layoutupdate4 est utilisé dans l'opération LAYOUTCOMMIT pour porter les mises à jour à la disposition et des informations supplémentaires au serveur de métadonnées. Il est défini dans NFSv4.1 [RFC5661] comme suit :

```
struct layoutupdate4 {
    layouttype4         lou_type;
    opaque              lou_body<>;
};
```

Le type layoutupdate4 est une valeur opaque au niveau générique de client pNFS. Si le type de disposition lou_type est LAYOUT4_OSD2_OBJECTS, alors la valeur opaque lou_body est définie par le type pnfs_osd_layoutupdate4.

Les clients de pNFS fondé sur l'objet ne sont pas autorisés à modifier la disposition. Donc, les informations passées dans pnfs_osd_layoutupdate4 sont utilisées seulement pour mettre à jour les attributs du fichier. En plus des informations génériques que le client peut passer au serveur de métadonnées dans LAYOUTCOMMIT comme le plus fort décalage auquel le client a écrit et la dernière fois où il a modifié le fichier, le client PEUT utiliser pnfs_osd_layoutupdate4 pour porter la capacité consommée (ou livrée) par les écritures en utilisant la disposition, et pour indiquer que des erreurs d'entrée/sortie ont été rencontrées dans de telles écritures.

6.1. pnfs_osd_deltaspaceused4

```
/// union pnfs_osd_deltaspaceused4 switch (bool dsu_valid) {
///     case TRUE:
///         int64_t     dsu_delta;
///     case FALSE:
///         void;
/// };
///
```

pnfs_osd_deltaspaceused4 est utilisé pour porter des informations d'utilisation de l'espace au moment de LAYOUTCOMMIT. Pour que le système de fichiers maintienne correctement les informations de capacité utilisées, il a besoin de garder trace de la capacité consommée par les opérations WRITE effectuées par le client. Dans ce protocole, le OSD retourne la capacité consommé par un write (*), qui peut être différente du nombre d'octets écrits à cause de frais généraux internes comme une allocation au niveau du bloc et de blocs indirects, et que le client reflète en retour au serveur pNFS afin qu'il puisse suivre précisément les quotas. Le serveur pNFS peut choisir de faire confiance à ces informations venant des clients et donc éviter d'interroger les OSD au moment du LAYOUTCOMMIT. Si le client est incapable d'obtenir cette information de l'OSD, il retourne simplement un olu_delta_space_used invalide.

6.2 pnfs_osd_layoutupdate4

```
/// struct pnfs_osd_layoutupdate4 {
///     pnfs_osd_deltaspaceused4    olu_delta_space_used;
///     bool                        olu_ioerr_flag;
/// };
///
```

"olu_delta_space_used" est utilisé pour porter en retour les informations d'usage de capacité au serveur de métadonnées.

Le fanion "olu_ioerr_flag" est utilisé quand des erreurs d'entrée/sortie sont rencontrées lors de l'écriture du fichier. Le client DOIT rapporter les erreurs en utilisant la structure pnfs_osd_ioerr4 (paragraphe 8.1) à l'instant LAYOUTRETURN.

Si le client réussit à mettre à jour le fichier avant de toucher les erreurs d'entrée/sortie, il PEUT utiliser LAYOUTCOMMIT pour mettre à jour le serveur de métadonnées comme décrit ci-dessus. Normalement, dans le cas sans erreurs, le serveur PEUT continuer et mettre à jour les attributs du fichier sur les appareils de mémorisation. Cependant, si des erreurs d'entrée/sortie ont été rencontrées, le serveur ferait mieux de ne pas tenter d'écrire les nouveaux attributs sur les appareils de mémorisation avant d'avoir reçu le rapport d'erreurs d'entrée/sortie ; donc, le client DOIT régler le fanion olu_ioerr_flag à "vrai". Noter que dans ce cas, le client DEVRAIT envoyer les deux opérations LAYOUTCOMMIT et LAYOUTRETURN dans le même COMPOUND RPC.

7. Récupération d'erreurs d'entrée/sortie de client

Le client pNFS peut rencontrer des erreurs quand il accède directement aux appareils de mémorisation d'objet. Cependant, il est de la responsabilité du serveur de métadonnées de traiter les erreurs d'entrée/sortie. Quand le type de disposition LAYOUT4_OSD2_OBJECTS est utilisé, le client DOIT rapporter les erreurs d'entrée/sortie au serveur à l'instant LAYOUTRETURN en utilisant la structure pnfs_osd_ioerr4 (voir le paragraphe 8.1).

