Université Paris 8 de Vincennes à Saint-Denis
Département d'Informatique
Laboratoire d'Intelligence Artificielle
2, rue de la liberté
93526 Saint Denis CEDEX 02
Tél : 01 49 40 64 04

Analyse et réalisation d'un système de disques répartis à forte disponibilité

Marc Hufschmitt
mh@ai.univ-paris8.fr

sous la direction de Jean Méhat

Mémoire de DEA d'Intelligence Artificielle
et Optimisation Combinatoire

25 Septembre 2002

Résumé

Nous présentons une étude et une réalisation d'un système distribué de stockage réparti à forte disponibilité reposant sur l'algorithme distribué de consensus *Paxos*. L'accès au système de stockage réparti est effectué au travers de *disques virtuels*, disponibles sur des serveurs de disques. Un disque virtuel est caractérisé par un schéma de répartition des données. Ce schéma décrit l'organisation du stockage de données sur l'ensemble des serveurs. La répartition et la redondance des données dépendent uniquement de ce schéma et permettent, d'une part, les accès simultanés aux données identiques, et d'autre part, assurent la fiabilité et la disponibilité de l'accès aux données. L'algorithme de consensus maintient la cohérence de ces données.

Notre système de disque est composé de cinq modules : le premier gère l'accès aux données et réalise la répartition de la charge sur l'ensemble des serveurs de disques composant le disque virtuel ; le deuxième traduit les adresses virtuelles en adresses physiques ; le troisième assure la cohérence (l'intégrité) de l'état du disque virtuel sur chaque serveur ; le quatrième contrôle en permanence la disponibilité des serveurs de disques ; le cinquième se charge de restaurer la cohérence après une panne. La reconfiguration d'un disque virtuel, la modification du schéma de répartition des données, l'ajout et le retrait de serveurs de disques, sont transparents aux utilisateurs. Leurs exécutions ne nécessitent pas l'arrêt du système.

La répartition des structures, contrôlant la répartition des données entre les serveurs de disques, est assurée par l'algorithme de consensus *Paxos*. La propriété de Paxos est de garantir l'obtention d'un consensus pour une information sur un ensemble de machines asynchrones. Cette propriété est utilisée dans notre système pour le maintien de la structure de contrôle. La terminaison de ce consensus est garantie si une majorité de machines reste fonctionnelle durant le déroulement de l'ensemble des échanges le constituant.

Nous avons réalisé un serveur de disques virtuels fiable et disponible. Nous avons réalisé une implémentation de l'algorithme de consensus par processus légers qui respecte les contraintes temporelles d'accès aux données. Les mesures montrent que : 1/ les performances de notre disque virtuel augmentent linéairement avec le nombre de serveurs ; 2/ l'utilisation de stations de travail comme serveurs et clients de disque virtuel est une solution possible. Notre disque virtuel atteint un débit crête en lecture de 65 Mo/s avec 8 machines ; ce qui représente 93% de la bande passante du réseau Ethernet 100Mbits. Les modules d'état et de restauration assurant la fiabilité du système nécessitent l'intégration de *Paxos*.

Nous concluons sur les perspectives et les évolutions de notre système permettant l'intégration de l'algorithme de consensus *MultiPaxos* et la réalisation d'un système de fichiers distribué.

Remerciements

Je remercie Monsieur Jean Méhat pour ses précieux conseils et sa disponiblité sans réserve.

Je remercie Messieurs Edward Lee, Chandramohan Thekkath et Ivan Sipos pour leurs conseils.

Ce travail a été réalisé au sein du laboratoire d'Intelligence Artificielle de Paris 8. Je tiens à en remercier les membres et en particulier Erik Adelbert, Vincent Boyer, Karim Jaber et Nicolas Jouandeau pour leurs conseils et leurs encouragements.

Je remercie Raoul Kuczyk d'avoir maintenu le parc de machines du centre de calcul en parfait état.

Je remercie Maureen Chappuit pour ses encouragements et son soutien quotidien.

Table des matières

Chapitre 1

Introduction

Nous présentons dans cette introduction les motivations qui nous ont conduit à la réalisation d'un disque virtuel réparti qui se caractérise par l'absence d'un serveur centralisé. Nous commencons par exposer les besoins du partage de fichiers, puis nous donnons les caractèristiques et les désavantages des systèmes distribués existants, et finalement nous proposons notre solution en explicitant ses avantages.

1.1 Évolution des capacités de stockage des stations de travail individuelles

Les données qui ne sont pas stockées dans la mémoire vive, sont enregistrées sur un périphérique de stockage secondaire, généralement des disques, ou sauvegardées sur un support de stockage tertiaire comme les bandes magnétiques, sous la forme de *fichiers*[1]. L'accès à une bande est séquentiel, c'est à dire que les fichiers — leurs contenus en fait — sont stockés les uns à la suite des autres. Les *disques durs* sont organisés sous forme de *pistes* et de *secteurs*[2] et permettent un accès direct aux fichiers[3].

La lecture d'un fichier consiste à lire les blocs dans lesquels il a été enregistré. Pour faciliter cette organisation , une table stocke la correspondance entre blocs et fichiers. Cette table est stockée sur le disque. L'accès à cette table est réalisé par une interface d'abstraction qui présente les données sous la forme de nom de fichiers. Le *système de fichiers* utilise des *unités d'allocations*. Une unité contient la correspondance entre un fichier, c'est à dire le nom du fichier, et les secteurs qu'il utilise[4]. Ces unités sont regroupées sur une ou plusieurs parties du disque dans une

[1] Un fichier est un ensemble structuré d'informations qui constituent une unité pour un ordinateur, et qui est stocké sur un support permanent.

[2] Cette organisation trouva ses limites avec un nombre de secteurs plus grands que les registres des ordinateurs. D'autres unités se sont révélées nécessaires : les *blocs* (ou *clusters*). Un *bloc*, ou groupe de secteurs, est constitué d'un nombre déterminé de secteurs (deux ou plusieurs), chacun d'eux pouvant contenir un certain nombre d'octets, selon la capacité du support. Les groupes de secteurs sont utilisés par le système d'exploitation pour lire ou écrire des données.

[3] L'écriture de fichiers consécutifs et la lecture du disque de manière séquentielle pour retrouver un fichier est envisageable mais fait perdre le bénéfice de l'accès direct.

[4] Les apports du système de fichier sont plus nombreux. Le nom de fichier permet de distinguer les données sans en lire le contenu. Les fichiers sont effaçables indépendamments les uns des autres ;

table d'allocation.

La capacité des disques sur les stations de travail[5] est en augmentation permanente, sans commune mesure avec l'espace utilisé par le système d'exploitation. Nous constatons que cette augmentation[6], de par l'utilisation de fichiers intégrant des sons, des images et des vidéos, est exponentielle[CM99]. L'espace libre est dilapidé car l'abondance de place n'encourage pas les utilisateurs à gérer efficacement leur espace de stockage.

1.2 L'importance grandissante du partage de fichiers entre stations de travail indifferenciées

Deux cas peuvent être distingués pour les configurations usuelles de postes de travail. Le premier concerne les postes de travail configurés pour être parfaitement autonomes. Le second concerne les postes qui utilisent des serveurs pour partager un environnement commun. Chacune des configurations possède des avantages et des inconvénients que nous allons présenter.

La configuration de postes autonomes garantit la tolérance aux pannes puisque l'arrêt du service de l'un des postes ne concerne que celui-ci ; en revanche, il est nécessaire de dupliquer les fichiers sur chacun des postes où il sont succeptibles d'être utilisés. Cette redondance fait perdre de l'espace de stockage. Chaque mise à jour des postes de travail demande une intervention particulière[7]. Des problèmes de synchronisation doivent être résolus lorsque les fichiers sont modifiés[8]. Les postes de travail autonomes contiennent des données personnelles, liées à l'utilisateur du poste ; ces postes ne sont pas interchangeables. Ce type de configuration pose un problème d'encombrement, de prix et de maintenance.

La configuration des postes qui partagent un environnement commun n'ont pas les problèmes précédemment cités. Par contre ils ne sont pas autonomes et ont donc un problème de tolérence aux pannes. En cas de défaillance d'un serveur, le poste est inutilisable. De plus la disponibilité varie en fonction du nombre de postes connectés. Plus il y a de postes clients, plus le serveur est surchargé et moins il offre des temps de réponse acceptables. Ajouter des serveurs suplémentaires déplace le problème, car les serveurs coûtent le prix de plusieurs postes autonomes et le problème de la synchronisation des fichiers entre les serveurs apparait.

Chaque station conserve un ensemble de fichiers. Cet ensemble se décompose en deux parties. Une partie de ces fichiers concernent le système d'exploitation de la

le système de fichier libère des secteurs attribués au fichier, ce qui augmente l'espace disque libre. Les fichiers peuvent augmenter de taille sans nécessiter le ré-enregistrement complet ; le système de fichier ajoute à l'entrée du fichier des secteurs encore inutilisés.

[5]Nous ne faisons pas ici de distinction entre les stations de travail et les postes basés sur des ordinateurs personnels, puisque de toute évidence leurs rôles sont de moins en moins distincts, sinon en ce qui concerne des aspects mineurs qui ressortent plutôt du marketing.

[6]La capacité des disques des ordinateurs personnels est multiplié par deux tous les dix-huit mois. La capacité moyenne était de 3,2 Go en 1998, et est de 40 Go en 2002. Ces capacités sont des moyennes car les disques de 80 Go sont courant. Maxtor Corparation a annoncé le 10 septembre 2002 un disque dur de 320 Go pour la fin de l'année.

[7]La situation est plus difficile encore car les machines en panne ne peuvent pas être mises à jour.

[8]Le problème de la synchronisation est différent de celui de la mise à jour. La mise à jour consiste en une modification d'un fichier ou d'un ensemble de fichiers réputé comme valides. La synchronisation consiste en une mise à jour d'un fichier par la version plus recemment modifiée.

machine. Pour des raisons de performances les fichiers utilisés par le système d'exploitation doivent être placés sur un disque local étant donné la fréquence de leur utilisation[9]. Les documents et les applications, par opposition aux fichiers requis par le système, peuvent être placés sur un support de stockage partagé. La centralisation des applicatifs facilite leur mises à jour et augmente l'espace libre sur les stations de travail. Le partage des documents depuis un serveur de fichiers diminue la redondance des informations, facilite le travail coopératif entre les utilisateurs et permet de définir une politique d'accès précise.

1.3 Les systèmes actuels qui permettent l'échange de fichiers

La mise en commun des fichiers et des espaces disques disponibles est physiquement réalisable par le biais d'interfaces réseau. Ainsi tout fichier est partagé et accessible à partir d'une quelconque machine de ce réseau. Pour rendre ce partage transparent, l'accès aux fichiers sur la station de travail est réalisé par une interface qui émule le système de fichier local et permet d'accéder aux fichiers d'une machine serveur distante : le système de fichiers est *distribué*. Le rôle de l'interface est de dissimuler les communications réseaux et d'offrir les même services que le système de fichier local.

Un système de fichiers distribué repose sur un protocole de communication. Les interfaces qui utilisent un protocole de *transfert* de fichiers pour présenter un système de fichiers ne peuvent pas être considérées comme des systèmes de fichiers distribués. Le protocole doit supporter la sémantique d'accès aux fichiers[10]. Les protocoles de transfert de fichiers se distinguent par l'impossibilité d'accéder à une sous-partie d'un fichier de manière directe. L'adoption d'un protocole est soumise à son adaptatbilité à chaque système. D'une part le protocole doit rester simple. Un protocole qui repose sur les spécifités d'un système est difficilement transposable sur d'autres. D'autre part, le protocole doit supporter l'ensemble des fonctionnalités demandées par le client du système de fichier distribué et permises par le serveur qui stocke les fichiers.

Les protocoles NFS et SMB sont les plus anciens[11] des protocoles de partages et sont toujours utilisés. Les choix qui ont présidé à la conception de NFS étaient : interopérabilité, pour unifier la distribution de fichiers entre des systèmes différents et des protocoles différents ; et simplicité, avec un serveur de fichiers sans-état[12]. Si ces choix ont permis une large diffusion de NFS, ils laissèrent de côté des aspects important du partage de fichiers : les mécanismes de *caches*[13], la gestion de la

[9] Les terminaux graphiques utilisent les services d'un serveur spécialisé pour obtenir leur configuration réseau et leur système d'exploitation. Ce dernier est léger puisque sa fonction consiste uniquement à gérer l'affichage et les périphériques de saisie. Toutes les tâches de calcul et d'accès aux données sont réalisées sur le serveur. Les terminaux ne gèrent pas de périphérique de stockage de masse.

[10] La sémantique d'accès aux fichiers concerne le parcours et modification de l'arborescence, positionnement dans le fichiers, lecture et écriture de plages d'octets, création et suppression de fichiers.

[11] Le protocole NFS (Network File system) a été proposé par Sun en 1984, pour unifier le partage de fichiers sur les système UNIX. La même année, le protocole SMB a été proposé par IBM pour le partage de fichier sur les PC. Cette date correspond à l'adoption plus large des réseaux locaux avec la normalisation du protocole d'Ethernet en 1985.

[12] L'état des fichiers ouverts et de clients connectés n'est pas maintenu par le serveur.

[13] Un système de cache permet de limiter le traffic et les temps de latence du réseau. Chaque

concurrence en écriture[14], la *sécurité*, la *répartition*[15] des données. NFS repose sur les appels de procédure à distance (RPC). À chaque révision du protocole, les services RPC concernés nécessitent également une mise à jour[16]. Le protocole NFS a été faiblement utilisé sur les ordinateurs équipés de systèmes mono-utilisateur basés sur DOS ou Windows. Les systèmes de fichiers de ces systèmes d'exploitation ne possèdent pas d'attributs de propriété par fichier.

Le développement du protocole SMB a été repris en 1998 par Microsoft pour être intégré aux système DOS et Windows. Le protocole SMB est un protocole de partage de ressources : fichiers, imprimantes, appels de procédure à distance, ports de communication. Ce protocole est complexe et généraliste. De plus les extensions ajoutées par Microsoft sont mal documentées, voire soumises à licence, voire confidentielles. Pour utiliser les ressources distantes, une session doit être ouverte mais il existe une dizaine de méthodes d'authentification différentes avec leur enchaînement propre d'échanges de messages. Les extensions du protocole en SMB/CIFS donnent aux dernières versions de Windows, un niveau de sécurité plus acceptable, sans apporter la compatibilité espérée avec les systèmes UNIX [Ts'97].

Les systèmes de fichiers distribués qui centralisent les données, comme NFS, sont très sollicités et leur performances décroissent avec le nombre de machines clientes connectées. Ils nécessitent du matériel spécialisé et coûteux. Ils ne laissent comme alternative en cas de panne que l'attente de la reprise du service[17]. La fiabilité d'un système distribué peut être renforcée en dupliquant les données sur plusieurs serveurs.

Les systèmes de fichiers distribués avec *réplication*, comme CODA [SK90] et DFS [Mic02] qui sont issus de AFS, ont des problèmes de synchronisation lorsque les fichiers sont modifiés. Ces systèmes de fichiers avec serveurs de réplication sont plus adaptés au partage de données entre sites distants[18] . Leur mode de fonctionnement ressemble plus aux protocoles d'échange et de téléchargement de fichiers.

Les systèmes de fichiers *peer-to-peer*[19], sans serveur centralisé, sont compliqués à mettre en œuvre. Deux problèmes majeurs doivent être gérés en parallèle. Le premier concerne la répartition des données avec le choix du nœud de stockage des données. Le nœud de stockage peut être choisi pour sa stabilité ou pour sa proximité. Le second concerne la gestion de l'accès aux données avec la gestion des caches de données des clients. Si le cache d'un client peut être invalidé par un pair, le client

système d'exploitation possède ses optimisations internes au noyau, comme par exemple l'UNIX de Berkeley BSD avec son système de cache différé [Mac94]. Le cache augmente la *disponibilité* et la *transparence* d'utilisation. Le serveur sans état ne peut implémenter un système d'invalidation des caches clients pour la gestion de la *concurrence* des écritures.

[14]Le serveur NFS ne peux pas appliquer des verrous sur des fichiers, puisqu'il est sans état.

[15]Les implémentations du système de fichier NFS sont stables à force de corrections. Plutôt que de tout refaire, des projets de système de fichiers répartis utilisent ce protocole pour créer de nouvelles solutions. NFSP [LD02] utilise la version 2 de NFS pour répartir les fichiers, dont les noms et les emplacements sont décidé par un serveur de méta-données.

[16]Le système de fichier Coda utilise également les RPC. Pour permettre la synchronisation de plusieurs serveurs simultanément, le système RPC2 a été créé.

[17]Pire encore, le système des clients qui ne peut plus accéder au serveur, est bloqué sur les connexions comme s'il s'agissait d'une saturation du réseau.

[18]Le point fort de Coda est de supporter l'informatique mobile. Les stations connéctées à un volume Coda, répliquent les fichiers localement et permettent l'accès sans connexion. Les modifications sont reportées sur le serveur lors de la réintégration de la station au réseau. Mais les problèmes de synchronisation des versions de fichiers restent problèmatiques et non automatisables, du moins sans pertes.

[19]Cette appellation désigne les systèmes où chaque machine est à la fois client et serveur, c'est un système pair à pair. Un service *peer to peer* est un service partagé sur plusieurs nœuds par opposition à centralisé.

doit reporter les modifications sur le serveur. Une méthode plus judicieuse consiste à expédier les modifications au client qui souhaite accéder au fichier. Or la gestion des caches est liée aux blocs de données, donc au système de stockage. La gestion des caches au niveau du système de fichiers est *contre nature*. Enfin, les clients doivent dynamiquement tenir à jour une table des serveurs capables de répondre. Ces particularités sont difficilement intégrables dans les systèmes d'exploitations actuels[20].

1.4 Disques virtuels répartis et systèmes de fichiers spécifiques

Les systèmes de fichiers distribués existants montrent leur limites. Les systèmes centralisés ne supportent pas l'augmentation du nombre de client, offrent une faible tolérence aux pannes et la protection des données est assurée par le système de stockage. Les systèmes avec réplication permettent de répartir les accès et donc offrent plus de disponbilité et plus de fiabilité mais par contre la synchronisation des données est problèmatique. Les systèmes totalement répartis sont difficilement intégrables aux implémentations des systèmes actuels. Les systèmes de fichiers distribués sont les vestiges d'une vision du partage calquée sur les protocoles de transfert de fichiers comme FTP.