Le serveur de métadonnées analyse l'erreur et détermine les opérations de récupération exigées comme de réparer les incohérences de parité, les défaillances de support de récupération, ou de reconstruire les objets manquants.

Le serveur de métadonnées DEVRAIT se rappeler de toutes les dispositions en instance pour lui permettre un accès en écriture exclusif aux surfaces à récupérer et pour empêcher que d'autres clients subissent la même condition d'erreur. Dans ce cas, le serveur DOIT achever la récupération avant de traiter d'autres dispositions sur les gammes d'octets affectées.

Bien qu'il PUISSE être acceptable que le client propage une erreur correspondante à l'application qui a initié l'opération d'entrée/sortie et élimine toutes les données non écrites, le client DEVRAIT tenter de réessayer l'opération originale d'entrée/sortie en demandant une nouvelle disposition utilisant LAYOUTGET et de réessayer la ou les opérations d'entrée/sortie en utilisant la nouvelle disposition, ou le client PEUT juste réessayer la ou les opérations d'entrée/sortie en utilisant des opérations régulières NFS READ ou WRITE via le serveur de métadonnées. Le client DEVRAIT tenter de restituer une nouvelle disposition et réessayer l'opération d'entrée/sortie en utilisant d'abord des commandes d'OSD et seulement si l'erreur persiste, réessayer l'opération d'entrée/sortie via le serveur de métadonnées.

8. Retour de disposition fondée sur l'objet

layoutreturn_file4 est utilisé dans l'opération LAYOUTRETURN pour porter des informations spécifiques du type de disposition au serveur. Il est défini dans NFSv4.1 [RFC5661] comme suit :

```
struct layoutreturn_file4 {
        offset4         lrf_offset;
        length4         lrf_length;
        stateid4        lrf_stateid;                    /* données spécifiques de layouttype4 */
        opaque          lrf_body<>;
 };

union layoutreturn4 switch(layoutreturn_type4 lr_returntype) {
        case LAYOUTRETURN4_FILE:
             layoutreturn_file4      lr_layout;
        default:
             void;
 };

struct LAYOUTRETURN4args {                              /* fichier CURRENT_FH: */
        bool                    lora_reclaim;
        layoutreturn_stateid    lora_recallstateid;
        layouttype4             lora_layout_type;
        layoutiomode4           lora_iomode;
        layoutreturn4           lora_layoutreturn;
 };
```

Si le type de disposition lora_layout_type est LAYOUT4_OSD2_OBJECTS, alors la valeur opaque lrf_body est définie par le type pnfs_osd_layoutreturn4.

Le type pnfs_osd_layoutreturn4 permet au client de rapporter les informations d'erreur d'entrée/sortie au serveur de métadonnées comme défini ci-dessous.

8.1 pnfs_osd_errno4

```
/// enum pnfs_osd_errno4 {
///     PNFS_OSD_ERR_EIO                = 1,
///     PNFS_OSD_ERR_NOT_FOUND          = 2,
///     PNFS_OSD_ERR_NO_SPACE           = 3,
///     PNFS_OSD_ERR_BAD_CRED           = 4,
///     PNFS_OSD_ERR_NO_ACCESS          = 5,
///     PNFS_OSD_ERR_UNREACHABLE = 6,
///     PNFS_OSD_ERR_RESOURCE           = 7
/// };
///
```

pnfs_osd_errno4 est utilisé pour représenter les types d'erreur quand des erreurs de lecture/écriture sont rapportées au serveur de métadonnées. Les codes d'erreur servent d'indications au serveur de métadonnées qui peuvent l'aider à diagnostiquer la raison exacte de l'erreur et à la réparer.

- PNFS_OSD_ERR_EIO indique que l'opération a échoué parce que l'appareil de mémorisation d'objet a rencontré une défaillance en essayant d'accéder à l'objet. La source la plus courante de ces erreurs est une erreur du support, mais d'autres erreurs internes pourraient aussi causer cela. Dans ce cas, le serveur de métadonnées devrait aller examiner l'objet endommagé de plus près ; donc, il devrait être utilisé comme code d'erreur par défaut.

- PNFS_OSD_ERR_NOT_FOUND indique que l'identifiant d'objet spécifie un objet qui n'existe pas dans l'appareil de mémorisation d'objet.

- PNFS_OSD_ERR_NO_SPACE indique que l'opération a échoué parce que l'appareil de mémorisation d'objet n'a plus de capacité libre durant l'opération.

- PNFS_OSD_ERR_BAD_CRED indique que les paramètres de sécurité ne sont pas valides. La principale cause en est que la capacité a expiré, ou que l'étiquette de politique d'accès (autrement dit, le numéro de version de capacité) a été changé pour révoquer les capacités. Le client va devoir retourner la disposition et en obtenir une nouvelle avec des capacités fraîches.

- PNFS_OSD_ERR_NO_ACCESS indique que la capacité ne permet pas l'opération demandée. Cela ne devrait pas se produire en fonctionnement normal parce que le serveur de métadonnées devrait donner les capacités correctes, ou aucune.

- PNFS_OSD_ERR_UNREACHABLE indique que le client n'a pas achevé l'opération d'entrée/sortie à l'appareil de mémorisation d'objet du fait d'un échec de la communication. Si l'opération d'entrée/sortie a été exécutée ou non par l'OSD est indéterminé.

- PNFS_OSD_ERR_RESOURCE indique que le client n'a pas produit l'opération d'entrée/sortie à cause d'un problème local chez l'initiateur (c'est-à-dire, côté client) par exemple, quand il n'a plus de mémoire disponible. Le client DOIT garantir que la commande d'OSD n'a jamais été envoyée à l'OSD.

8.2 pnfs_osd_ioerr4

```
/// struct pnfs_osd_ioerr4 {
///     pnfs_osd_objid4     oer_component;
///     length4             oer_comp_offset;
///     length4             oer_comp_length;
///     bool                oer_iswrite;
///     pnfs_osd_errno4     oer_errno;
/// };
///
```

La structure pnfs_osd_ioerr4 est utilisée pour retourner des indications d'erreur pour des objets qui ont généré des erreurs durant les transferts de données. Ce sont des indications au serveur de métadonnées qu'il y a des problèmes avec cet objet. Pour chaque erreur, "oer_component", "oer_comp_offset", et "oer_comp_length" représentent l'objet et la gamme d'octets au sein de l'objet composant dans lequel l'erreur s'est produite ; "oer_iswrite" est réglé à "vrai" si l'opération OSD qui a échoué modifiait des données, et "oer_errno" représente le type d'erreur.

Les gammes d'octet composantes dans la structure facultative pnfs_osd_ioerr4 sont utilisées pour récupérer l'objet et DOIVENT être réglées par le client à couvrir toutes les opérations d'entrée/sortie défaillantes au composant.

8.3 pnfs_osd_layoutreturn4

```
/// struct pnfs_osd_layoutreturn4 {
///     pnfs_osd_ioerr4           olr_ioerr_report<>;
/// };
///
```

Quand des opérations d'entrée/sortie OSD sont défaillantes, "olr_ioerr_report<>" est utilisé pour rapporter ces erreurs au serveur de métadonnées comme un dispositifs d'éléments de type pnfs_osd_ioerr4. Chaque élément dans le dispositif représente une erreur qui s'est produite sur l'objet spécifié par oer_component. Si aucune erreur n'est à rapporter, la taille du dispositif olr_ioerr_report<> est réglée à zéro.

9. Indication de disposition de création fondée sur l'objet

Le type layouthint4 est défini dans NFSv4.1 [RFC5661] comme suit :

```
struct layouthint4 {
   layouttype4          loh_type;
   opaque               loh_body<>;
};
```

La structure layouthint4 est utilisée par le client pour passer une indication sur le type de disposition qu'il aimerait créer pour un fichier particulier. Si le type de disposition loh_type est LAYOUT4_OSD2_OBJECTS, alors la valeur opaque loh_body est définie par le type pnfs_osd_layouthint4.

9.1 pnfs_osd_layouthint4

```
/// union pnfs_osd_max_comps_hint4 switch (bool omx_valid) {
///     case TRUE:
///         uint32_t          omx_max_comps;
///     case FALSE:
///         void;
/// };
///
/// union pnfs_osd_stripe_unit_hint4 switch (bool osu_valid) {
///     case TRUE:
///         length4           osu_stripe_unit;
///     case FALSE:
///         void;
/// };
///
/// union pnfs_osd_group_width_hint4 switch (bool ogw_valid) {
///     case TRUE:
///         uint32_t          ogw_group_width;
///     case FALSE:
///         void;
/// };
///
/// union pnfs_osd_group_depth_hint4 switch (bool ogd_valid) {
///     case TRUE:
///         uint32_t          ogd_group_depth;
///     case FALSE:
///         void;
/// };
```

```
///
/// union pnfs_osd_mirror_cnt_hint4 switch (bool omc_valid) {
///     case TRUE:
///         uint32_t           omc_mirror_cnt;
///     case FALSE:
///         void;
/// };
///
/// union pnfs_osd_raid_algorithm_hint4 switch (bool ora_valid) {
///     case TRUE:
///         pnfs_osd_raid_algorithm4    ora_raid_algorithm;
///     case FALSE:
///         void;
/// };
///
/// struct pnfs_osd_layouthint4 {
///     pnfs_osd_max_comps_hint4        olh_max_comps_hint;
///     pnfs_osd_stripe_unit_hint4      olh_stripe_unit_hint;
///     pnfs_osd_group_width_hint4      olh_group_width_hint;
///     pnfs_osd_group_depth_hint4      olh_group_depth_hint;
///     pnfs_osd_mirror_cnt_hint4       olh_mirror_cnt_hint;
///     pnfs_osd_raid_algorithm_hint4   olh_raid_algorithm_hint;
/// };
///
```