La gestion des caches de données, qui est un facteur important de l'amélioration des accès aux fichiers distants, est intégrée dans les systèmes d'exploitation au niveau du système de stockage. La réplication et la synchronisation des données est bien maîtrisée sur les systèmes de stockage avec les technologies RAID, matérielles ou logicielles. La gestion des fichiers et le transfert des données sont deux aspects distincts du stockage de fichiers.

Nous proposons une nouvelle approche avec un système de stockage réparti qui distribue les données et un système de fichiers qui exploite les spécificités de cet espace de stockage. Elle est adoptée de plus en plus dans le monde industriel avec les Storage Area Network (SAN), mais ce sont des périphériques de stockage spécialisés[21]. Cette réalisation permet d'exploiter les disques de grande capacité des stations de travail.

Les stations de travail coopèrent pour partager leur espace disque libre. Les accès à cet espace sont de type lecture et écriture de blocs. La répartition et la réplication sont réalisées par l'ensemble des disques des stations à la manière d'un contrôleur RAID qui contrôle un groupe de disques durs.

Les problèmes posés par la réalisation d'un tel disque réparti sont de deux ordres : la définition d'une structure d'adressage qui supporte dynamiquement le change-

[20]Serverless File system (xFS) [WAD98] est un système de répartition de fichiers sans serveur centralisé, la couche système de fichier actuelle dans les systèmes UNIX, fondée sur les *vnodes*, n'est pas suffisamment mature pour qu'on puisse y ajouter une gestion des caches. L'interaction entre la gestion des fichiers et les problèmes posés par la répartition conduit à une complexité qui rend la réalisation d'un tel système problématique.

[21]Les SAN possèdent une interface de communication de type réseau qui offrent des services de type lecture et écriture de blocs sur un réseau dédié. Ils sont autonomes en ce qui concerne le transfert de données. Un ou plusieurs serveurs de fichiers distribués se synchronisent pour accèder concurremment à cet espace de stockage. Il ne doivent pas être confondus avec les NAS qui sont des serveurs spécialisés de type serveur de fichiers centralisé avec les mêmes défauts que ces derniers.

ment du nombre de serveurs et la réalisation d'un mécanisme de tolérance aux pannes.

1.5 Plan de lecture

Nous consacrons le chapitre 2 à la présentation des systèmes de stockage répartis et aux techniques de distribution des données. Cette présentation nous permet de mettre nos travaux en perspective et de justifier l'architecture de notre système.

Nous décrivons dans le chapitre 3 l'architecture générale de notre système de disques virtuels distribués. Il distingue par sa haute disponibilité, son extensibilité et l'utilisation de périphériques de stockage conventionnels. Il consiste en un ensemble de serveurs connectés entre eux qui gèrent de façon coopérative leurs disques physiques. Sa principale caractéristique est d'assurer la répartition des blocs d'un disque virtuel et non des fichiers eux-mêmes. Ceci apporte une relative simplicité, par rapport aux systèmes de fichiers répartis traditionnels. Cette approche nous a permis de réaliser dans le cadre notre ce travail : la *réplication* des données entre les serveurs, qui garantit la continuité de fonctionnement dans le cas de panne de certains serveurs ; et la reconfiguration dynamique avec l'intégration de nouveaux serveurs en cours de fontionnement.

Dans le chapitre 4 nous présentons en détail l'algorithme réparti de consensus utilisé dans notre système afin d'assurer le maintien d'un état global cohérent : il permet à notre système de continuer à fonctionner quoiqu'avec une baisse de performance, tant qu'une majorité des serveurs est en mesure de communiquer à un moment donné.

Dans la conclusion nous présentons les étapes restantes pour intégrer l'algorithme de consensus général au disque virtuel présenté au chapitre 3, puis nous présentons les pistes que nous comptons suivre pour la réalisation d'un système de fichiers qui exploite les caractéristiques du disque virtuel réparti.

Chapitre 2

État de l'art sur les systèmes de stockage distribués

Ce chapitre est consacré à la présentation des systèmes de stockages répartis et aux techniques de distribution des données. Cette présentation nous permet de mettre nos travaux en perspective et de justifier l'architecture de notre système.

2.1 Systèmes de stockage

Nous présentons dans cette section les systèmes de stockage en général, sans nous attacher à leurs caractéristiques de répartition.

2.1.1 Périphériques de stockage de masse

Sur les ordinateurs, les données qui ne sont pas conservées dans la mémoire vive sont placées sur des périphériques dits de *stockage de masse*. Parmi ces périphériques, il est possible de faire la distinction entre ceux qui ne permettent qu'un accès séquentiel aux données qui y sont stockées (périphériques à *accès séquentiel*) et ceux qui permettent d'accéder directement à n'importe quelle partie des données (périphériques à *accès directs*) [1].

Les disques magnétiques

Les périphériques les plus couramment utilisés pour le stockage de masse à accès direct sont les disques durs. Ils se composent de surfaces magnétisées sur lesquelles se déplacent un bras. Ce bras porte des têtes qui permettent l'écriture et la lecture de signaux.

[1] La distinction entre accès direct et accès séquentiel est en fait arbitraire ; il est parfaitement possible de considérer que le déroulement d'une bande magnétique pour amener un bloc sous la tête de lecture est du même ordre que le déplacement du bras de lecture sur un disque magnétique ; seul le temps nécessaire change. On trouve d'ailleurs des systèmes de fichiers qui sont dédiés au stockage des données sur les bandes magnétiques comme *TapeFS* [Lab00] sous *Plan 9*.

La capacité des disques durs disponibles à l'heure actuelle est de quelques dizaines de Giga-octets[2]. Les caractéristiques importantes pour la mesure des perfomances des disques magnétiques sont principalement la vitesse de transfert de données, qui est de l'ordre de la dizaine de Méga-octets par seconde sur les disques courants, et le temps de positionnement des têtes de lecture qui est de l'ordre de la dizaine de millisecondes[3].

Le système d'exploitation écrit les données sous forme de blocs d'octets appelés des *secteurs*, qui ont en général des tailles de l'ordre du Kilo-octet. Le système d'exploitation s'adresse au contrôleur de disques qui ne permet que la lecture et l'écriture de secteurs entiers. Chaque secteur est identifié par un numéro[4].

2.1.2 Systèmes de fichiers

L'adressage des données sur le disque sous la forme de secteurs adressés par leurs numéros ne convient pas à l'organisation du stockage de données hétérogènes. Pour cette raison, l'interface avec le reste de l'ordinateur se fait sous la forme d'un *système de fichier*. Celui-ci offre la possibilité de regrouper les données en *fichiers*, qui représentent une partie des données stockées sur le disque avec des attributs : un nom, des caractéristiques de sécurité.

Organisation des données

Ces fichiers sont organisés d'une façon hiérarchique, les fichiers étant regroupés dans des répertoires, qui sont eux-mêmes assimilés à des fichiers spécialisés.

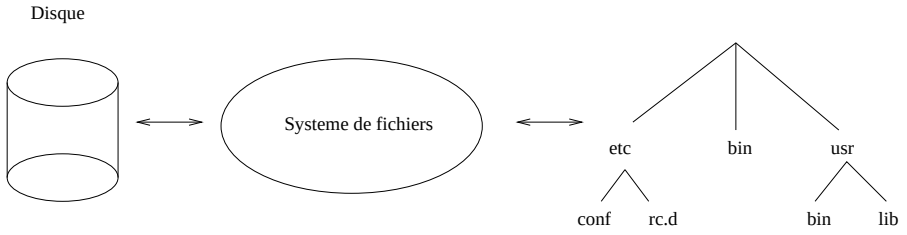

FIG. 2.1 – Le système de fichiers sert d'interface entre la réalité physique du périphérique de stockage et la présentation sous la forme d'une arborescence de fichiers.

Le rôle du système de fichiers est de faire la traduction entre la réalité physique du stockage sur le disque magnétique et l'organisation hiérarchique des fichiers.[5].

[2] Les serveurs de stockage, ou baies de disques, présentent actuellement des capacités de l'ordre d'une dizaine de Téra-octet. Au regard de l'évolution de capacités utilisées durant les 5 dernières années, elles devraient être de l'ordre du Péta-octet en 2007.

[3] À la différence de la capacité et du débit qui augmentent régulièrement, les temps de positionnement des bras sont quasi-constants.

[4] Le fait que certaines interfaces divisent le numéro de secteur en numéro de cylindre, numéro de tête et numéro de secteur sur la piste ne change rien : ces numéros combinés entre eux forment le numéro du secteur ; certains protocoles, comme l'interface SCSI, ne permettent même pas de connaître ces valeurs.

[5] La réalité physique du stockage peut être plus complexe qu'un simple disque dur : elle peut

Le rôle du système de fichier est d'assurer la création, la modification et la destruction des fichiers. De plus les fichiers peuvent changer de taille. Il est important d'optimiser le placement des données de façon à minimiser les mouvement du bras de lecture.

2.1.3 Les techniques des systèmes de fichiers

Nous présentons ici les techniques liées à l'implémentation et à l'optimisation des systèmes de fichiers.

Le système de fichiers virtuel

Il y a une vingtaine d'années, les systèmes d'exploitations était développés avec leur propre système de fichiers et ne supportaient que celui-ci. Kleiman proposa une architecture [Kle86] permettant l'accès à plusieurs systèmes de fichiers sur le système SunOS[6]. Le *Virtual Filesystem Switch* repose sur une structure de base appelée *vnode* et masque les différences entre les systèmes de fichiers. Le système d'exploitation manipule des descripteurs de fichiers. Le système de fichier manipule au moyen de structures complexes, la correspondance entre le nom des fichiers et les secteurs qu'ils occupent. L'interface VFS sépare les aspects spécifiques du système de fichier et ceux du système d'exploitation.

Mais la motivation initiale pour la réalisation de l'interface VFS a été surtout l'intégration des systèmes de fichiers distribués sans insérer l'accès à distance dans l'implantation du système de fichier local existant. À l'époque il s'agissait d'intégrer NFS à SunOS. Le concept de VFS a été repris dans le noyau Linux qui supporte actuellement plus de 50 systèmes de fichiers.

Volume logique

Un volume logique, ou volume virtuel, est un disque virtuel sur lequel un système de fichiers stocke des données. Un volume est composé d'une ou plusieurs partitions[7] sur un ou plusieurs disques physiques.

Nous avons montré dans la section 2.1.2 que le système de fichiers cache à l'utilisateur la réalité de l'adressage par blocs du disque. Les volumes aggrégés apparaissent avec la technologie RAID : les disques et les partitions sont associés pour former de plus grandes capacités de stockage. Les blocs de chacun des périphériques de stockage physiques ont un numéro unique dans le volume logique : le numéro de

concerner soit seulement une partie d'un disque, comme dans le cas d'une partition, soit plusieurs disques avec ou sans réplication comme dans le cas du RAID, soit des dispositifs éventuellement indisponibles au système de fichiers comme dans le cas de certains systèmes de fichiers répartis.

[6]Trois systèmes de fichiers locaux etait en concurrence : L'Unix File System (UFS) de Sun, Le Berkeley Fast File system (FFS) et le système de fichier DOS.

[7]La partition est une subdivision d'un disque physique. La table des partitions est stockée au début du disque. Le principe du partitionnement est nécessaire pour palier au manque d'extensibilité des systèmes de fichiers traditionnels. Les systèmes de fichiers statiques ne peuvent adresser qu'un nombre fini d'unités d'allocations dont la taille dépend de l'étendue de la partition ; la capacité de stockage réduit avec la quantité d'unités d'allocations requises par le stockage des fichiers. Chaque partition possède son système de fichiers. Ce cloisonnement limite les erreurs du système de fichiers à la partition concernée.

bloc logique. La correspondance des blocs réels sur les disques et des blocs logiques d'un volume est assurée soit par un contrôleur RAID matériel soit par un module logiciel intégré au noyau du système pour des raisons de performances. Dans le premier cas, le volume est vu par le système d'exploitation comme un disque physique. Dans le second cas, le périphérique de stockage est appelé disque virtuel et présente les même caractéristiques d'accès qu'un disque physique grâce au gestionnaire de volume du système d'exploitation.

La notion de disque logique introduit une nouvelle interface [dJKH93] qui permet de séparer la gestion des fichiers de celle des disques en utilisant des numéros de blocs logiques. La gestion par blocs logiques et listes de blocs permet de reconfigurer la disposition de blocs pour améliorer les performances, notamment pour l'adaptation à plusieurs types de systèmes de fichiers.

Il est possible également de partager un disque via le réseau. Une émulation d'un disque virtuel partagé par plusieurs machines est décrite dans [ABP+94] et réalisée sous Linux par Pavel Macheck en 1987 sous le nom de *Network Block Device* [BLA00]. L'accès à un disque est réalisé en redirigeant les accès au périphérique vers le réseau. La distribution d'un disque entre plusieurs machines est possible mais il n'existe à priori aucun mécanisme d'exclusion pour les accès concurrents simultanés.

Journalisation du système de fichier

La journalisation des systèmes de fichiers permet de gérer des disques de grande capacité, un grand nombre de fichiers, diminue la durée des arrêts pour maintenance. Les performances des accès aux fichiers de petite taille et à ceux de grande taille sont équivalentes. La fragmentation du disque et la fragmentation des fichiers est réduite.

La journalisation améliore les performances d'un système de fichiers en terme de rapidité d'accès et de fiabilité. La première caractéristique est obtenue par écriture séquentielle d'enregistrements dans un journal correspondant à la modification de blocs sur le support de stockage [DST87] ; elle permet de minimiser les temps de latence pour l'accès aux pistes du disque en retardant l'exécution des écritures. En cas de panne, le journal est relu et les opérations non appliquées sont répercutées sur le système de fichier. La seconde est l'utilisation d'un système de stockage fiable plus coûteux mais réservé au journal.

Le gain apporté par un système de fichiers journaliser ne peut être ignoré avec les périphériques de stockage volumineux actuels[8]. Le développement de techniques de journalisation est très actif actuellement[9].

[8]La vérification de l'intégrité du système de fichier natif *Ext2FS* de *Linux* après plantage est de l'ordre de la minute par Giga-octets.

[9]Sous Linux le système de fichier *ext3* [Twe98] est une extention du système de fichiers natif *Ext2FS*, autorisant la migration de l'un vers l'autre . *ResiserFS* [Rei00] est un implantaion complète d'un système de fichiers journalisé.

Capacités d'extension

L'espace de stockage augmente continuellement, et avec lui le nombre de fichiers et de répertoires dans le système de fichiers. Un système de fichiers moderne doit s'adapter à de plus grands volumes sans perte de fiabilité ni de disponibilité. Le système XFS de *Irix* [SDH+96] a été conçu pour s'adapter à de grands volumes.

La correction de la table d'allocation des fichiers après une panne brutale augmente avec la taille du volume de stockage[10]. L'utilisation d'un système journalisé réduit ce temps de maintenance. Seules les dernières opérations écrites dans le journal mais non répercutées sur le système de fichier, sont relues et éxécutées.

Le nombre maximal d'unités d'allocation, ou *inodes*, est fixé lors de la création du système de fichier. Si ce nombre est atteint, la seule possibilité est de sauvegarder les données et de recréer le système de fichier avec une autre limite. La table d'allocation occupera alors plus de place sur l'espace de stockage. *L'allocation dynamique* d'inodes simplifie l'administration[11]. du système de fichiers. Mais la technique d'allocation dynamique d'inodes est complexe et ralentit la gestion du système.

Les systèmes de fichiers avec une taille d'allocation fixe s'accomodent mal de l'augmentation de la taille des volumes de stockage[12]. Une unité d'allocation d'espace de taille variable ou *extent* stocke le numéro du premier bloc, le nombre de blocs utilisés par le fichier, ainsi que la position de ces données dans le fichier. Un fichier est stocké sous la forme *d'extents*, qui permettent d'utiliser de petites unités d'allocations. Le bénéfice du procédé est perdu si chaque *extent* n'adresse que peu de blocs logiques. Cette technique, issue des systèmes de gestion de bases de données, a été intégrée dans des systèmes de fichiers journalisés comme XFS [SDH+96], ReiserFS [Rei00] et Ext3FS [Twe98]. La gestion des fichiers à trous, ou *sparse files*[13], est une autre des possibilités offertes par les *extents*.

La recherche de fichiers dans un répertoire est séquentielle car les structures utilisées sont des listes de structures. Avec des répertoires contenant des milliers de fichiers et de répertoires, les temps de recherche deviennent inacceptables et l'utilisation d'autres algorithmes et d'autres structures est impératif. Par exemple les arbres équilibrés, ou *B+ Tree*, sont utilisés depuis une vingtaine d'années dans les SGDB. La recherche de blocs libres pour les grands fichiers, stockés sous la forme des tables de bits habituellement, est inefficace pour trouver des plages de blocs contigues. Les *B+ Tree* donnent de meilleurs résultats. Les *B*-Trees* sont utilisés

[10] Les commandes de restauration, comme *fsck* sous UNIX, ont une durée liée à la taille du volume de stockage et à son taux d'occupation. Dans la pratique, la taille de la mémoire est un paramètre supplémentaire important, puisque la lecture intensive des tables d'allocations pendant leur vérification, est réduite par la mémoire-tampon.

[11] Lorsque le nombre maximal d'i-nœuds est atteint, le système de fichiers doit être reconfiguré avec d'autres valeurs. L'administrateur doit évaluer les contraintes imposées par la taille des tables d'allocations, du nombres d'i-nœuds nécessaires. Cette manipulation requiert la sauvegarde et la restauration des données sur un autre support.

[12] Si l'unité d'allocation est trop petite, c'est à dire proche de la taille d'un bloc logique, l'enregistrement des grands fichiers nécessite un grand nombre d'unités qui remplissent inutilement l'espace de stockage. Un grand fichier utilise de nombreux blocs qui, s'ils ne sont pas contigus, augmentent les temps d'accès ; c'est la fragmentation externe. Avec de grandes unités d'allocations, les petits fichiers n'utilisent qu'une petite partie d'une unité et l'espace restant est inutilisé. De façon plus générale les fichiers dont la taille n'est pas divisible par une unité sous-utilisent la dernière unité d'allocation. Cette sous-utilisation des blocs logiques est également appelée fragmentation interne.