Ce type porte des indications pour la transposition de données désirée. Tous les paramètres sont facultatifs de sorte que le client peut donner des valeurs pour les seuls paramètres dont il se soucie, par exemple il peut fournir une indication pour le nombre désiré de composants reflétés, sans considération de l'algorithme RAID choisi pour le fichier. Le serveur devrait tenter de respecter les indications, mais il peut en ignorer certaines ou toutes à son entière discrétion et sans faire échouer l'opération CREATE concernée.

"olh_max_comps_hint" peut être utilisé pour limiter le nombre total d'objets composants du fichier. Toutes les autres indications correspondent directement aux différents champs de pnfs_osd_data_map4.

10. Segments de disposition

Les opérations de disposition pnfs opèrent sur des gammes d'octets logiques. Il n'y a pas d'exigence dans le protocole pour des relations entre les gammes d'octets utilisées dans LAYOUTGET pour acquérir des dispositions et des gammes d'octets utilisées dans CB_LAYOUTRECALL, LAYOUTCOMMIT, ou LAYOUTRETURN. Cependant, l'utilisation de capacités de gamme d'octet OSD met des limitations sur ces opérations car les capacités associées aux segments de disposition ne peuvent pas être fusionnées ou partagées. Les lignes directrices suivantes devraient être respectées pour un fonctionnement approprié des dispositions fondées sur l'objet.

10.1 CB_LAYOUTRECALL et LAYOUTRETURN

En général, le pilote de disposition fondée sur l'objet devrait garder trace de chaque segment de disposition qu'il a obtenu, garder un enregistrement du mode d'entrée/sortie, du décalage, et de la longueur du segment. Le serveur devrait permettre au client d'obtenir plusieurs segments de disposition se chevauchant mais est libre de rappeler la disposition pour empêcher le chevauchement.

En réponse à CB_LAYOUTRECALL, le client devrait retourner tous les segments de disposition qui correspondent au mode d'entrée/sortie donné et se chevauchent avec la gamme rappelée. Quand il retourne les dispositions pour cette gamme d'octets avec LAYOUTRETURN, le client NE DOIT PAS retourner une sous gamme d'un segment de disposition qu'il a ; chaque LAYOUTRETURN envoyé DOIT couvrir complètement au moins un segment de disposition en instance.

Le serveur, à son tour, devrait livrer tout segment qui correspond exactement à l'identifiant de client, au mode d'entrée/sortie, et à la gamme d'octets donnés dans LAYOUTRETURN. Si une correspondance exacte n'est pas trouvée, le serveur devrait alors livrer tous les segments de disposition qui correspondent à l'identifiant de client et au mode

d'entrée/sortie et qui sont entièrement contenus dans la gamme d'octets retournée. Si aucun n'est trouvé et si la gamme d'octets est un sous ensemble d'un segment de disposition en instance avec le même identifiant de client et mode d'entrée/sortie, le client peut alors être considéré comme dysfonctionnant et le serveur DEVRAIT rappeler toutes les dispositions provenant de ce client pour réinitialiser son état. Si ce comportement se répète, le serveur DEVRAIT refuser tous les LAYOUTGET provenant de ce client.

10.2 LAYOUTCOMMIT

LAYOUTCOMMIT est seulement utilisé par pNFS fondé sur l'objet pour porter des indications d'attributs modifiés et/ou pour rapporter la présence d'erreurs d'entrée/sortie au serveur de métadonnées. Donc, le décalage et la longueur dans LAYOUTCOMMIT4args sont réservés pour une utilisation future et devraient être réglés à 0.