[13] Il est inutilie d'allouer la totalité des inodes pour un fichier sur lequel ont été écrits les 10 premiers octets puis 1 millions d'octets plus loin, 10 octets supplémentaires. Deux extents adressant chacun un bloc de 10 octets suffisent pour stocker les informations.

depuis une dizaine d'années par Apple dans le *Hierarchical File System* de *MacOS* et XFS implémente des *B+Tree* [SDH+96].

Enfin, la taille des structures et les valeurs entières stockées sur 32 bits, ne permettent pas d'adresser les tailles de volumes et de fichiers existant ou à venir. Avec un pointeur sur 32 bits et un bloc d'allocation de 8 Kilo-octets, un système de fichiers est limité à une taille maximale de 32 Téra-octets.

Fiabilité

La fiabilité est la capacité d'un système de fichier à se conformer à ses spécifications sur une période donnée. Les fichiers doivent êtres écrits sur les blocs du système de stockage et la lecture de ces blocs doit restituer le fichier enregistré sans altération.

L'approche traditionnelle consiste à conserver de manière fiable la dernière version de chacun des fichiers. Le système de fichiers *ElephantFS* [SFH+99] propose une technique d'enregistrement par version. Chaque modification de l'arborescence ou des fichiers est conservée et met l'utilisateur à l'abri de fausses manœuvres. Ce système requiert un stockage fiable. En revanche, dans le système *Elephant FS*, il n'existe pas de mécanisme automatique pour éliminer les versions les plus anciennes des fichiers ou pour faire migrer ces versions sur un support de sauvegarde tertiaire[14]. Le taux d'occupation utile de l'espace de stockage s'en trouve fortement réduit.

2.2 Les systèmes de fichiers distribués

Nous présentons dans cette section les caractéristiques des systèmes de fichiers distribués, et nous décrivons les qualités qu'ils cherchent à présenter : uniformité, fiabilité et performance.

2.2.1 Définitions

Un système de fichiers est dit distribué[15]dès lors que les données ne peuvent pas être conservées sur un périphérique de stockage qui appartient à la machine locale, mais que l'interface d'accès à ces données est identique avec celle qui permet l'accès aux fichiers locaux.

Dans le cas le plus courant, les données sont conservées sur d'autres ordinateurs reliés au poste client par un réseau local[16].

[14]Les périphériques de sauvegarde sont appelés supports tertiaires. Généralement ce sont les périphériques à bande.

[15]Le terme *distribué* qualifie les services fournis à des machines clientes distantes. Nous utiliserons le terme *réparti* pour qualifier un volume de stockage qui regroupe des espaces de stockage situés sur plusieurs périphériques ou sur plusieurs machines distinctes.

[16]Le réseau, reliant l'ordinateur clients au serveur où les données sont conservées, n'a aucunement besoin d'être un réseau local ; c'est cependant le plus souvent le cas pour des raisons de performances ; quand le réseau est trop lent, la lenteur des transferts de données rend en général préférable l'utilisation d'une autre interface.

L'interface offerte par des clients et serveurs de protocoles spécialisés, comme FTP, ne peut pas être qualifiée de système de fichiers distribué. dans la mesure où elle diffère totalement de celle qui permet l'accès aux fichiers locaux[17].

2.2.2 Uniformité de vue des fichiers

Le besoin en systèmes de fichiers distribués a commencé à devenir crucial lors du remplacement des mini-ordinateurs centraux par des stations de travail séparées : il était alors souhaitable d'éviter les recopies inutiles de données et de permettre à toutes les stations d'offrir un environnement uniforme.

Les premières stations de travail avaient des disques de taille modeste, voire inexistant dans le cas des stations sans disque. Il était donc nécessaire de leur permettre d'accéder via le réseau aux données stockées sur des serveurs qui centralisaient les capacités de stockage.

D'autre part, le partage des stations de travail entre les utilisateurs rend souhaitable que toutes les stations disposent d'un environnement équivalent. Cette transparence permet le travail sur n'importe quelle station. Cela n'est possible qu'en offrant de toutes les stations, une vue uniforme des fichiers stockés sur le réseau.

Finalement, avec l'apparition des clusters de calcul, la migration des tâches impose que la vision de l'espace de stockage soit identique depuis tous les nœuds du cluster.

2.2.3 Fiabilité de l'accès aux fichiers

Les systèmes de fichiers répartis d'améliorer la disponibilité des données à travers deux mécanismes antagonistes : la centralisation et la réplication.

La centralisation sur une machine simplifie l'administration et la maintenance ; elle est plus facile sur un ordinateur unique ; les sauvegardes sont simplifiées.

La réplication consiste à stocker les fichiers sur plusieurs disques différents, éventuellement connectés à des ordinateurs différents ; cela permet de garantir que le mauvais fonctionnement d'un des ordinateurs qui conserve une des copies n'empêche pas irrémediablement d'y accéder puisqu'on peut en trouver une copie sur un autre ordinateur.

2.2.4 Performance des lectures et des écritures

Depuis une vingtaine d'années, la vitesse des réseaux locaux est du même ordre de grandeur que celle des bus internes des ordinateurs. Par contre les vitesses de lecture sur les disques ne s'améliorent que lentement du fait des faibles progrès sur les temps de positionnement des têtes de lecture. De ce fait, un transfert à travers le

[17] L'existence d'un démon qui se charge de faire la traduction entre un protocole local et le protocole FTP, comme le démon `ftpfs` du système Plan 9, rend cette distinction délicate ; cependant, ce démon met en place une stratégie de gestion de cache qui lui est propre et qui n'appartient en aucune façon au protocole FTP.

réseau de données stockées dans la mémoire d'un ordinateur voisin est plus rapide que la lecture de ces mêmes données sur le disque local[18].

2.3 Éléments de typologie des systèmes de fichiers distribués

Dans cette section, nous catégorisons les propriétés des systèmes de fichiers distribués. Nous illustrons ces propriétés en donnant des exemples de systèmes déployés ou bien expérimentaux.

2.3.1 Sémantique des opérations sur les fichiers

Il existe une sémantique au niveau de l'accès aux fichiers et une sémantique de partage des fichiers.

Gestion de la sémantique sous NFS

Le fait que NFS soit un protocole sans état l'empêche de transcrire exactement la sémantique des opérations sur les fichiers du système Unix pour lequel il a été initialement conçu.

Sous Unix, les opérations sur un fichier doivent être encadrées par une opération d'ouverture et de fermeture, or la notion de fichier *ouvert* n'a pas d'équivalent dans un serveur sans état. Sous Unix, il est possible, une fois un fichier ouvert, de continuer à y lire et y écrire même si le fichier est détruit[19] ; cela est impossible avec le protocole NFS pur. La façon dont le problème est résolu consiste à retarder la destruction effective du fichier du coté du serveur, lorsqu'un client demande sa destruction ; à la place, le fichier est renommé avec un nom temporaire unique jusqu'à l'expiration d'un délai de grâce.

2.3.2 Sécurité de l'accès aux données

La sécurité de l'accès aux données doit être appliquée à deux niveaux : Un mécanisme qui authentifie les accès de la machine au serveur de fichiers et une authentification des utilisateurs du système de fichiers.

La sémantique des systèmes de fichiers inclue généralement des attributs de propriété[20]. Ces attributs doivent être conservés dans la sémantique de partage des systèmes de fichiers distribués.

[18] Le temps moyen d'un positionnement de tête est de l'ordre de la dizaine de millisecondes, c'est à dire le temps nécessaire pour transférer 1 Mega-bits sur un réseau local ordinaire.

[19] Cette caractéristique est largement employée par les utilitaires Unix pour avoir des fichiers temporaires : une application crée et ouvre un fichier, puis le détruit mais continue à opérer sur ce fichier, ce qui garantit que ce fichier temporaire disparaît quand le processus de l'application se termine.

[20] Le système de fichier du système mono-utilisateur DOS n'a pas été conçu avec des attributs de propriété par utilisateur.

Exemple de NFS

Le protocole NFS repose sur un partage des identités des utilisateurs qui est transmise avec chaque requête. Dans un environnement Unix homogène, elle suppose que les base de données utilisateurs soient identiques du coté du client et du serveur ; cela peut notamment se faire à l'aide des sc nis, un autre protocole de Sun, fondé également sur le même mécanisme de sc rpc.

Le serveur NFS fait confiance au client pour transmettre correctement l'identité de l'utilisateur ; il est possible, en modifiant le client, d'utiliser de fausses identités pour contourner les mécanismes de protection des fichiers.

La brèche de sécurité la plus évidente ouverte à ce point est obturée par deux mécanismes : la limitation des clients et des dispositifs spécifiques pour les utilisateurs privilégiés. Au début des opérations entre le client et le serveur, le serveur vérifie que le client apparaît dans une liste limitative des ordinateurs qui disposent du droit d'accéder à cette hiérarchie de fichier[21], éventuellement en lecture seule. Le serveur d'autre part transforme, si on l'a configuré pour cela, les requêtes en provenance des utilisateurs privilégiés (dont l'identité vaut 0) vers un utilisateur anonyme dont les droits sont minimaux.

Pour les autres systèmes d'exploitation , il est nécessaire de mettre en place des mécanismes ad-hoc de transformation entre l'identité locale au client et les numéros d'utilisateurs du coté du serveur.

Un autre problème réside dans la façon dont les clients acquièrent un descripteur de fichier initial à la racine de l'arborescence exportée par le serveur. Un aspect spécifique du protocole existe à cette effet, dénommé le *mount protocol*; le client transmet un chemin de données et reçoit en retour un *file handle*. Cependant, la vérification des droits d'accès n'est faite par le serveur que pour le répertoire passé au *mount server*. Il est donc possible, dans une arborescence qui n'est pas accessible sur le serveur parce que l'un des catalogues du chemin d'accès est protégé, d'accéder à un sous-catalogue qui lui ne l'est pas.

Les échanges de données ne sont pas encryptés, si bien qu'ils peuvent être interceptés facilement.

De nombreuses tentatives ont été faites pour résoudre ces problèmes dans NFS version 2 et 3, y compris avec des mécanismes d'échanges de clés publiques Diffie-Hellman [Sch94]. Étant donné la la base existante d'installations qui utilisent NFS et à cause de problèmes confus de stratégie industrielle, ces extensions n'ont jamais connues une large acceptation.

2.3.3 Concurrence et synchronisation

Gestion de la concurrence sur NFS

Le protocole NFS n'inclue pas de système interne de verrouillage des fichiers pour les accès concurrents. Ceci est dû principalement au fait que le serveur NFS est

[21] Comme l'identité du client n'est reconnue que par l'adresse IP qu'il utilise pour se connecter au réseau, qui peut facilement être empruntée, cette solution se contente de déplacer le problème.

sans état, ce qui lui interdit de conserver une liste des fichiers verrouillés. Dans la plupart des implémentations des versions 2 et 3, on trouve, à coté du serveur NFS, un démon spécifique qui est chargé d'assurer le verrouillage des fichiers. Ce démon, le *lock manager*, est un démon à état, ce qui lui permet de gérer ces verrous. Le serveur NFS s'adresse au *lock manager* lors de l'accès aux fichiers . Les verrous sont posés par les clients au niveau des fichiers, et non des enregistrements, ce qui assure un accès exclusif au prix de verrouillages à gros grains.

2.3.4 Politique de gestion des caches

L'utilisation d'un cache de données[22] sur le client est un facteur majeur d'accélération de l'accès aux données. Les données, une fois transférées à travers le réseau, sont stockées dans le cache du client et tous les accès sont réalisées depuis la machine locale. Lorsque le cache est implanté en mémoire comme celui des disques locaux, il est rapide mais sa taille est limité par la quantité de mémoire de la machine cliente. Sur un WAN, où les débits entre les nœuds de communications sont faibles, l'existence d'un cache de grande capacité est primordiale et rejoint le principe de la réplication des données.

L'inconvénient d'un système de cache est le problème de la cohérence des données[23]. Lorsque les données sont changées par le client, les modifications doivent être écrites sur le serveur, le support de stockage stable. L'utilisation d'un cache en écriture améliore la disponibilité mais l'intégrité des données est menacée par les défaillances possibles du système. La suppression du cache en écriture[24] est très pénalisante lorsque le système est stable mais réduit la probabilité de pertes importantes de données.

La défaillance du serveur provoque la perte des modifications de tous les clients connectés. Un protocole sans état comme NFS [PJS+94] ne peux pas gérer la réintégration des caches après une reprise sur panne. Le système *Spritely* NFS [Mog94] gère ce problème de cohérence des caches avec une table de contrôle des caches clients, stockée sur le serveur.

La gestion de la réintégration des caches a longtemps été repoussée sur NFS car les accès concurrents en écriture sont rares. Les enseignements de Spritely NFS [SM89] et de Coda [KS92, KS95] ont été intégrés dans NFS version 4 [PSB+00] sous la forme de bails. Le serveur accorde un bail lors de l'ouverture d'un fichier. Ce bail doit être renouvelé par le client à intervalles réguliers. Lorsqu'un fichier est modifié, les clients renouvelant leur bail sont avertis de la modification du fichier. Un mécanisme de rappel des clients par le serveur autorise la révocation des caches. Le système *Echo* utilise un système de jetons et bails pour implanter la cohérence dans un cache différé[25] [MBH+94].

[22]Le cache de données des disques est similaire au système d'anté-mémoire, qui stocke dans une mémoire très rapide les données présentes en mémoire et utilisés fréquemment par le processeur. Les blocs des disques sont placés en mémoire pour réduire les temps de latence du disque.

[23]La cohérence est la propriété selon laquelle les données contenues dans la cache de données reflètent bien la réalité qu'elles représentent. Quand la cohérence des données est respectée, il n'y a pas de contradiction entre les données et elles ne créent pas de confusion pour ceux qui les utilisent. La cohérence des données inclut la validité des données et l'intégrité des données.

[24]Les données à écrire sont renvoyées modifiées dans le cache et retournées au serveur immédiatement. Cette méthode est appelée *write-thru*.

[25]L'écriture des données présentes dans le cache est retardé jusqu'à l'invalidation de celles-ci — *delayed write-back*.

Le projet Intermezzo [Bra99, BN99] montre que des performances similaires aux systèmes de fichiers locaux peuvent être atteints avec une méthode de *write-back*[26]. Pour cela, le serveur doit posséder une méthode de *call-back* vis à vis du serveur de cache[27]. Intermezzo est d'abord un prototype destiné à montrer qu'un système de fichiers aussi riche que *Coda* peut être implémenté d'une manière plus simple.

L'utilisation des caches de données est poussée à son paroxysme dans le système Coda qui est l'un des nombreux descendants de AFS, pour *Andrew File System*. Lorsqu'un client veut lire un fichier, il est demandé au serveur, puis lorsque le fichier est fermé, le fichier est retourné au serveur si des modifications ont été réalisées. Ce système de cahche de données est proche de la réplication de fichiers.

2.3.5 Réplication des données

La réplication des données est un élément de fiabilité et de disponibilité des systèmes de fichiers distribués. L'existence de plusieurs copies d'une donnée diminue les possibilités de sa destruction ou la panne d'un serveur ou du réseau. Mais la contrepartie est le problème de la synchronisation des copies lorsque l'une d'entre elles est modifiée volontairement. Lorsqu'il existe plusieurs exemplaires d'un fichier, la disponibilité augmente car il existe plusieurs sources de diffusion entre lesquelles sont répartis les accès des clients.

Le système de fichier Coda implante un système de réplication par volume ; le volume *Coda* est un ensemble de répertoires d'un système de fichiers. Le volume est répliqué par un ensemble de serveur, un groupe de stockage de volume VSG. La cohérence entre les copies est assurée par des *Call-backs*. Un client retourne les modifications faites sur un fichier, aux serveurs accessibles AVSG, sans rien tenter pour les serveurs défaillants. C'est lors d'un accès ultérieur à un fichier sur le serveur le plus proche du AVSG, qu'un client préviendra l'ensemble des serveurs, par la diffusion de la version du fichier. Les serveurs qui possèdent une copie plus ancienne se synchroniseront avec ceux possèdant une version à jour. La synchronisation de copies divergentes, pour laquelle il n'existe pas de solution automatique respectant l'intégrité des données, a été partiellement résolue dans Coda [KS95]. Les utilisateurs peuvent lier des commandes de réparation à des types de fichiers[28].

La réplication des données au niveau des fichiers n'est pas efficace à cause de la taille très variable de ceux-ci, même si la plupart des fichiers utilisés sont généralement de petite taille et que le partage de fichier est rare [Sat81].

2.4 Techniques des systèmes de fichiers distribués

Dans cette section nous présentons les techniques utilisées pour les systèmes de fichiers distribués en terme d'interconnexions, de transport, de partage des données et de mise en œuvre.

[26] Les données sont écrites seulement lorsqu'elles sont vidées du cache.
[27] Le serveur devient client et initie une requête.
[28] La fusion de deux carnets d'adresses stockés sous formes de lignes de texte est possible, le code source de deux programmes divergents ne l'est déjà plus.

2.4.1 Réseau et méthodes de communication

Topologie réseau et stockage

Les SAN[29] sont des sous-réseaux haut-débits de périphériques de stockage. Ces périphériques sont composés d'un disque ou d'une grappe de disques avec leur propre adresse réseau. Actuellement les efforts de mise en œuvre visent à améliorer les débits sur ce réseau dédié. Trois normes de liaisons sont en concurrence :

- Fast-Ethernet, avec un débit de 100 Mbits/s ou 1 Gbits/s, est la plus simple des solutions à mettre en œuvre et la moins coûteuse. De plus l'arrivée des interfaces gigabits (10 Gbits/seconde) en ethernet supplante le FibreChannel en ce qui concerne le débit.
- FibreChannel (FC) avec un débit minimum de 100Mbits est une norme complète[30] qui se propose de remplacer ethernet en raison de sa plus faible consommation en ressources de calcul.
- iSCSI (SCSI-over-IP) est une proposition initiée par IBM en 2001 pour encapsuler les commandes SCSI sur le protocole IP. Cette norme permet d'interconnecter des périphériques de stockage NAS, souvent en TCP/IP, et les SAN connectés principalement par des interfaces SCSI. Mais la charge de calcul requise par le protocole IP reste trop importante, et ce malgré des cartes réseau dédiées TOE[31]. La technologie risque de disparaitre après l'annonce du désengagement de IBM en juin 2002[32].

Protocoles de transport

Les serveurs de stockage spécialisés : Network-Attached Storages

L'utilisation dominante de NFS, malgré ses faiblesses pour la montée en charge, a initié de nombreuses recherches pour réduire le goulot d'étranglement du serveur : parallélisation des requêtes d'accès, diminution du temps de redémarrage après défaillance, réduction au minimum de la maintenance. Ces études ont débouché, pour NFS et CIFS[33], sur des serveurs spécialisés avec un micro-noyau : les NAS (Network-Attached Storage). Ils gèrent une baie de disques de stockage et une ou plusieurs interfaces réseau limitées à Ethernet et TCP/IP pour la couche de transport. L'interface d'administration est généralement limitée à un service http ou telnet pour le nommage des volumes, les droits associés ainsi que les protocoles de partage à activer.

---

[29] Storage Area Network

[30] La norme peut être calquée sur le modèle OSI avec ses câblages et son type de signal, son format de trame et leur séquencement, multiplexage et protocole de transports (SCSI, IP, ATM, SBCCS, HIPPI...)

[31] TCP Offload Engine.

[32] Finalement, le consortium SNIA (Storage Networking Industry Association — `http://www.snia.org/`) a annoncé le 4 septembre 2002 la finalisation du rapport technique du standard auprès de l'IETF (Internet Engeneering Task Force — `http://www.ietf.org/internet-drafts/draft-ietf-ips-iscsi-17.txt`) IPS Working Group. Le iSCSI est prêt pour un déploiement industriel.

[33] Le protocole CIFS est utilisé par les serveurs Microsoft et par leur équivalents Unix comme Samba qui tentent de proposer une solution proche de l'originale malgré l'inexactitude des spécifications, l'incompatibilité avec les standards [Ts'97] ou bien encore les menaces de procès pour violation de brevets `http://us6.samba.org/samba/ms_license.html`, `http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnkerb/html/Finalcifs_LicenseAgrmnt_032802.asp`

Sans connexion (UDP)
L'établissement d'une connexion point à point représente une part non négligeable du temps de latence, surtout si elle est réitérée fréquemment. C'est pour cette raison que les RPC, le principe de communications sur lequel NFS [PJS+94] repose, utilisent le protocole UDP. Mais comme ce mode de transmission n'est pas fiable, il est nécessaire d'adjoindre au système les mécanismes ad-hoc.

Par exemple, ARDP [SON99] est un protocole fiable de requêtes et réponses sur UDP qui permet d'éviter le coût de la connection TCP. Il s'adapte à la bande passante tout en garantissant la fiabilité des échanges. Son mécanisme de décongestion du réseau donne des résultats légèrement supérieurs à TCP. C'est donc sur des échanges de petites tailles qu'il peut rivaliser. ARDP fait partie du projet Prospero [NR94].

Mais l'utilisation de UDP sur un WAN est compliquée par les règles de filtrage des routeurs pare-feu[34]. C'est pour cette raison que la version 4 du protocole NFS [PSB+00] utilise TCP, ainsi que pour son système de révocation des verrous d'accès exclusif aux fichiers.

Multidiffusion : IP **multicast**
Plusieurs tentatives d'implémenntation en multidiffusion ont été réalisées. C'est le cas de *JetFile* [GWP99] qui obtient des performances proche de celle d'un disque local. Mais le *multicast* n'est pas une solution intéressante pour un système de fichier. D'abord la mise en place du système, qui comprend configuration des routeurs, est très lourde. De plus la diffusion de données identiques simultanément est un cas rare, utilisée uniquement pour le démarrage de parcs de stations sans disques.

2.4.2 Compression des données et optimisation des transferts

Des algorithmes ont été développés pour réduire la quantité de données échangées sur le réseau. Le plus connu et le plus répandu est l'algorithme de Rsync [TM96] qui instaure un échange de sommes de contrôles calculées par les deux serveurs à synchroniser. L'idée a été reprise pour le système LBFS [MCM01], pour réduire la bande passante consommée en utilisant un système d'empreintes [Rab81][35].

Dans le système de fichier Coda, un mécanisme de *propagation des mises à jour basées sur les opérations* [LLS99] permet de ne transmettre que les modifications du client. Les opérations sont retournées au serveur qui les ré-éxécute, et en cas d'échec les données sont transmises par le biais des opérations d'entrée/sortie conventionnelles.

[34] Les RPC utilisent un port de communication bien connu (WKPN 111 rpcbind) pour contacter le *Portmapper*. Celui-ci a la charge de conserver les ports, sur lesquels les autres services RPC écoutent les clients, qui eux sont attribués dynamiquement et ne peuvent donc être énumérés dans les règles statiques d'un pare-feu.

[35] Le système suggéré par M. Rabin était à l'origine destiné à protéger les fichiers de modifications accidentelles ou malveillantes en calculant une somme de contrôle sur une zone de données, de façon aléatoire en utilisant une clef secrète. Il a déterminé qu'une clef de 63 bits fournissait une infime probabilité de valeurs identiques. Cet algorithme a été utilisé par [Man94] pour trouver des fichiers similaires dans un volumineux système de fichiers.

Communications

La conservation de la sémantique UNIX d'accès au fichier se paye par l'éxécution d'opérations atomiques, par l'émission de requêtes courtes et l'attente du retour du résultat des opérations. Si les systèmes de cache de données réduisent les temps de latence, leur taille doit augmenter avec les systèmes de fichiers pour rester efficaces. Or certaines opérations se succèdent systématiquement, par exemple l'accès à un répertoire suivi de la lecture de chaque entrée. Dans NFS version 4 [PSB+00], la concaténation des requêtes est implantée sous l'appelation *Compounds*. La réponse à un *Compound* est soit un seul message résultat, soit un ensemble de résultats, contenus dans un seul message. Si l'éxecution d'une commande contenue dans la requête aggrégée échoue, les commandes suivantes sont éliminées de la suite d'opération. Cette technique nécessite plus de mémoire sur le serveur.

L'éxécution de primitives systèmes à distance ralentit considérablement les performances d'une application. Par exemple, la recherche d'un motif dans les fichiers d'une arborescence requiert la traversée des répertoires et le transfert de chaque fichier afin d'en filtrer le contenu localement. Les *DynamicSets* [Ste97] permettent l'éxecution de selections à distance des noms de fichiers. Cette méthode permet d'effectuer de la lecture anticipée et de réduire les temps de latence. Les *CompositeCall* [BJPP+00] permettent d'éxécuter des opérations d'entrées-sorties en faisant migrer les opérations et non les données.

Le système de fichier Jade [RP93], à l'instar des *Name Spaces* du système Plan 9 [PPT+92], donne accès dans un même espace de nommage à des ressources hétérogènes en terme de protocole d'accès aux fichiers. Le système de fichiers est constitué des différentes ressources constituées en sous-arbres et forme la hiérarchie complète. La granularité de partage est au niveau des sous-arbres.

2.4.3 Granularité de la distribution et sémantique de partage

L'intégrité des données doit être conservée. les accès concurrents doivent être bloqués lorsque les données sont modifiées. Le verrouillage peut être réalisé au niveau du fichier, c'est le verrouillage à gros grain, par les applications ou par le système de fichier. Lorsque le système de fichier repose sur un système de volumes partagés, le verrouillage doit être réalisé au niveau des blocs, c'est le verrouillage à grain fin.

Une approche plus simple pour l'implantation d'un systèmes de fichiers distribués non centralisé est de séparer l'aspect stockage de la partie système de fichiers.

Le système de disques logiques partagés pour système de fichiers distribués, décrit dans [RAS96], permet de placer le problème de cohérence dans le cache de données au niveau du système de stockage par un mécanisme de vérrouillage au niveau des blocs réalisé par le système de fichier. Ce principe a été repris dans le couple Petal pour le serveur de disque réparti et Frangipani pour le système de fichier [LT96, TML97].

Le volume logique et le disque virtuel partagé peuvent être associés pour combiner plusieurs disques en un seul volume logique réparti. Cette solution a initié le projet Global File System (GFS) [And99].

Systèmes d'échange nœud à nœud

Le modèle idéal de système de stockage est réparti au maximum. Chaque client participe au stockage et à la diffusion des données, il n'existe pas de serveur dédié. La disponibilité est accrue par une réplication massive. L'espace de stockage est étendu et compense la forte redondance des informations. Les projets *FreeNet* [CSWH00] et *FreeHaven* [DKK+01] ont initié ce principe de stockage. La réplication des données du sytème est un premier atout contre la censure[36] et contre la destruction volontaire de données.

Ces systèmes d'échange de fichier ne supportent pas la sémantique d'un système de fichier traditionnel. La recherche des fichiers est réalisée par une recherches de chaîne par motifs et l'accès aux données est séquentiel. Pour garantir l'anonymat des participants, il existe dans *FreeNet* un mécanisme de routage propre au système. Les nœuds et les fichiers sont identifiés par des clés de hachage pour authentifier l'origine et verifier l'intégrité des données.

Le système *OceanStore* [KBC+00] utilise ces propriétés pour implanter un véritable système de fichiers. Les algorithmes de nommage des fichiers, de routage des échanges, de propagation des modifications et de sécurisation utilisés sont complexes et délicats à mettre au point.

Le système CFS, pour *Cooperative File system* [DKK+01], utilise toutes les propriétés des systèmes *Peer-to-peer* mais la granularité est plus fine car il répartit des blocs de données à la manière d'un disque réparti[37].

2.4.4 Fiabilité du stockage

Le projet Archipellago [JF99] appelé aussi IFS, pour Island-based File System, fournit un système d'accès par ilot (une machine) qui continue à fournir les services demandés même lorsque les connexions avec d'autres machines ne sont plus disponibles.

Le système de stockage parallèle Scotch [GSC+95] explore et étudie les évolutions du système RAID, que ce soit en terme de prix/performance, de réduction de la complexité de maintenance, de permettre aux applications séquentielles d'utiliser le stockage parallèle et même d'étendre ses avantages aux systèmes de calculs parallèles et distribués.

xFS [ADN+95] donne l'exemple d'un sytème de fichiers complètement réparti, sans serveur central. La réalisation de serveurs de fichiers complètement répartis est actuellement encore du domaine de la recherche [BDET00, DW01].

[36] Les réseaux d'échanges de fichier comme *Napster* ont été victimes de la centralisation d'une partie du système : les serveurs de recherche stockant la localisation des fichiers sur les nœuds d'échange.

[37] Cette approche est utilisée dans les système d'échange de fichiers Mojo-Nation ou Edonkey pour tronçonner les fichiers en blocs et permettre un accès direct aux blocs disponibles des fichiers.

2.4.5 Disponibilité

La disponibilité d'un système de fichiers est sa capacité à répondre sans interruption, ni délai, ni dégradation au moment où il est sollicité. La redondance des serveurs diminue la probalité d'arrêt du système en cas de panne d'une machine ou de la coupure du réseau. Les techniques de cache et de duplication des données sur des serveurs plus proches diminuent les temps de latence d'accès aux données. La duplication des données permet de répartir la charge d'accès entre les serveurs et augmente la disponibilité.

La tendance actuelle est de mettre à jour les serveurs de fichiers avec des processeurs plus puissants et des disques plus rapides. Comme le montre Erik Riedel dans [RG96], cette démarche est souvent injustifiée. La charge moyenne des serveurs de fichiers distribués, AFS dans son cas, ne représente que 3% de la charge des processeurs. Pourtant les ressources de calculs de ces serveurs sont saturées à certains moments, par exemple la première heure de la journée dans un entreprise. Le mécontentement des utilisateurs est le premier signe visible par l'administrateur d'une mauvaise disponibilité. La solution réside plus dans la gestion des caches et dans la réplication différée, par exemple répartir les données pendant les périodes de faible utilisation du réseau. Des solutions de type NAS, évitent des recopies inutiles de données entre disques et processeurs et déchargent les serveurs de fichiers. Les stockages de type NAS ont leur propres interfaces réseau.

Zebra [HO95] implémente un système RAID-4/5 avec journalisation sur une grappe de machines. Sur le même pricipe que les disque extractible à chaud, les serveurs peuvent integrer et disparaitre du cluster de stockage. L'augmentation des performances est de l'ordre de 20%-200% pour les petits fichiers et jusqu'à 400% pour les fichiers de grande taille qui sont fortement répartis.

Peter Braam [Bra99] fait un tour d'horizon des systèmes de fichiers utilisables pour un cluster du point de vue des protocoles utilisés. Dans un autre article [BN99], il montre que des performances proches de celles d'un système de fichiers local peuvent être obtenues, surtout en lecture-seule, avec Coda et Intermezzo.

2.4.6 Méthodes d'implantation

Librairies d'interposition

La programmation de systèmes de fichiers au niveau du noyau est délicate car elle nécessite une grande rigueur et de la patience, les erreurs se soldent le plus souvent par un redémarrage du système, et le débogage est difficile. C'est pourquoi l'utilisation des modules dans les noyaux qui les supportent ou dans les micro-noyaux, comme Mach [Tan94], facilite le développement en réduisant les temps de compilation.

Plus simple encore il est possible de créér une librairie de programmation (API) à utiliser lors de la compilation de l'application. Le projet SLIC [GPRA98] utilise une interface de programmation modulaire, c'est à dire que chaque ajout de fonctionnalité est réalisé par un module qui est enregistré dans l'interface fixe appelé *dispatcher*. Mais cette solution requiert la recompilation de chaque application utilisant cette API.

Une autre méthode consiste à utiliser un système d'interposition qui permet de détourner les appels systèmes[38] effectués par les applications pour accéder aux fichiers. Cette méthode comporte des avantages. Le temps de developpement s'en trouve réduit. De plus il existe déjà un système de developpement appelé *Bypass* qui permet de gérer les interpositions [TL00] et ce sur plusieurs niveaux d'appels de librairies[39]. Elle présente par contre des inconvénients. En effet, l'interposition n'est possible que pour les applications qui ont été compilées dynamiquement avec les librairies d'entrées/sortie du système[40]. L'interposition de librairies nécessite d'écrire de nombreuses fonctions d'interpositions s'étendant sur tous les appels systèmes.

2.5 Conclusion

Nous avons présenté différents systèmes de fichiers distribués. Le système de fichiers idéal supporte un grand nombre d'utilisateurs, il s'adapte au gré des évolutions et ne requiert que peu de maintenance ; il doit être disponible, extensible et fiable.

La répartition des données permet d'obtenir de meilleurs temps d'accès en distribuant les accès sur plusieurs machines ; la disponibilité est augmentée. Le système de fichier doit supporter un grand nombre de fichiers et doit supporter l'augmentation ou la réduction de l'espace de stockage ; il doit être extensible. Il reste fiable en présence de pannes, que ce soit d'un serveur, d'une disque ou du réseau.

[38] Fonctions liées à la hierarchie du système de fichiers (représentée par les répertoires), aux attributs de fichiers (droits et dates d'accès) et à la lecture et à l'écriture des données même.

[39] un appels a la fonction `read` de la librairie standard peut faire appels a une autre fonction de lecture sur un peripherique `prefixread` (utilisé généralement par les pilotes de périphérique) et entre les appels à tous deux, une fonctions d'interposition peut être placée.

[40] C'est le cas des commandes Unix de la librairie *C* standard sous le système Solaris ou de GNU/Linux. Un système comme FreeBSD, par exemple, ne permet pas d'utiliser ces agents d'interposition pour les commandes de base car les commandes sont compilées statiquement par défaut pour optimiser le système.

Chapitre 3

Réalisation de notre serveur de disques virtuels répartis

Nous décrivons dans cette partie l'architecture de notre système de disques virtuels distribués. Ce système se distingue par sa haute disponibilité, son extensibilité et l'utilisation de systèmes de stockage conventionnels. Il consiste en un ensemble de serveurs connectés entre eux qui gèrent de façon coopérative leurs disques physiques.

Notre projet ressemble à *Petal*, présenté sommairement dans [Lee95] et [LT96] et auquel nous n'avons pu avoir accès. Petal a été conçu pour être intégré dans des baies de disques de type NAS, des contrôleurs de disques interconnectés par un réseau dédié très haut débit.

Nous pensons que notre système est utilisable dans le cadre d'un réseau local de stations de travail. Nous détaillons l'architecture logicielle de notre système.

3.1 Architecture matérielle du serveur de disques virtuels

Nous présentons ici l'environnement pour lequel est destiné notre serveur de disques virtuels.

3.1.1 Une grappe de serveurs de disques

Le serveur de stockage est constitué d'un groupe de serveurs de disques. L'interface de communication présente un périphérique de stockage de bas niveau de type *bloc*[1] comme le montre la figure 3.1. Sur ces périphériques de stockage, les clients du serveur de disques virtuels peuvent implémenter un système de fichiers utilisant les spécifités du système. L'interface de communication entre les clients et le serveur

[1]Sous Unix il existe deux types de représentation des périphériques. La première de type *bloc* concerne les périphériques à accès direct, la seconde de type *caractère* concerne les périphériques à accès séquentiel comme les lecteurs de bandes, les ports séries ou les périphériques dits de console.

est réalisée par TCP/IP comme dans le projet Network Block Device [BLA00].

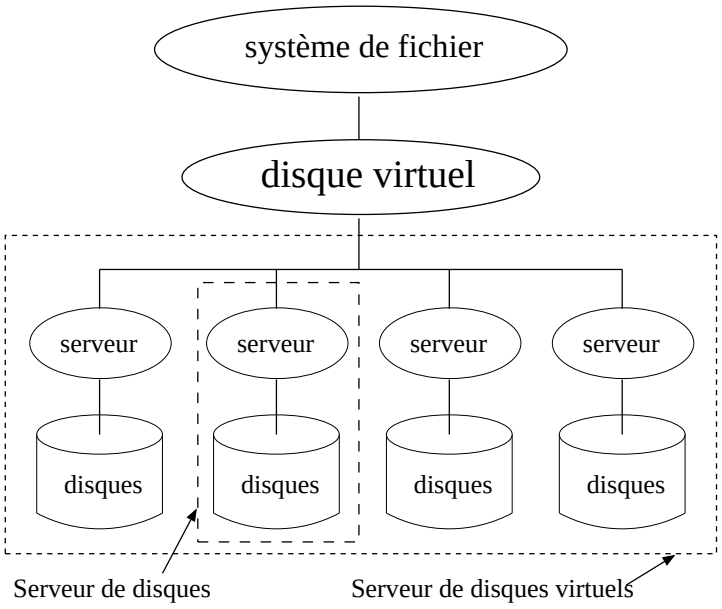

FIG. 3.1 – Architecture du serveur de stockage

3.1.2 Réseau d'interconnexions

Les communications entre les différentes entités nécessitent d'apporter une attention particulière au choix de la topologie du réseau d'interconnexions. Il existe deux types de communications :

1. Entre un client et le serveur de disques virtuels : principalement lors des accès au disque virtuel ou bien lors de la reconfiguration du disque virtuel.
2. Entre les serveurs de disques physiques : lors de la mise à jour des tables de configuration par consensus et de façon intensive lors de la redistribution des blocs entre les disques physiques après reconfiguration du disque virtuel.