11. Rappels de dispositions

Le serveur de métadonnées fondé sur l'objet devrait rappeler les dispositions en instance dans les cas suivants :

- Quand la politique de sécurité du fichier change, c'est-à-dire, quand les bits de liste de contrôle d'accès (ACL, *Access Control List*) ou de mode de permission sont établis.
- Quand la carte d'agrégation de fichier change, rendant invalides les dispositions en instance.
- Quand elles partagent des conflits. Par exemple, le serveur va produire des segments de disposition alignés sur la surface pour les objets RAID-5. Pour empêcher la corruption de la parité du fichier, plusieurs clients ne doivent pas détenir des dispositions d'écriture valides pour les mêmes surfaces. Une disposition READ/WRITE (RW) en instance devrait être rappelée quand un LAYOUTGET en conflit est reçu d'un client différent pour LAYOUTIOMODE4_RW et pour une gamme d'octets en chevauchement avec le segment de disposition en instance.

11.1 CB_RECALL_ANY

Le serveur de métadonnées peut utiliser l'opération de rappel CB_RECALL_ANY pour notifier au client de retourner certaines ou toutes ses dispositions. NFSv4.1 [RFC5661] définit les types suivants :

```
const RCA4_TYPE_MASK_OBJ_LAYOUT_MIN    = 8 ;
const RCA4_TYPE_MASK_OBJ_LAYOUT_MAX   = 9 ;

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

Normalement, CB_RECALL_ANY va être utilisé pour rappeler l'état du client quand le serveur a besoin de réclamer des ressources. Le gabarit binaire craa_type_mask spécifie le type de ressources qui sont rappelées et la valeur craa_objects_to_keep spécifie combien des objets rappelés le client est autorisé à garder. Les fanions de gabarit de type de disposition fondée sur l'objet sont définis comme suit. Ils représentent le mode d'entrée/sortie des dispositions rappelées. En réponse, le client DEVRAIT retourner les dispositions du mode d'entrée/sortie rappelées dont il a le moins besoin, gardant au plus craa_objects_to_keep dispositions fondées sur l'objet.