Nous pouvons donc distinguer les communications internes entre les serveurs de disques et les communications entre les clients et le disque virtuel.

Le serveur se présente comme un NAS et les échanges internes de données sont généralement privilégiés par un réseau haut-débit dédié[2].

L'interconnexion entre les serveurs composant le disque virtuel et ses clients est réalisée par des commutateurs Ethernet (*switches*). Cette segmentation du réseau permet de restreindre le traffic de trames aux machines concernées. La commutation évite la saturation du réseau Ethernet[3].

[2]Le prototype Petal a été testé sur un réseau ATM à 155 Mbit/s.

[3]Lorsque le traffic est trop important sur un réseau Ethernet, les collisions de trames augmentent jusqu'à diminuer le débit maximal. Ce phénomène est dû à la négociation du médium par les

Nous avons décider d'évaluer notre prototype sur un réseau Ethernet 100Mbits/s car nous voulons étudier la possibilité d'utiliser ce type de serveur sur l'espace disque inutilisé de stations de travail.

3.2 Architecture logicielle d'un serveur de stockage

Notre serveur de disque virtuel est composé de cinq modules fonctionnels interdépendants. Cette structure permet d'isoler ses fonctionnalités et en simplifie l'implémentation. Le choix précis des types de données est primordial pour permettre l'implémentation sur des architectures dissemblables, notamment pour le passage de message qui se doit d'être indépendant de l'architecture des machines.

3.2.1 Types de données

Notre système de disque virtuel possède un espace d'adressage sur 64 bits, ce qui représente 16 Exa-octets[4]. Cette dimension est justifiée par la taille des serveurs de stockage actuels et il est raisonnable d'estimer que l'Exa-octet sera courant en 2015. D'autre part cet espace d'adressage représente une facilité d'organisation pour notre système où la disposition des données peut être clairsemée.

3.2.2 Tables d'adressage de blocs

Nous donnons ici la structure de la table d'adressage qui permet d'établir la correspondance entre un bloc sur disque virtuel et le bloc réel sur un disque physique de l'un des serveurs de disques. Cette table est divisée en trois parties dont certaines sont dupliquées sur chacun des serveurs de disques, comme représenté sur la figure 3.2 par des pointillés.

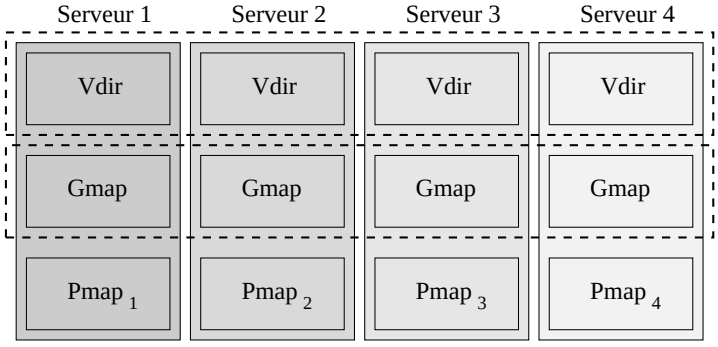

FIG. 3.2 – Table d'adressage des blocs

interfaces par un dispositif d'attente aléatoire dont la durée augmente exponentiellement avec le nombre de collisions.

[4]16 Exa-octets = 16 milliards de Giga-octets.

Table des disques virtuels

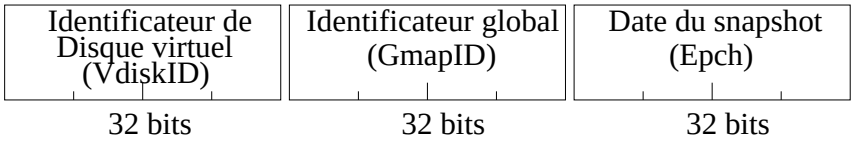

FIG. 3.3 – Une entrée de la table des disques virtuels (Vdir)

Le répertoire virtuel (VDIR) contient une table de correspondance entre les identifiants des disques virtuels et un identifiant dans la table globale. Cette indirection supplémentaire permet de gérer le mécanisme d'image de sauvegarde. L'identificateur VdirID d'une image de sauvegarde est distinct du disque virtuel original. Il est associé à une date qui sera jointe à l'idenficateur GmapID lors de l'accès au disque virtuel. Le VDIR est répliqué sur chacun des serveurs de disque du disque virtuel.

Table d'adressage globale

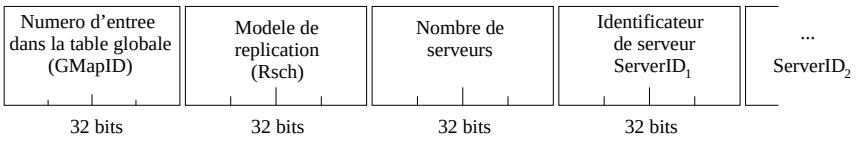

FIG. 3.4 – Une entrée de la table d'adressage global (Gmap)

La table d'adressage global (GMAP) est une table immuable, créée au moment de la configuration du disque virtuel. Cette table contient l'identificateur des serveurs sur lesquels est réparti le disque virtuel, ServerID_n sur la figure 3.4, ainsi que le modèle de redondance destiné à protéger les données.

La table est répliquée sur chacun des serveurs de disque. Une entrée dans la table GMAP ne peut être modifiée. Pour changer le modèle de réplication où modifier le groupe de serveurs de disques, une autre entrée doit être créée, les blocs sont déplacés entre les serveurs et l'ancienne entrée est détruite.

Table d'adressage physique

Chaque serveur de disque contient une table physique PMAP. Les disques physiques sont découpés en segments de 64Kilo-octets. Chacun de ces segments correspond à une entrée dans la PMAP, figure 3.5. Chaque serveur de disque possède un table physique privée.

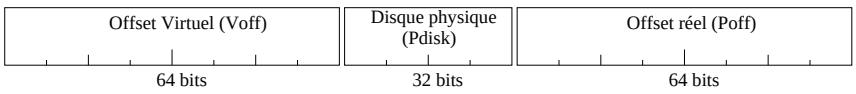

FIG. 3.5 – Une entrée de la table d'adressage physique (Pmap)

3.2.3 Modules de gestion

Notre serveur de disque virtuel est composé de cinq modules interdépendants. L'organisation générale est donnée sur la figure 3.6.

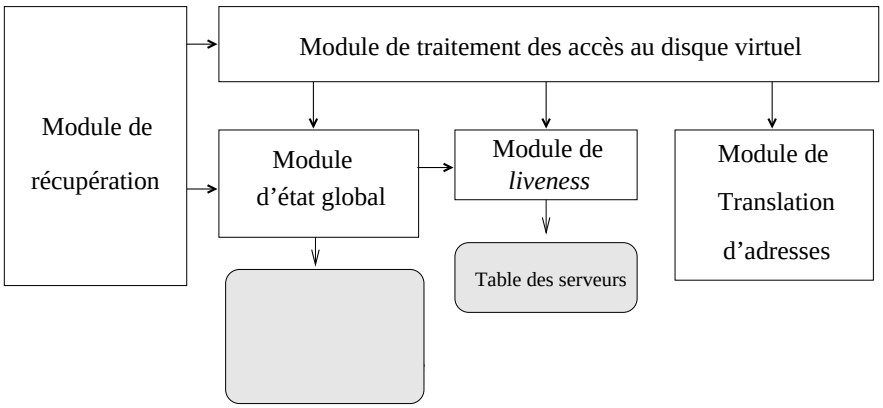

FIG. 3.6 – Organigramme des modules du serveur de disques virtuels

Le module d'accès reçoit les requêtes des clients du serveur de disques virtuels. Il peut s'agir de requêtes d'accès aux données ou bien de requêtes concernant l'administration des disques virtuels.

Le module de *Liveness* est chargé de vérifier continuellement que chaque serveur est présent et met à jour la table des serveurs.

Cette table est utilisée par le module d'état global pour répliquer et synchroniser la table GMAP et la table VDIR.

Le module de translation d'adresses traduit une adresse logique du disque virtuel en adresse réelle sur le disque physique de l'un des serveurs composant le disque virtuel.

Un schéma de réplication permet de stocker un bloc sur deux serveurs distincts. La panne d'un serveur ne bloque pas l'écriture du bloc sur l'autre serveur. l'écriture est retardée jusqu'à la réintégration du serveur défaillant. Le module de récupération est chargé de synchroniser les blocs répliqués.

Translation de blocs virtuels en blocs réels

Notre système de disques répartis utilise un système de pagination de l'espace disque physique. Ce principe, initialement utilisé pour la gestion de la mémoire [CLBHL92], permet d'allouer et de libérer des zones appelées *segments*. Cette technique possède deux avantages :

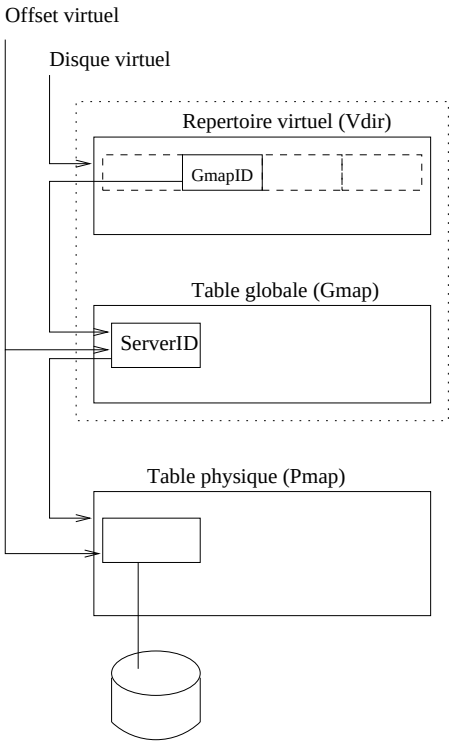

FIG. 3.7 – Mécanisme de translation d'adresses

- Faciliter le déplacement des segments lors d'une reconfiguration du disque virtuel.
- Permettre au système de fichiers de tirer parti de l'adressage sur 64 bits pour définir des zones pour les tables d'allocation, et les différentes zones de stockage en fonction de la taille des fichiers[5].

Le concept de disque logique permet de réaliser une séparation nette entre la vue que les clients ont du système de stockage et les disques physiques qui le constituent. Il est utilisé pour l'agrégation de disques sur des systèmes RAID logiciels, comme par exemple le système de disques distribués GFS [And99].

Nous décrivons dans cette section le mécanisme de translation des adresses virtuelles de notre système de stockage en adresses réelles sur les disques physiques

[5]Lorsque la taille des blocs de stockage référencés dans la table d'allocation est fixe, les fichiers de grande taille utilisent plus d'entrées dans la table. En augmentant la taille de ces blocs, l'espace de stockage est perdu entre la fin du fichier et la fin du dernier bloc alloué. Une table d'allocation avec tailles de blocs variables, permet de réduire l'espace perdu par les petits fichiers et de réduire les temps d'accès sur les grands fichiers.

présents sur les serveurs.

Le problème est de transformer une adresse du type *(identifiant de disque virtuel, position sur le disque virtuel)* en adresse de type *(numéro de serveur, identifiant de disque réel, position sur le disque réel)*. Les avantages de ce mécanisme de virtualisation des adresses sont décrits par [dJKH93].

Répartition de l'accès aux disques

Le problème de la répartition des accès aux données est crucial dans un système distribué. Un équilibre doit être trouvé entre le modèle à serveur unique et le modèle totalement distribué. Le premier forme un goulot d'étranglement pour l'accès aux données. Le second demande de nombreuses communications internes pour la synchronisation des serveurs.

Dans notre serveur de disques, les accès sont équilibrés par une technique de redirection. Celle-ci est liée à l'étape de translation des adresses. Lorsque le serveur ne contient pas le bloc logique demandé, il retourne un message de redirection. De plus, certains schémas de réplication rendent les données accessibles sur plusieurs serveurs de disques. Dans ce cas le client utilise le serveur le plus disponible, c'est à dire celui dont la file des requêtes en attente est la moins remplie.

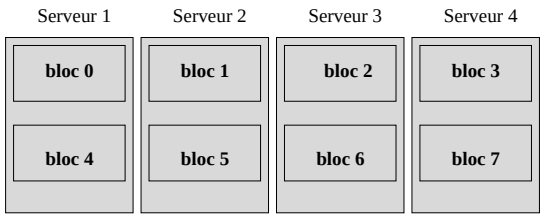

FIG. 3.8 – Disposition des blocs pour la fragmentation linéaire

La Fragmentation sans redondance

Cette méthode appelée *stripping* permet de répartir les accès sur l'ensemble des serveurs de disques. La disposition des blocs est décrite sur la figure 3.8. Elle n'offre pas de redondance et donc l'arrêt d'un seul serveur de disque provoque l'arrêt du serveur de disque virtuel. Lors de la translation, la GMAP calcule le numéro de serveur d'un segment de données d'après la formule suivante :

$$S(b) = b \ mod \ N$$

où S est le serveur contenant le bloc b et N est le nombre de serveur composant le disque virtuel.

Dégroupement chaîné

La méthode de *chained declustering* [HD90] est la deuxième méthode que nous

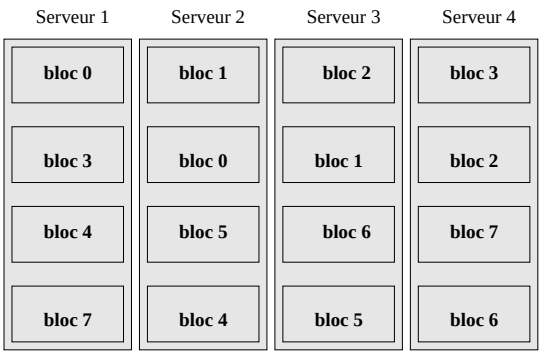

FIG. 3.9 – Disposition des blocs pour le dégroupement chaîné

avons implémentée dans notre serveur de disques virtuels. Elle offre une redondance identique au RAID 0, appelée mirroring. En contrepartie, la capacité de stockage représente la moitié de l'espace disque disponible comme le montre la figure 3.9. Le serveur S_1 contenant un bloc b et le serveur S_2 contenant le réplicat du bloc sont déterminés via la GMAP par la formule suivante :

$$S_1(b) = b \ mod \ N$$

et

$$S_2(b) = (b+1) \ mod \ N$$

L'avantage apporté par cette méthode est la robustesse. Le serveur fonctionne même avec $N/2$ serveurs défaillants non contigus dans le groupe. Par exemple si le serveur 2 ne sert plus les requêtes, le bloc 0 est toujours accessible sur le serveur 1 et le bloc 1 sur le serveur 3.

De plus la répartition de charge reste uniforme même en cas de défaillance d'un serveur. La figure 3.10 compare l'équilibrage de la charge sur quatre serveurs entre le RAID-1[6] et le *chained declustering*, lorsque le serveur 2 est en panne. En RAID=1, le serveur 1 doit compenser la défaillance du serveur 2 et fournit 200% de sa charge nominale. Le dégroupement chaîné ré-équilibre la charge sur l'ensemble des serveurs. Le serveur 1 et le serveur 3 fournissent 100% de la charge pour les blocs 0 et 1, mais délèguent 16% de la charge des blocs 2 et 3 au serveur 4. Chaque serveur fournit bien 133% (4/3) de sa charge nominale.

Support pour la sauvegarde

La sauvegarde d'un volume de stockage requiert du temps car elle se fait généralement sur bande dont le temps d'accès est lent. Pour que le contenu soit stable, il est préférable de supprimer l'accès au volume de stockage par les utilisateurs pendant la durée de la sauvegarde. Cette condition s'impose du point de vue des fichiers en cours d'utilisation et donc partiellement écrits. La stabilité de l'arborescence est

[6]Le RAID-1 est appelé mirroring & duplexing. Les serveurs sont couplés par paires en "mirroir" et les blocs sont répartis sur ces "duplex".

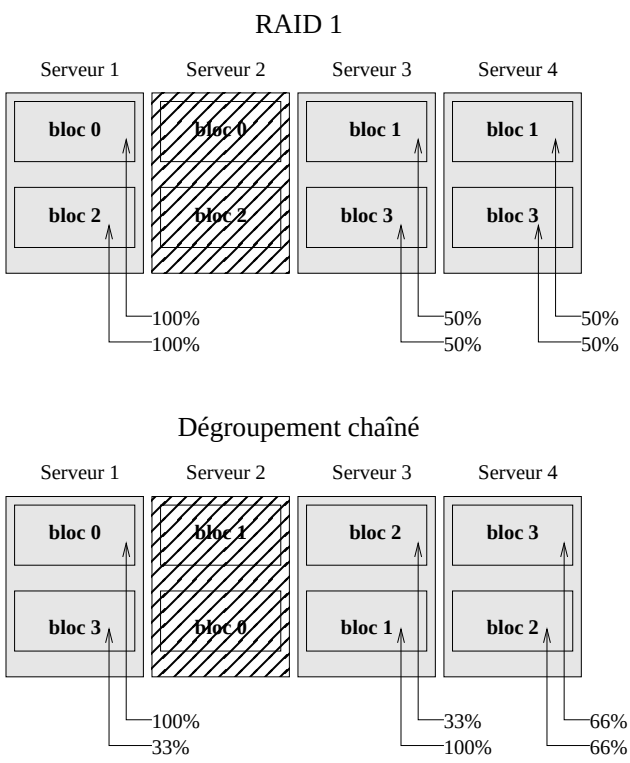

FIG. 3.10 – Comparaison de la répartition de charge entre RAID 1 et *chained declustering*

également requise. La sauvegarde est généralement réalisé au niveau du système de fichier en parcourant récursissivement l'arborescence des répertoires.