```
/// enum pnfs_osd_cb_recall_any_mask {
///     PNFS_OSD_RCA4_TYPE_MASK_READ = 8,
///     PNFS_OSD_RCA4_TYPE_MASK_RW   = 9
/// };
///
```

Le fanion PNFS_OSD_RCA4_TYPE_MASK_READ notifie au client de retourner les dispositions de mode d'entrée/sortie LAYOUTIOMODE4_READ. De même, le fanion PNFS_OSD_RCA4_TYPE_MASK_RW notifie au client de retourner les dispositions de mode d'entrée/sortie LAYOUTIOMODE4_RW. Quand les deux fanions de gabarit sont établis, il est demandé au client de retourner les dispositions de l'un et l'autre mode d'entrée/sortie.

12. Clôture de client

Dans les cas où les clients sont pas en communication et où leur prêt a expiré, ou quand les clients ne retournent pas les dispositions rappelées au moins dans une période de prêt (voir "Rappel d'une disposition"[RFC5661]) le serveur PEUT révoquer les dispositions du client et/ou les transpositions d'adresse d'appareil et réallouer ces ressources à d'autres clients. Pour éviter la corruption des données, le serveur de métadonnées DOIT ériger une barrière pour empêcher les clients d'accéder aux objets respectifs, comme décrit au paragraphe 13.4.

13. Considérations sur la sécurité

L'extension pNFS partage le protocole de système de fichiers NFSv4 en deux parties, le chemin de contrôle et le chemin des données (protocole de mémorisation). Le chemin de contrôle contient toutes les nouvelles opérations décrites par cette extension ; tous les mécanismes et caractéristiques de sécurité existants de NFSv4 s'appliquent au chemin de contrôle. La combinaison des composants dans un système pNFS est exigée pour préserver les propriétés de sécurité de NFSv4 par rapport à une entité accédant aux données via un client, incluant des contre-mesures de sécurité pour se défendre contre les menaces pour lesquelles NFSv4 fournit des défenses dans les environnements où ces menaces sont considérées comme significatives.

Le serveur de métadonnées applique la politique de contrôle d'accès du fichier à l'instant LAYOUTGET. Le client devrait utiliser des accréditifs d'autorisation convenables pour obtenir la disposition pour le mode d'entrée/sortie demandé (READ ou RW) et le serveur vérifie les permissions et l'ACL pour ces accréditifs, éventuellement en retournant NFS4ERR_ACCESS si il n'est pas permis au client d'avoir le mode d'entrée/sortie demandé. Si l'opération LAYOUTGET réussit, le client reçoit, au titre de la disposition, un ensemble de capacités d'objet lui permettant l'accès en entrée/sortie aux objets spécifiés correspondant au mode d'entrée/sortie demandé. Quand le client agit sur des opérations d'entrée/sortie au nom de ses utilisateurs locaux, il DOIT authentifier et autoriser l'utilisateur en produisant les invocations OPEN et ACCESS respectives au serveur de métadonnées, similaire à avoir des délégations de données NFSv4. Si l'accès est permis, le client utilise les capacités (READ ou RW) correspondantes pour effectuer les opérations d'entrée/sortie aux appareils de mémorisation d'objet. Quand le serveur de métadonnées reçoit une demande pour changer les permissions ou ACL d'un fichier, il DEVRAIT rappeler toutes les dispositions pour ce fichier et il DOIT changer l'attribut de version de capacité sur tous les objets composant le fichier pour invalider implicitement toutes les capacités en instance avant d'engager les nouvelles permissions et ACL. Faire cela assure que les clients ré-autorisent leurs dispositions conformément aux permissions et ACL modifiées en demandant de nouvelles dispositions. Rappeler les dispositions dans ce cas est une faveur du serveur destinée à empêcher les clients d'avoir une erreur sur les entrées/sorties faite après le changement de version de capacité.

Le protocole de mémorisation d'objet DOIT mettre en œuvre les aspects de sécurité décrits dans la version 1 de la définition de protocole OSD T10 [OSD]. La norme définit quatre méthodes de sécurité : NOSEC, CAPKEY, CMDRSP, et ALLDATA. Pour fournir un niveau minimum de sécurité permettant la vérification et l'application de la politique de contrôle d'accès du serveur en utilisant les accréditifs de sécurité de disposition, la méthode de sécurité NOSEC NE DOIT PAS être utilisée pour une opération d'entrée/sortie. Le reste de cette section fait un survol des mécanismes de sécurité décrits dans cette norme. L'objectif est de donner au lecteur une compréhension de base du modèle de sécurité d'objet. Toute discordance entre ce texte et la norme réelle est évidemment à résoudre en faveur de la norme OSD.

13.1 Types de données de sécurité d'OSD