Afin d'obtenir un état stable du système de stockage sans interrompre les utilisateurs, il est nécessaire de créer une nouvelle image du disque virtuel. Pour cela il existe des techniques basées sur le *copy-on-write* qui duplique les blocs lors d'une tentative d'écriture. Cette technique[7] est principalement utilisée dans les SAN et pour des périphériques à écriture unique [Qui91]. Après la création d'une image de sauvegarde, il existe deux entrées dans le répertoire virtuel de notre système de stockage :

1. Une nouvelle entrée, qui représente l'image du disque destinée à la sauvegarde, référence tous les blocs du disque virtuel écrits avant la date[8] de création de l'image.
2. L'entrée originale du disque qui référence les blocs modifiés après la date de la (des) sauvegarde(s). Lors d'un accès au disque, le mécanisme de translation utilise cette entrée VDIR, et si les blocs n'y sont pas référencés, le bloc sera pris dans l'entrée référençant ce bloc dans la sauvegarde la plus récente. À chaque

[7] L'application de cette technique aux systèmes de stockage est appelée le *snapshoting*.

[8] La date est ici la combinaison de la date et de l'heure representée par le nombre de secondes écoulées depuis le 1^{er} janvier 1970 à minuit, et appelée *epoch*. Le type utilisé par la norme POSIX est un entier non signé de 32 bits (`time_t`) . Ce type sera obsolète en 2038. Un entier relatif de 64 bits permettrait de définir une date de +/− 292 milliards d'années.

demande de modification d'un bloc uniquement référencé dans la sauvegarde, le bloc est dupliqué sur un bloc libre et référencé dans l'entrée initiale du disque virtuel.

3.3 Évaluation du serveur de disque

Nous présentons l'etat actuel du dévelopement de notre serveur de disques virtuels et les résultats obtenus par le prototype. Nous annonçons également les prochaines étapes de notre réalisation.

3.3.1 Les fonctionnalités réalisées

Notre prototype de serveur est constitué dans l'immédiat des trois premiers modules de l'architecture complète du serveur de disques virtuels : le module d'accès, le module de translation et le module de vérification de la vivacité. Ces trois modules ont permis de valider notre hypothèse de possibilité d'implantation dans un réseau local non dédié.

La répartition des accès sur plusieurs serveurs masque parfaitement les temps de latences des disques. Lors d'accès intensifs au disque virtuel constitué d'au moins deux serveurs, une station de travail cliente n'est limitée que par la bande passante maximum du réseau local Fast-Ethernet à 12 Méga-octets par seconde.

L'accès parallèle au disque virtuel composé de 4 serveurs de disques par 8 machines clientes donnent des résultats a peine inférieurs avec 6 Méga-octets par secondes. Ces résultats sont bien meilleurs qu'un serveur NFS qui, malgré les optimisations apportées lors des 15 dernières années, plafonne à 2 Mo/s par clients dans la limite de 5 clients.

Le vérificateur de vivacité prévu au départ pour les vérifications entre serveurs, à été étendu aux clients des serveurs de disques. Il permet de gérer la déconnexion et la reconnexion des machines. Notre système de communication entre clients et serveurs est de type connectée. Nous bénéficions de la fiabilité de ce type de protocole.

L'intégration du schéma de réplication *dégroupage-chaîné* couplé au vérificateur de vivacité ont permis de constater la fiabilité du système. Nous avons provoqué l'arrêt d'un serveur de disques et les clients reportent leurs requêtes, sans dégradation notable, sur les serveurs restants lorsque les serveurs leur notifient le changement.

3.3.2 Premiers résultats

Nous avons mesuré les débits en lecture de notre serveur de disques virtuels sur des stations Digital ALpha 233 Mhz equipées de disques SCSI-2 (20 Mo/s). Les machines sont reliées entre elles par un réseau Fast-Ethernet (100Mbits/s) commuté par des switchs Cisco 2350XL.

Les tableaux 3.11 et 3.12 contiennent les informations suivantes : le nombre de serveurs par disque virtuel et le nombre de clients accédant simultanément à ce disque. La colonne *temps* correspond au temps réel (temps d'éxécution) pour transférer le volume de données indiqué dans la colonne *Volume*. Le temps comprend le temps de l'émission de la requête par le client et l'accès au disque et le temps de transfert de la réponse du serveur. La colonne *débit client* indique le débit perçu par le client. La colonne *débit Vdisk* indique le volume total de données servi par l'ensemble des serveurs constituant le disque virtuel.

Analyse de la répartition avec clients et serveurs distincts

Clients	Serveurs	Temps (s)	Débit client (Mo/s)	Volume servi (Ko)	Débit Vdisk (Mo/s)
1	1	0,35	11,11	4000	11,09
2	1	0,68	5,70	8000	11,37
4	1	1,38	2,83	16000	11,29
1	2	0,35	11,18	4000	11,14
2	2	0,36	10,97	8000	21.89
4	2	0,71	5,46	16000	21,80
1	4	0.35	11,18	4000	11,15
4	4	0.38	9,78	16000	39,04
6	4	0,50	7,76	24000	45.33

FIG. 3.11 – Débit aggrégé du disque virtuel. Les clients et les serveurs sont éxécutés sur des machines distinctes.

La première partie du tableau 3.11 montre les limites du serveur unique, équivalent à un serveur centralisé. Le débit maximal servi par le disque est limité par la bande passante du réseau Fast-Ethernet, qui représente environ 11,4 Mo/s. Chaque client supplémentaire reduit le débit servi par le disque virtuel.

Dans la deuxième partie du tableau, deux serveurs composent le disque virtuel. Deux clients utilisent la totalité de la bande passante cumulée (21,9 Mo/s) des deux serveurs. Le débit maximal pour un seul client est légérement supérieure au cas précédent du serveur unique. La charge de calcul qui est engendrée par les transferts entre le disque dur et l'interface réseau du serveur, est répartie sur deux machines.

Dans la dernière partie du tableau 3.11, le débit total servi par le disque virtuel est supérieur à celui d'un disque dur Ultra Wide SCSI[9]. Avec un nombre de clients supérieur à celui des serveurs, nous remarquons que la limite est encore imposée par la bande passante cumulées des quatres serveurs avec 45,3 Mo/s.

Ces tableaux ne témoignent pas de la charge de calcul engendrée par la lecture et la transmission des données. Les serveurs de disques et les clients du disques virtuel sont distincts. Cette première configuration correspond à un SAN sur un réseau dédié aux serveurs. Le réseau 100 Mbits/s permet d'atteindre des taux de transfert comparables aux disques haut de gamme, avec des disques bon marchés.

Analyse des performances avec clients et serveurs confondus

Le tableau 3.12 présente un test de lecture intensive, mais la configuration diffère de celle utilisée précédemment. Les serveurs et les clients cohabitent sur les même machines. Cette configuration induit deux changements notables :

1. Sur chaque machine, le client et le serveur se disputent la bande passante entre la machine et le commutateur,

[9]Les disques durs Ultra-Wide SCSI, ou SCSI-3, ont un débit crête de 40Mo/s. Ce débit ne peut être atteint que lors de la lecture de pistes complètes et contigues.

Clients	Serveurs	Temps (s)	Débit client (Mo/s)	Volume servi (Ko)	Débit Vdisk (Mo/s)
1	1	1,21	16,12	20000	16,12
2	2	1,86	10,51	40000	20,47
4	4	2.14	9,16	80000	36,55
8	8	2,38	8,21	160000	65,56

FIG. 3.12 – Débit total du disque virtuel. Sur chaque machine est éxécutés un client et un serveur.

2. Les échanges entre un client et un serveur placés la même machine, ne transitent pas par le réseau.

Les débits sont un peu inférieurs à ceux présentés dans le tableau 3.11. Dans le cas du serveur et du client uniques, il n'y a aucune communication sur le réseau. Le débit de 16,12 Mo/s représente le débit maximal du serveur : la limite supérieure atteinte par le disque et l'interface réseau. Dans le cas de 2 ou 4 serveurs, le débit total servi par le disque virtuel représente 93% du débit maximal indiqué dans le premier tableau.

Ces résultats montrent l'interêt de la répartition des données. Ils indiquent également que l'utilisation de serveurs non dédiés offre une solution intérressante.

3.3.3 Les modules à implémenter

La cohérence de l'état du disque virtuel sur chaque serveur de disque est nécessaire pour être utilisée par un système de fichiers. Elle est assurée par les modules d'état global et de restauration.

La réalisation du module d'état global est dépendante de l'intégration d'un mécanisme de distribution de l'état du serveur de disque. Ce mécanisme de distribution doit être fiable, supporter les pannes des serveurs, les coupures des interconnexions, c'est à dire du réseau et garantir l'intégrité des données échangées. Nous avons choisi l'algorithme Paxos.

Chapitre 4

Notre implémentation de l'algorithme Paxos

> *Un système distribué est un système dans lequel la panne d'un ordinateur dont vous ignoriez totalement l'existence peut rendre le votre inutilisable.* L. Lamport [Lam87]

Nous avons montré dans la partie précédente que la disponibilité du système disque est gérée par un module opérationnel et un module d'état global. Le premier réalise une vérification permanente de l'état de chacun des autres serveurs. Le second assure la consistance de l'information globale sur les disques virtuels. Les opérations de gestion du système, tels que la création ou la suppression des disques virtuels, l'ajout où le retrait de serveurs, sont tolérants aux défaillances.

L'algorithme Paxos conçu par Leslie Lamport [Lam98][1], est un algorithme de consensus tolérant aux pannes. Il peut s'agir soit une panne de serveur qui cesse de fonctionner, soit la perte, la duplication ou le réordonancement de messages. Paxos garantit un fonctionnement correct tant qu'une majorité de serveurs peuvent communiquer entre eux pendant le temps nécessaire à l'établissement du consensus. L'application de Paxos à la réplication de données est illustrée dans [GL00].

4.1 Problématique du consensus

La synchronisation de données dans un système distribué consiste à établir un consensus sur ces données entre les membres du système. Un algorithme de consensus distribué permet à chacun des processus de décider d'une valeur de sortie appartenant au type des valeurs présentées en entrées et pour lesquelles trois buts sont

[1] Encouragé par sa réussite à vulgariser le problème du consensus avec *Les généraux byzantins*[LLP82], Leslie Lamport présenta son algorithme Paxos de façon humoristique en le nommant le *Le parlement à temps partiel* et en le présentant comme une découverte archéologique dans une ancienne île grecque. Il poussa la plaisanterie jusqu'à utiliser les noms d'informaticiens du domaine et à les traduire approximativement (Λινσεϊ pour P. Lindsai, Γραι pour J. Gray, etc.). Cependant, lorsqu'il proposa son algorithme en 1990, la commission de lecture des "Transactions on Computer System" de l'ACM manqua d'humour et ce n'est qu'en mai 1998 que Lamport vit son article enfin accepté.

atteints : *accord*, *validité* et *terminaison*.

La condition sur l'accord implique qu'il ne peut y avoir que deux valeurs de sortie. L'appartenance de la valeur de sortie à l'ensemble des valeurs initiales est requise par la condition de validité. Et enfin la terminaison qualifie un algorithme dans lequel chaque processus non-défectueux émet une valeur.

4.1.1 Les éléments du système distribué

Notre système est composé de machines et d'un réseau local de type Ethernet qui les relie. Les machines sont des ordinateurs non dédiés, c'est à dire que les utilisateurs sont susceptibles de provoquer des blocages en saturant le système ou en redémarrant inopinément. Ces ordinateurs sont munis de disques susceptibles de tomber en panne mais dont les données ne peuvent être détruites volontairement. Les communications entre ces machines sont de type UDP/IP, des échanges de segments de données sans garantie d'acheminement, d'unicité ni de séquencement.

Ces conditions sont exactement celles pour lesquelles l'algorithme Paxos a été conçu. Sur chaque machine, nous démarrons un processus Paxos qui participe au consensus.

4.1.2 Principe de fonctionnement

L'algorithme Paxos est exprimable sous la forme d'un automate représenté sur la figure 4.1. Celui-ci peut être décomposé sous la forme de trois sous-automates :

1. *l'assesseur* appelé LEADER,
2. *l'électeur* appelé AGENT,
3. *l'annonceur* appelé SUCCESS.

Chaque processus Paxos, exécuté individuellement, tente d'atteindre un état final où une valeur proposée est décidée par une majorité de processus et connue par tous à terme. Dans chaque processus, le sous-automate AGENT participe au consensus. Le consensus est dirigé par un processus Paxos élu maître. Dans ce processus le sous-automate détecteur LEADER est actif. Finalement le sous-automate SUCCESS est activé dans chaque processus pour annoncer la valeur décidée.

Paxos garantit, quel que soit le nombre de maîtres élus, qu'une seule décision est valide. Afin que l'algorithme atteigne un état terminal en l'absence de panne[2], il doit être augmenté d'un automate de *démarrage*. Nous appelons cet automate STARTER. Il est chargé d'initier une nouvelle ronde lorsque la précédente ne s'est pas achevée dans l'intervalle de temps imparti.

Les processus maîtres sont élus par un module *électeur de maître* ou LEADER-ELECTOR. Nous remarquons que si celui-ci ne garantit pas l'existence d'un unique *maître*, il ne compromet en aucun cas la validité de la valeur. Mais l'état terminal de Paxos ne peut être atteint qu'en la présence d'un unique processus *maître*.

[2]Paxos garantit la prise de décision lorsque le système est stable pendant le minimum requis pour effectuer les trois phases du protocole.

Le consensus se déroule en trois phases successives.

1. Une phase de *collecte* des valeurs décidées aux rondes précédentes.
2. Une phase de *proposition* de la valeur.
3. Une dernière phase dite d'*annonce* de la valeur décidée.

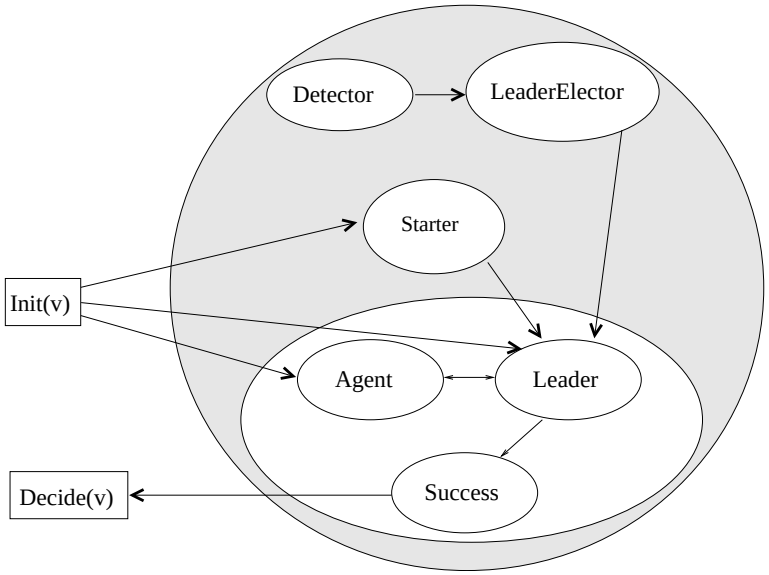

FIG. 4.1 – Processus Paxos simple

4.1.3 Contraintes

Ordre sur les rondes

Dans un contexte réparti, il n'existe pas d'horloge globale commune aux machines. Pour la construire il faudrait utiliser leur horloge physique, qu'il faudrait synchroniser. Mais il n'existe pas de garantie que deux horloges donnent à tout instant la même date, principalement en raison du délai de transit arbitraire des messages de synchronisation. Il est plus simple de réaliser un système de datation logique [Lam78].

Le choix d'une valeur est fait par tournées d'élection ou *rondes* qui doivent être distinctes pour pouvoir ordonner les événements dans le temps. Plusieurs rondes peuvent être menées concurremment - des rondes sont susceptibles d'être commencées par plusieurs processus concurrents, et doivent être distinguables. De plus le délai de transmission des messages, s'il est fini, est arbitraire.

Pour cela chaque numéro de ronde R est composé d'un couple de valeurs (r, p) composé d'un entier non-négatif r et d'un numéro de processus p. Ainsi il est possible d'établir un ordre total sur les événements dans un système concurrent. L'ordre

total est défini sur les éléments R de $\mathcal{R}$, l'ensemble des numéros de rondes R, par $(x, i) < (y, j) \equiv (x < y) \vee ((x = y) \wedge (i < j))$.

4.2 Mise en œuvre de Paxos

Nous avons décomposé et implanté l'automate de Paxos sous la forme de threads concurrents qui communiquent entre eux par passage de message UDP. Nous avons repris le modèle proposé par Roberto De Prisco [PLL97].

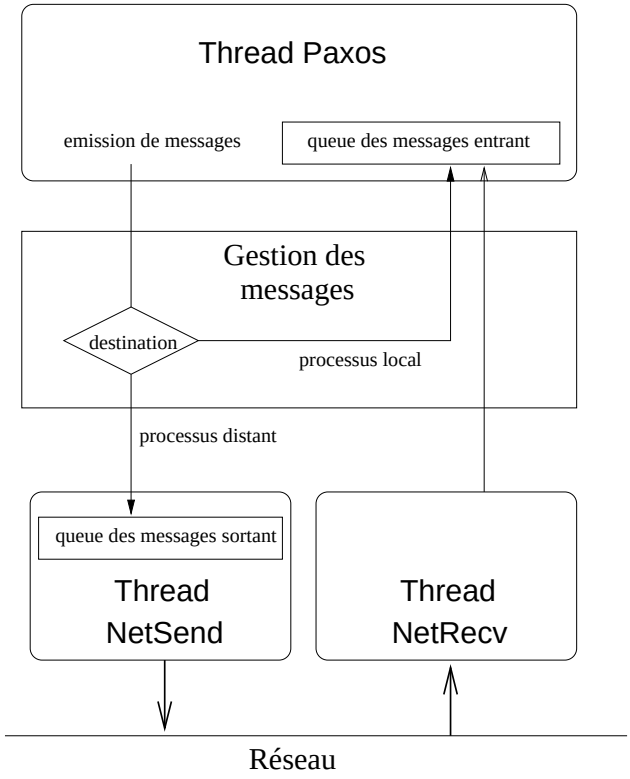

FIG. 4.2 – Les mécanismes de passage de messages au sein d'un processus Paxos. Le thread Paxos est générique et represente l'un des six threads Paxos : bpAgent, bpLeader, bpSuccess, bpDetector, bpLeaderElector ou bpStarter

Le cœur de l'automate est composé des threads bpAgent, bpLeader, bpSuccess, bpDetector, bpLeaderElector et bpStarter. Il correspondent aux automates de la figure 4.1.

Les échanges de messages sont de deux types : Les échanges entre les processus Paxos qui transitent par le réseau et les messages entre les threads d'un même processus. Les échanges de messages entre ces threads sont réalisés via des files de messages. Lorsque l'identificateur de destination est local, le message est directement placé dans la queue de réception du thread concerné. Les échanges de messages entre les processus sont traités par deux threads supplémentaires d'émission NetSend et de réception NetRecv. La gestion des messages est représentée sur la figure 4.2.

Le thread de réception identifie le type de message et le place directement dans la queue de réception du thread Paxos concerné. Chaque type de message correspond à

un échange particulier entre deux threads de Paxos. Par exemple le message de type *Collect* est diffusé par thread BPLEADER vers les threads BPAGENT, appartenant au processus local ou à un processus Paxos distant.

4.2.1 Le détecteur de défaillance et l'électeur de maître

Le détecteur d'échec et l'électeur de maître permettent de mettre en place l'algorithme d'élection du processus maître Paxos. Ils sont implantés avec des contraintes relachées. Il ne fournissent des informations valides que lorsque le système est stable. Ceci est parfaitement suffisant pour implantation de Paxos.

Le détecteur DETECTOR

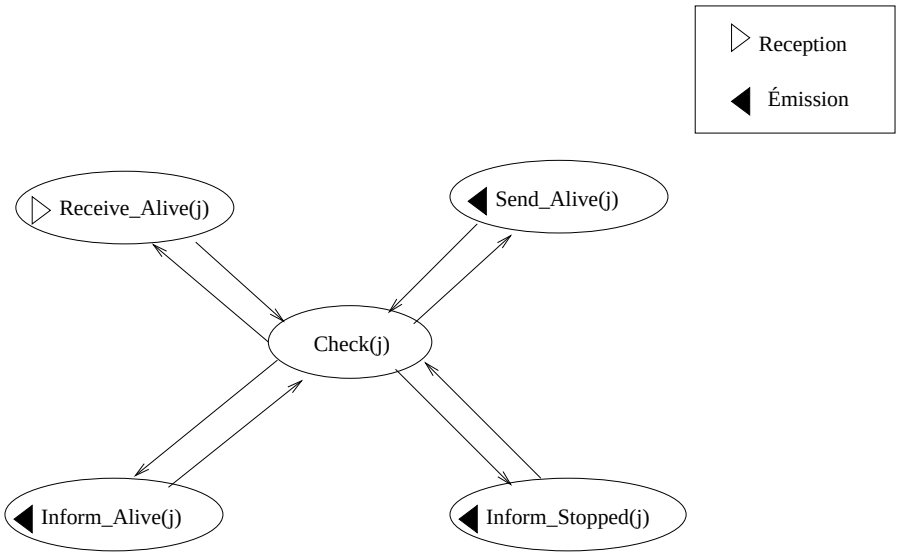

FIG. 4.3 – Le sous-automate détecteur pour le processus i

Le fonctionnement du sous-automate DETECTOR est basé sur l'envoi constant de messages *Alive* aux autres processus — pour indiquer que son fonctionnement est correct — et sur la vérification que ces messages sont effectivement reçus par ceux-ci. Un message est émis à chaque intervalle de temps fixe z ; Autrement dit un message de temps est envoyé au temps t alors le suivant doit être envoyé au temps $t + z$. La vérification de l'arrivée des messages *Alive* distants est faite pour tous les intervalles de temps c. Le détecteur informe le sous-automate LEADERELECTOR par des messages de type *InformStopped(j)* et *InformAlive(j)*.

Le déroulement de deux exécutions est montré sur la figure 4.4. La constante l est le temps maximum d'exécution d'une action dans le sous-automate et d est le temps d'acheminement d'un message entre deux processus Paxos. l n'a pas été représenté sur la figure car il est inférieur à d dans un rapport de cent[3].z et c sont

[3]Le temps d'une action au sein du détecteur est de l'ordre de quelque dizaines de microsecondes

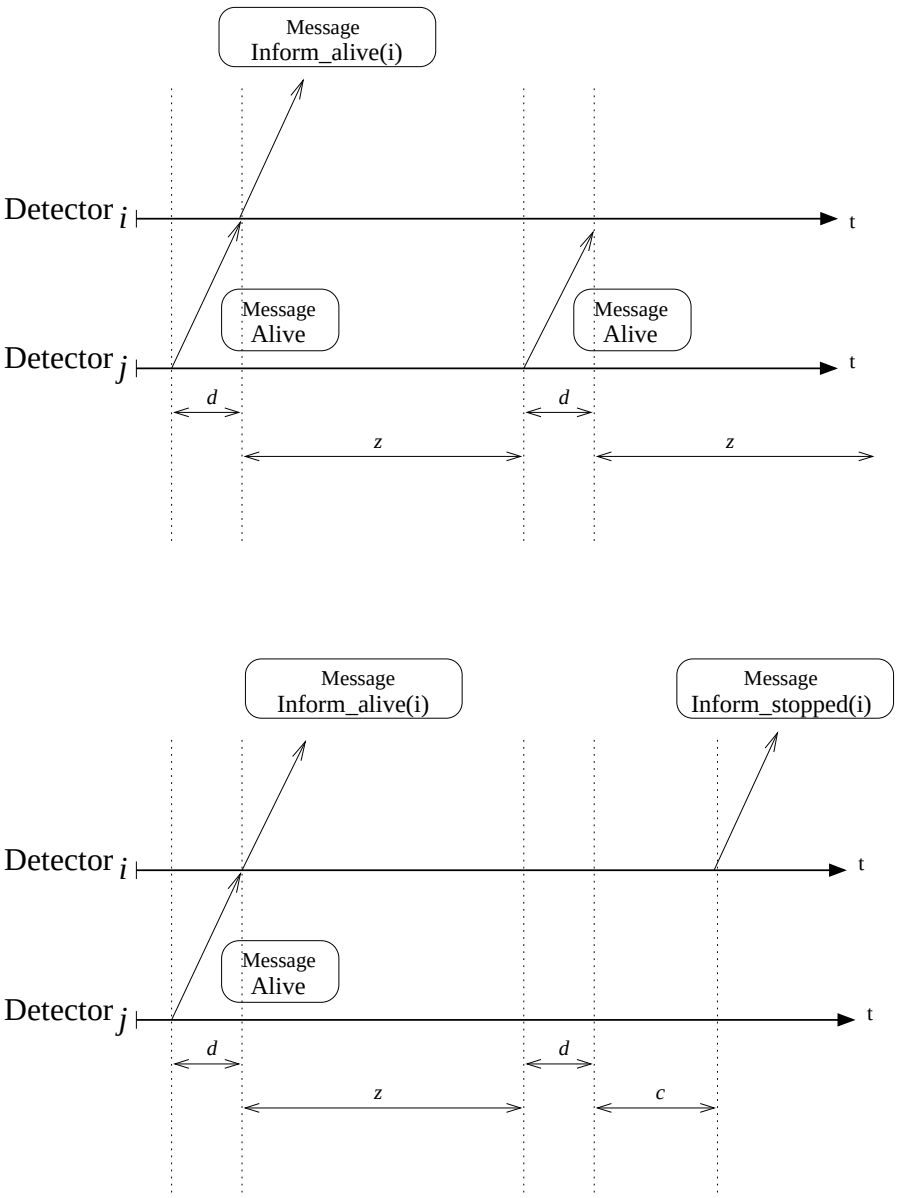

FIG. 4.4 – Deux chronogrammes d'exécution pour le DETECTOR$(z, c)_i$. d est le temps de transmission d'un message. z est l'intervalle entre deux émissions consécutives du message *Alive* et c celui entre deux réceptions. Dans le premier, le processus j envoie *Alive* et l'électeur maître LEADERELECTOR de i est informé par un message *InformAlive(j)*. Le second montre la détection de l'absence du message *Alive* de j. Le thread BPDETECTOR$_i$ l'annonce à LEADERELECTOR$_i$.

les intervalles de temps entre deux émissions et entre deux réceptions de messages *Alive*.

et celui de l'échange d'un message sur le réseau local et de sa réception est de quelques millisecondes.

Si l'exécution du sous-automate DETECTOR$(z, c)_i$ est stable pendant une durée au moins égale à $z + d + l$ et que le processus Paxos j est actif pendant cette période alors un message *InformAlive(j)*$_i$ est envoyé à l'électeur de maître LEADERELECTOR$_i$ et il ne peut y avoir de message *InformStopped(j)*$_i$.

Si le processus Paxos j est stoppé pendant une durée au moins égale à $z+c+l+2d$ et que l'exécution du processus i est stable, alors un message *InformStopped(j)*$_i$ est envoyé au sous-automate LEADERELECTOR$_i$ à l'issue cette période.

L'électeur de maître LEADERELECTOR

La difficulté à réaliser l'élection d'un maître dans un système distribué est du même ordre que celle du consensus qui dans un système asynchrone sujet aux erreurs est sans solution [FP85]. Puisque le détecteur n'est pas fiable, l'électeur de maître ne peut l'être non plus. Par conséquent soit plusieurs processus Paxos sont simultanément maîtres, soit celui qui devrait être maître n'est pas actif. Cependant lorsque l'exécution du sous-automate DETECTOR est correcte pendant la durée minimale requise, il devient fiable et par conséquent le LEADERELECTOR l'est également.

Le sous-automate LEADERELECTOR reçoit des messages de type *InformStopped(j)* et *InformAlive(j)* et met à jour l'état du processus j correspondant. Il existe deux types de messages qui modifient l'état connu du processus distant j dans le processus local i. L'état de j est soit `alive` soit `stopped`. Nous avons utilisé un tableau de bits, indexé par j, pour coder ces deux états. Un processus dans l'état `alive` est dit actif.

Le processus avec le plus grand identifiant parmi les processus actifs se déclare *maître*. L'identifiant utilisé pour cet élection doit être unique pour chaque processus Paxos et nous avons utilisé une combinaison de l'adresse réseau et du numéro de processus système. Ces deux composantes sont uniques dans leur ensemble de définition. Elles sont échangées dans l'en-tête des messages inter-processus à l'initialisation des connexions réseaux.

A chaque réception de *InformStopped(j)* et *InformAlive(j)*, le LEADERELECTOR vérifie s'il possède l'identifiant le plus grand parmi tous les processus actifs. Dans ce cas, il envoie un message *Leader* à l'automate Paxos. Dans le cas contraire LEADERELECTOR envoie un message *NotLeader*.

4.2.2 L'automate Paxos simple

L'automate Paxos simple tente de décider d'une valeur unique par consensus. Nous décrivons içi les trois sous-automates qui le composent : l'électeur AGENT, l'assesseur LEADER et l'annonceur SUCCESS. Cette décomposition permet de séparer les différents rôles de l'automate Paxos simple.

Les deux premières phases de *collecte* et de *proposition* de l'algorithme de consensus vont s'établir entre les élécteurs de chaque processus Paxos actif et l'assesseur du processus Paxos élu maître comme indiqué sur la figure 4.5.

La troisième et dernière phase d'*annonce* est réalisée, voir figure 4.6, entre les sous-automates SUCCESS. Le consensus ne peut-être achevé que si l'ensemble des

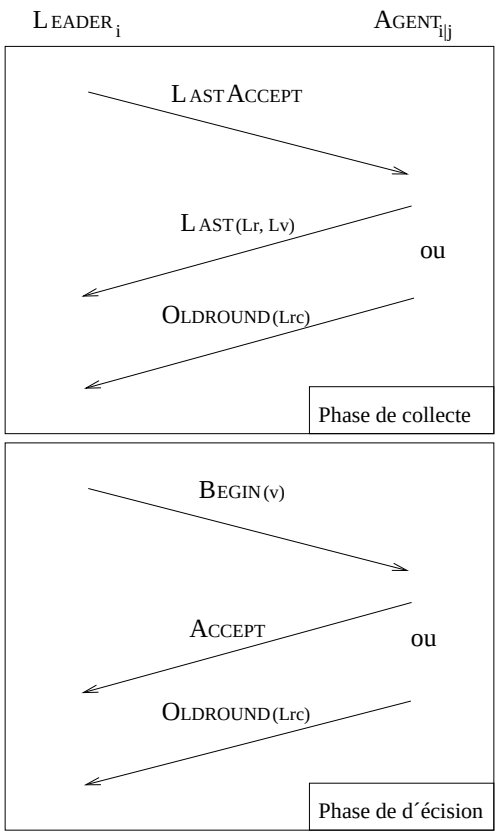

FIG. 4.5 – Messages échangés entre les électeurs et l'assesseur

processus Paxos a reçu la valeur décidée. L'un des processus peut-être défaillant et comme il n'est pas possible de distinguer ce cas d'une perte de message, l'annonce doit être réiterée périodiquement jusqu'à l'acquitement de tous les processus. De plus, le processus maître n'a aucun besoin d'acquitter la valeur décidée puisqu'il en est le détenteur. Ces particularités justifient l'existence d'un sous-automate séparé pour la phase d'annonce.

L'électeur AGENT

L'électeur AGENT opère dans les deux premières phases du déroulement du consensus. Il répond dans la phase de collecte au message *Collect(r)* et pendant la phase de décision au message *Begin(r,v)* envoyés par le LEADER[4].

L'origine des messages traités par l'électeur est montrée sur la figure 4.7. Ces messages ont deux origines :

1. Un contrôle extérieur à l'automate Paxos

[4]Il est à noter que si le processus i est dans l'état `leader`, alors le sous-automate AGENT_i reste actif : le processus i doit participer à l'élection que dirige LEADER_i pour qu'une majorité se dégage de l'ensemble des processus Paxos impliqués.

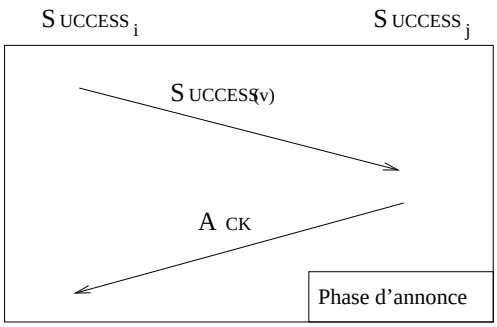

FIG. 4.6 – Messages échangés entre les annonceurs

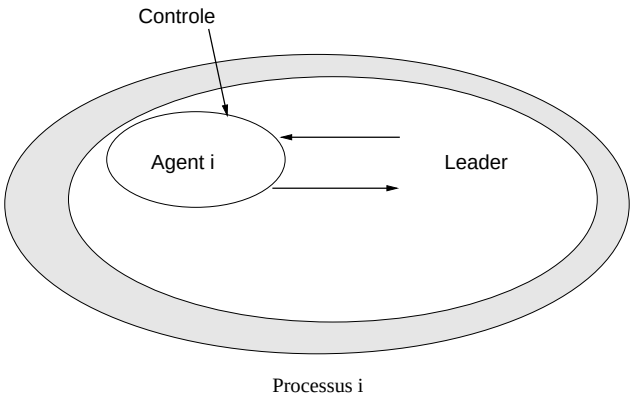

FIG. 4.7 – Les messages échangés par le sous-automate AGENT dans l'automate Paxos simple se font avec le LEADER ou sont reçus d'un contrôle extérieur qui permet d'instrumenter le système. Le sous-automate LEADER est soit local si le processus i est maître, soit distant.

  - Les messages de type *Alive* ou *Stopped* permettent de simuler la panne ou le rétablissement d'une machine.
  - Un message de type *Init(v)* fixe la valeur de décision *LastV* si celle-ci n'est pas encore définie.
2. L'assesseur du processus Paxos Maître
  - à un message de type *Collect(r)*, l'électeur répond par *Last(Lastr,Lastv)* pour indiquer qu'il a déjà participé à la décision de la valeur *Lastv* dans une ronde *Lastr* antérieure ou égale à r.
  - à un message de type *Begin(r,v)*auquel l'électeur répond par *Accept* pour accepter la valeur v. Les paramètres v et r sont affectés aux variables *Lastv* et *Lastr*.
  - si l'électeur a déjà accepté de participer à une ronde c ultérieure à r, il répond par une message de type *OldRound(c)*.

le thread BPAGENT, qui est l'impléntation du sous-automate AGENT, consiste en une lecture bloquante de la queue de messages AGENTQUEUE ; à chaque message

reçu correspond une action à traiter immédiatement. Trois variables locales sont requises : *Lastv*, *Lastr*, et *c*. Pour l'évaluation de l'implémentation, nous avons ajouté une variable contenant l'état d'activité `alive` ou `stopped`, pour simuler le début et la fin d'une panne. Cette variable est modifiée par l'envoi des messages *Stopped* et *Alive*.

L'assesseur LEADER

Le rôle de l'assesseur LEADER_i est de mener une élection à condition que le message *Leader* lui soit envoyé par LEADERELECTOR_i. La phase initiale du consensus, la collecte des valeurs, est exécutée lorsque le message *NewRound* est envoyé[5] par l'électeur de maître. Lorsque la phase de proposition de la valeur est terminée, il envoie un message *RndSuccess(v)* à l'annonceur qui prend en charge la phase finale du consensus.

Les processus électeurs qui répondent dans la deux premières phases sont notés dans une structure appelée *quorum*. Le quorum contient les identifiants uniques des processus. Le quorum pour la phase de collecte est nommé *InfoQuo* et le quorum contenant les élécteurs de la phase de proposition est *AcceptQuo*.

Si n processus participent au consensus, il suffit d'une majorité $(n/2) + 1$ pour que la valeur de décision soit acceptée. Il peut exister deux assesseurs car l'algorithme implanté dans l'automate LEADERELECTOR ne garantit pas l'existence d'un seul LEADER actif. Si dans un ensemble de n processus Paxos, deux majorités ($n/2 + 1$ processus) acceptaient la proposition d'un LEADER, alors il existerait un processus électeur Paxos commun aux deux majorités. Comme un sous-automate AGENT ne participe qu'à une seule décision dans une ronde donnée, le consensus ne peut aboutir qu'en présence d'un unique maître.