Trois principaux types de données sont associés à la sécurité d'objet : une capacité, un accréditif, et des paramètres de sécurité. La capacité est un ensemble de champs qui spécifient un objet et les opérations qui peuvent être effectuées sur lui. Un accréditif est une capacité signée. Seul un gestionnaire de sécurité qui connaît les clés secrètes d'appareil peut correctement signer une capacité pour former un accréditif valide. Dans pNFS, le serveur de fichiers agit comme gestionnaire de sécurité et retourne des capacités signées (c'est-à-dire, des accréditifs) au client pNFS. Les paramètres de sécurité sont des valeurs calculées par le producteur des commandes OSD (c'est-à-dire, le client) qui prouve qu'il détient des accréditifs valides. Le client utilise l'accréditif comme clé de signature pour signer les demandes qu'il fait à OSD, et met les signatures résultantes dans le champ security_parameters de la commande OSD. L'appareil de mémorisation d'objet utilise les clés secrètes qu'il partage avec le gestionnaire de sécurité pour valider les valeurs de signature dans les paramètres de sécurité.

Les types de sécurité sont opaques pour les couches génériques du client pNFS. Le contenu des accréditifs est défini comme opaque dans le type pnfs_osd_object_cred4. Au lieu de répéter ici les définitions, le lecteur est invité à se reporter au paragraphe 4.9.2.2 de la norme OSD.

13.2 Protocole de sécurité d'OSD

Le protocole de mémorisation d'objet s'appuie sur une capacité cryptographiquement sûre pour contrôler les accès aux appareils de mémorisation d'objet. Les capacités sont générées par le serveur de métadonnées, retournées au client, et utilisées par le client comme décrit ci-dessous pour authentifier leurs demandes à l'appareil de mémorisation fondée sur l'objet. Les capacités réalisent donc la vérification exigée de mode d'accès et d'ouverture. Elles permettent au serveur de fichiers de définir et vérifier une politique (par exemple, mode ouvert) et l'OSD pour appliquer cette politique sans en connaître les détails (par exemple, les identifiants et ACL d'utilisateurs).

Comme les capacités sont liées aux dispositions, et comme elles sont utilisées pour appliquer le contrôle d'accès, quand le fichier ACL ou le mode change, les capacités en instance DOIVENT être révoquées pour appliquer les nouvelles permissions d'accès. Le serveur DEVRAIT rappeler les dispositions pour permettre aux clients de retourner en douceur leurs capacités avant que les permissions d'accès changent.

Chaque capacité est spécifique d'un objet particulier, d'une opération sur cet objet, d'une gamme d'octets au sein de l'objet (dans OSDv2) et a un instant d'expiration explicite. Les capacités sont signées avec une clé secrète qui est partagée par les appareils de mémorisation d'objet et les gestionnaires des métadonnées. Les clients n'ont pas de clés d'appareil et sont donc incapables de falsifier les signatures dans les paramètres de sécurité. La combinaison d'une capacité, de l'identifiant de système OSD, et d'une signature est appelée un "accréditif" dans la spécification OSD.

Les détails du modèle de sécurité et de confidentialité pour la mémorisation d'objet sont définis dans la norme OSD T10. Le schéma d'algorithme suivant devrait aider le lecteur à comprendre le modèle de base.

LAYOUTGET retourne un CapKey et un Cap, qui, avec l'identifiant de système OSD, sont aussi appelés un accréditif. C'est une capacité et une signature sur cette capacité et l'identifiant de système. La norme OSD se réfère à la CapKey comme "valeur de vérification d'intégrité d'accréditif" et au ReqMAC comme la "valeur de vérification d'intégrité de la demande".

CapKey = MAC<SecretKey>(Cap, SystemID)
Credential = {Cap, SystemID, CapKey}

Le client utilise CapKey pour signer toutes les demandes qu'il produit pour cet objet en utilisant le Cap respectif. En d'autres termes, le Cap apparaît dans la demande à l'appareil de mémorisation, et cette demande est signée avec le CapKey comme suit :

ReqMAC = MAC<CapKey>(Req, ReqNonce)
Request = {Cap, Req, ReqNonce, ReqMAC}

Ce qui suit est envoyé à l'OSD : {Cap, Req, ReqNonce, ReqMAC}. L'OSD utilise la SecretKey qu'il partage avec le serveur de métadonnées pour comparer le ReqMAC envoyé par le client avec une valeur calculée localement :

LocalCapKey = MAC<SecretKey>(Cap, SystemID)
LocalReqMAC = MAC<LocalCapKey>(Req, ReqNonce)

et si elles correspondent, l'OSD suppose que les capacités sont venues d'un authentique serveur de métadonnées et il permet l'accès à l'objet, comme permis par le Cap.

13.3 Exigences de confidentialité du protocole

Noter que si la réponse du serveur LAYOUTGET, qui contient CapKey et Cap, est volée par un autre client, elle peut être utilisée pour générer des demandes OSD valides (sous réserve des restrictions d'accès de Cap).

Pour satisfaire les exigences de confidentialité requises pour la clé de capacité retournée par LAYOUTGET, le cadre GSS-API [RFC2743] peut être utilisé, par exemple, en utilisant la méthode de confidentialité RPCSEC_GSS pour envoyer l'opération LAYOUTGET ou en utilisant la clé SSV pour chiffrer la oc_capability_key en utilisant la fonction GSS_Wrap(). Deux façons générales pour fournir la confidentialité en l'absence de GSS-API qui sont indépendantes de NFSv4 sont un réseau isolé comme un VLAN ou un canal sûr fourni par IPsec [RFC4301].

13.4 Révocation de capacités

À tout moment, le serveur de métadonnées peut invalider toutes les capacités en instance sur un objet en changeant son attribut POLICY ACCESS TAG. La valeur de POLICY ACCESS TAG fait partie d'une capacité, et elle doit correspondre à l'état de l'attribut de l'objet. Si elles ne correspondent pas, l'OSD rejette les accès à l'objet avec la clé de sens réglée à ILLEGAL REQUEST et un code de sens supplémentaire réglé à INVALID FIELD IN CDB. Quand un client tente d'utiliser une capacité et est rejeté de cette façon, il devrait produire un LAYOUTCOMMIT pour l'objet et spécifier PNFS_OSD_BAD_CRED dans le paramètre olr_ioerr_report. Le client peut choisir de produire un LAYOUTRETURN/LAYOUTGET (ou LAYOUTCOMMIT/LAYOUTRETURN/LAYOUTGET) composé pour tenter d'aller chercher un ensemble de capacités rafraîchi.

Le serveur de métadonnées peut choisir de changer l'étiquette de politique d'accès sur un objet à tout moment, pour n'importe quelle raison (en comprenant qu'il y a probablement une pénalité de performance associée, en particulier si il y a des dispositions en instance pour cet objet). Le serveur de métadonnées DOIT révoquer les capacités en instance quand un des événements suivants se produit :

- o les permissions sur l'objet changent,
- o un verrou en conflit de gamme d'octets obligatoire est accordé, ou
- o une disposition est révoquée et réallouée à un autre client.

Un client pNFS va normalement détenir une disposition pour chaque gamme d'octets pour READ ou READ/WRITE. Les accréditifs de client sont vérifiés par le serveur de métadonnées à l'instant LAYOUTGET et il est de la responsabilité du client d'appliquer le contrôle d'accès parmi plusieurs utilisateurs accédant au même fichier. Il n'est ni exigé ni attendu que le client pNFS obtienne une disposition séparée pour chaque utilisateur accédant à un objet partagé. Le client DEVRAIT utiliser les invocations OPEN et ACCESS pour vérifier les permissions d'utilisateur quand il effectue les entrées/sorties afin que les politiques de contrôle d'accès du serveur soient correctement appliquées. Le résultat de l'opération ACCESS peut être mis en antémémoire pendant que le client détient une disposition valide car le serveur est supposé rappeler les dispositions quand les permissions d'accès ou l'ACL du fichier changent.

14. Considérations relatives à l'IANA

Comme décrit dans NFSv4.1 [RFC5661], de nouveaux numéros de type de disposition ont été alloués par l'IANA. Le présent document définit le protocole associé au numéro existant de type de disposition, LAYOUT4_OSD2_OBJECTS, et il ne demande aucune autre action de la part de l'IANA.

15. Références

15.1 Références normatives

[EUI-64] IEEE, "Guidelines for 64-bit Global Identifier (EUI-64) Registration Authority", <http://standards.ieee.org/regauth/oui/tutorials/EUI64.html>.

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

[OSD] Weber, R., "Information Technology - SCSI Object-Based Storage Device Commands (OSD)", ANSI INCITS 400-2004, décembre 2004.

[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)

[RFC2743] J. Linn, "Interface générique de programme d'application de service de sécurité, version 2, mise à jour 1", janvier 2000. (*MàJ par* RFC5554)

[RFC3720] J. Satran et autres, "Interface Internet des systèmes de petits ordinateurs (iSCSI)", avril 2004. *(Remplacée par* RFC7143)

[RFC3980] M. Krueger et autres, "Format de dénomination d'autorité d'adresse réseau T11 (NAA) pour les noms de nœuds iSCSI", février 2005. (*MàJ* RFC3720) *(P.S.) (Remplacée par* RFC7143)

[RFC4171] J. Tseng et autres, "Service de noms de mémorisation sur Internet (iSNS)", septembre 2005. *(P.S.)*

[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)

[RFC5662] S. Shepler, M. Eisler, D. Noveck, "Système de fichiers réseau (NFS) version 4.1 : Description de la norme de représentation des données externes (XDR)", janvier 2010. *(P.S.)*

[SAM-3] Weber, R., "SCSI Architecture Model - 3 (SAM-3)", ANSI INCITS 402-2005, février 2005.

[SPC-3] Weber, R., "SCSI Primary Commands - 3 (SPC-3)", ANSI INCITS 408-2005, octobre 2005.

15.2 Références pour information

[FC-FS-2] T11 1619-D, "Fibre Channel Framing and Signaling - 2 (FC-FS-2)", ANSI INCITS 424-2007, février 2007.

[OSD-2] Weber, R., "SCSI Object-Based Storage Device Commands -2 (OSD-2)", janvier 2009, <http://www.t10.org/cgi-bin/ac.pl?t=f&f=osd2r05a.pdf>.

[RFC4301] S. Kent et K. Seo, "Architecture de sécurité pour le protocole Internet", DOI 10.17487/RFC4301, décembre 2005. *(P.S.) (Remplace la* RFC2401)

[SAS-1.1] T10 1601-D, "Serial Attached SCSI - 1.1 (SAS-1.1)", ANSI INCITS 417-2006, juin 2006.

[SRP] T10 1415-D, "SCSI RDMA Protocol (SRP)", ANSI INCITS 365-2002, décembre 2002.

[TECC] MacWilliams, F. and N. Sloane, "The Theory of Error-Correcting Codes, Part I", 1977.

Appendice A. Remerciements

Todd Pisek était un co-éditeur des versions initiales de ce document. Daniel E. Messinger, Pete Wyckoff, Mike Eisler, Sean P. Turner, Brian E. Carpenter, Jari Arkko, David Black, et Jason Glasgow ont relu et commenté ce document.

Adresse des auteurs

Benny Halevy
Panasas, Inc.
1501 Reedsdale St. Suite 400
Pittsburgh, PA 15233
USA
mél : bhalevy@panasas.com

Brent Welch
Panasas, Inc.
6520 Kaiser Drive
Fremont, CA 95444
USA
mél : welch@panasas.com

Jim Zelenka
Panasas, Inc.
1501 Reedsdale St. Suite 400
Pittsburgh, PA 15233
USA
mél : jimz@panasas.com