Chacune des étapes, présentées sur la figure 4.9, du déroulement du consensus correspond à un état du sous-automate :

1. Pour quitter l'état initial `done`, un message de démarrage d'une nouvelle ronde *NewRound* est requis pour passer dans l'état `collect`. De manière générale, le message *NewRound* force l'état `collect`, aussi dans les états `gatherlast`, `wait` et `gatheraccept` dans lesquels la queue de réception des messages est interrogée.
2. Dans le mode `collect`, le LEADER émet un message *Collect(r)* pour chacun des n processus Paxos, y compris lui-même. Le numéro de ronde initial $valfrom$ est fixé à la plus petite valeur $(0, i)$, i répresente l'identifiant du processus local. Les quorums *InfoQuo* et *AcceptQuo* sont réinitialisés et l'état devient `gatherlast`.
3. La phase de *collecte* se termine par la réception des messages *Last* qui contiennent un numéro de ronde R et une valeur de décision v. Pour chaque message, L'électeur émetteur est ajouté au quorum $InfoQuo$. Si la valeur v est définie et que son numéro de ronde est supérieur à $valfrom$, alors elles sont affectées à la valeur de décision finale $value$ et à $valfrom$. L'état passe à `gathered` dès que $infoquo$ contient $(n/2) + 1$ éléments distincts.
4. LEADER ayant collecté les valeurs, soit la valeur est définie et il passe dans l'état `begincast`, soit elle n'est pas définie et l'état suivant est `wait`.

[5] Nous présentons dans la section suivante l'automate STARTER chargé de démarrer une nouvelle ronde en cas de défaillance.

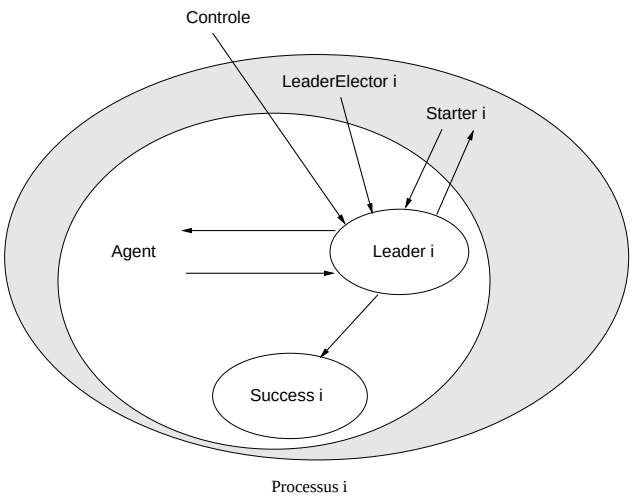

FIG. 4.8 – Les messages échangés par le sous-automate LEADER dans l'automate Paxos simple.

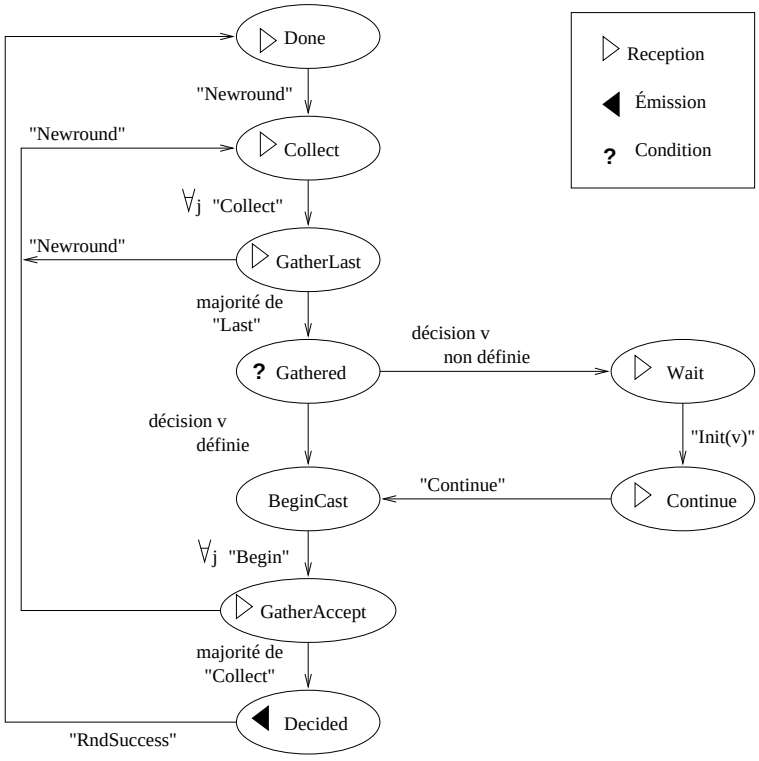

FIG. 4.9 – États du sous-automate LEADER dans le déroulement du consensus.

5. Dans l'état `wait`, le sous-automate attend une valeur d'initialisation exté-

rieure. Ce message *Init* peut ne jamais arriver, ou parvenir à un autre processus Paxos ; le rôle de l'automate de démarrage STARTER est de relancer une nouvelle ronde à intervalles réguliers. Si le message *Init* arrive, l'assesseur envoie un message *Continue* à STARTER pour indiquer que le déroulement du consensus progresse normalement avant de passer dans l'état `begincast`.

6. Après l'étape `begincast`, de diffusion d'un message *Begin* à tous les électeurs, l'état courant de *Leader* passe à `gatheraccept`.
7. La phase de *proposition* s'achève avec `begincast`. Les messages *Accept* sont collectés et les émetteurs distincts sont marqués dans le quorum *acceptquo*, et lorsque une majorité d'électeurs accepte la valeur pour la ronde courante, la valeur de décision est affectée et l'état `rndsuccess` est atteint.
8. Dans l'état `rndsuccess`, l'assesseur émet un message *RndSuccess(v)* vers le sous-automate SUCCESS. Le message est également transmis à STARTER pour indiquer que l'algorithme se déroule de façon conforme et évite le démarrage d'une nouvelle ronde. LEADER passe alors dans l'état `done`.

L'annonceur

Le rôle du sous-automate SUCCESS, figure 4.10, est de terminer la phase d'annonce du consensus et de fournir au programme ayant sollicité un consensus, la valeur de décision v.

Ce sous-automate est commun à l'ensemble des processus Paxos participants mais les actions sont différentes suivant que le processus est maître ou pas, c'est à dire que LEADERELECTOR a envoyé le message *Leader* ou *NotLeader*. Les états de SUCCESS sont représentés sur la figure 4.11.

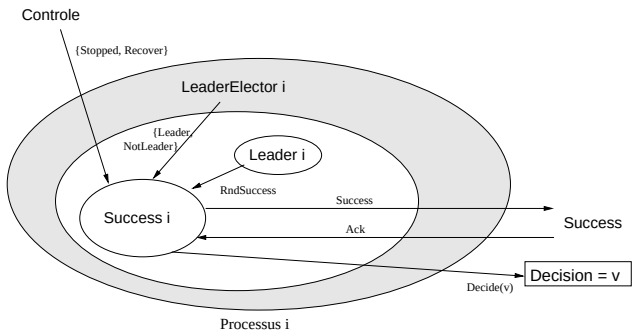

FIG. 4.10 – Les messages échangés par le sous-automate SUCCESS dans l'automate Paxos simple.

- Si le processus n'est pas élu maître, il répond à chaque message *Success(v)* par un message d'acquittement *Ack*. le message *Decide* retourne au programme ayant sollicité Paxos, la valeur de décision v.
- Si le processus Paxos i est maître, dès la réception du message *RndSuccess(v)* qui indique que la valeur de décision v a été acceptée, la valeur de décision v est connue et l'état `decide` peut-être atteint par le sous-automate SUCCESS, puis l'état suivant `sendsuccess` est validé. Le message *Success(v)* est envoyé

à chaque processus Paxos j non-maître, tel que $j \neq i$. Si le message d'acquittement d'un processus non-maître j ne parvient pas à i dans l'intervalle de temps $2d+2l$ l'état courant devient `check` et la vérification des acquittements reprend.

Il existe un temporisateur pour chacun des processus j Paxos participants. La durée $2d + 2l$ correspond à la durée du transit d'un message *Success(v)*envoyé par l'annonceur du processus i, de son traitement par le processus j, de la réponse *Ack*de j et de son traitement par i.

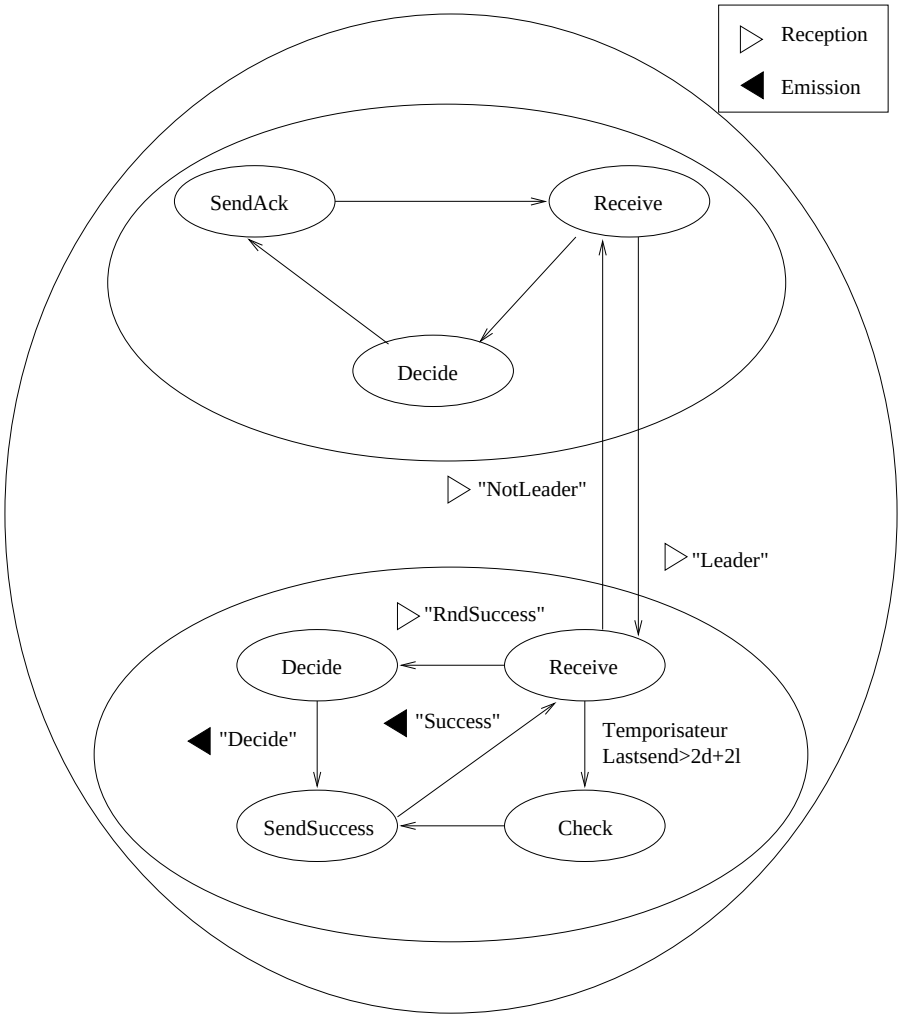

FIG. 4.11 – Les messages échangés par le sous-automate SUCCESS dans l'automate Paxos simple.

Automate de redémarrage STARTER

Pour atteindre le consensus, une ronde est initiée. Le démarrage d'une ronde est réalisé par l'automate STARTER. Le démarrage de plusieurs rondes par plusieurs

processus n'altère pas le fonctionnement du système mais chaque nouvelle ronde annule la précédente et peut empêcher l'automate d'atteindre le consensus. L'électeur de maître LeaderElector est donc utilisé pour autoriser le déclenchement de cet automate.

Dès que l'électeur de maître envoie le message *Leader*, Starter envoie le message *NewRound* pour démarrer une nouvelle ronde. Il envoie également ce message lorsque *Gathered(v)*, *Continue* et *RndSuccess* ne lui parviennent pas dans un délai donné.

4.2.3 Paxos multiple et application à la réplication de données

L'algorithme Paxos permet à des processus d'atteindre le consensus pour *une* valeur. Pour établir le consensus sur une séquence de k valeurs, Lamport a proposé, également dans [Lam98], l'algorithme *MultiPaxos*. L'implantation de cet algorithme nécéssite k instances de Paxos simples, constituées de k Agent, Leader et Success, et de k automates de démarrage Starter pour contrôler le déroulement.

MultiPaxos : une extension de Paxos simple

La méthode qui consiste à instancier k processus Paxos simples pose un problème dans la mesure où k est un paramètre. La création d'un grand nombre k de processus sature la mémoire du système. Les processus légers de communication doivent être modifiés pour distinguer les échanges de messages entre les k processus. L'étape de collecte est commune à toutes les instances k donc il est inutile que k processus maître dirigent k fois la première phase du consensus.

Notre implémentation de Paxos simple ne nécessite que quelques modifications pour implanter l'algorithme *MultiPaxos*. Ces modifications concernent principalement les variables et les paramètres des messages échangés. La première modification est l'extention des structures simples en tableaux de structures dynamiquement allouées en fonction du paramètre k. La seconde est la modification des arguments échangés dans les messages.

Pour démarrer la collecte en initiant une ronde, l'assesseur envoie un message *Collect* qui contient le numéro de la ronde, un ensemble D de numéros d'instances pour lesquelles la décision est connue et un ensemble W pour ceux dont la valeur est initialisée mais n'a pas été décidée. Les électeurs répondent par un message *Last* avec deux arguments. Le premier contient les couples $(k, d(k))$, composés d'une instance k et de la décision de l'instance $d(k)$, pour lesquels l'électeur connait la décision mais pas l'assesseur. Le second contient les triplets $(k, r[k], v[k])$ pour lesquels l'électeur a accepté la valeur initiale $v[k]$ dans une ronde $r[k]$ de l'instance k.

La phase de proposition dans *MultiPaxos* est la répétition de k phases de Paxos simple. Les numéros d'instances k sont ajoutés aux arguments des messages. L'assesseur émet vers chacun des processus un message *Begin* pour toutes les instances k où la valeur $v[k]$ a été initialisée. Pour distinguer l'état du sous-automate Leader suivant les instances k, les états `gathered`, `continue`, `begincast` et `gatheraccept` sont indexés par l'argument k des messages reçus. Les électeurs acceptent la valeur $v[k]$ s'il n'ont pas accepté de participer à une ronde $r[k]$ supérieure.

La phase d'annonce de *MultiPaxos*, gérée par le sous-automate SUCCESS, est similaire à celle de Paxos simple. Il suffit d'ajouter l'argument k aux messages et d'indexer les variables nécessaires : acquitements et temporisateurs. L'automate de démarrage STARTER requiert le même type de modification que SUCCESS.

La multiplication du nombre de messages échangés dans MultiPaxos pose un problème de garantie des délais dans les automates. MultiPaxos sature le réseau par des messages courts, de faible charge utile[6]. Nous avons utilisé des files de messages de taille fixe pour éviter l'allocation dynamique d'espace mémoire et accélérer la gestion des messages. Nous avons modifié le thread d'émission NETSEND de manière à ce qu'il regroupe les messages pour le même processus destinataire dans le même envoi de datagramme UDP. Le thread NETRECV est chargé de les découper à la réception.

4.3 Évaluation de notre implantation

Nous présentons notre implémentation de l'automate Paxos en décrivant le déroulement de son exécution. La durée du consensus est un paramètre à analyser pour valider son utilisation dans le gestionnaire d'état global du système de disque virtuel. Nous présentons les optimisations que nous avons apportées.

4.3.1 Mesure du temps de consensus

Nous présentons trois traces d'exécution d'une instance de Paxos simple. Le programme a été exécuté sur quatre machines. Les trois traces correspondent à une même exécution sur trois machines distinctes.

Sur chacune des figures 4.12, 4.14 et 4.13, le processus identifié par 0 représente le processus local.

Analyse de l'exécution d'un processus Paxos électeur

L'exécution d'un processus Paxos électeur 0 est présentée sur la figure 4.12. Le processus local est toujours identifié par le numéro 0. À la ligne 1, le thread BPAGENT reçoit le message *Collect* du processus Paxos 2. Ce processus s'est élu maître à tort. En ligne 2, le processus Paxos 1, lance une phase de collecte avec le numéro de ronde 12. Le numéro de ronde pour la phase de collecte est supérieur au précédent, le processus doit accepter d'y participer. Les messages *Alive* de la ligne 3 à 10, montrent que les threads BPDETECTOR vérifie en permanence le bon fonctionnement de tous les processus Paxos impliqués dans le consensus.

A la ligne 11, le processus Paxos 1 commence la phase de proposition par un message *Begin* pour la ronde 13 avec la valeur 4. Le processus Paxos accepte cette proposition, retourne *Accept*. Le processus maître a reçu un majorité de *Accept*et le thread BPSUCCESS du processus Paxos reçoit le message *Success*, ligne 13.

[6]Comme nous utilisons UDP, les messages qui représentent une dizaine d'octets en moyenne, sont encapsulés par UDP avec un en-tête de 8 octets, puis par IP avec un en-tête de 20 octets et enfin par Ethernet avec un en-tête de 18 octets, ce qui représente une charge utile de $10/46 = 21,7\%$.

```
 1   <6.548>  BPagent()   Collect from 2 for round=1,0
     <6.568>  BPagent()   Collect from 1 for round=12,0
     <7.348>  Detector()  Alive from 1
     <8.032>  Detector()  Alive from 3
 5   <8.128>  Detector()  Alive from 0
     <8.228>  Detector()  Alive from 2
     ...
     <16.078> Detector()  Alive from 3
     <16.158> Detector()  Alive from 0
10   <16.318> Detector()  Alive from 2
     <16.768> BPagent()   Begin from 1 for round=13,0
                          with value 4,val
     <17.140> BPsuccess() Success from 1 (decision=4)
     <17.380> BPsuccess() ******* DECIDE ****** = 4
15   main 25883 get Decision paxos : 4
```

FIG. 4.12 – Trace de l'exécution de Paxos simple pour un processus électeur.

La phase de collecte se termine à la ligne 10. Durant cette phase, l'exécution n'était pas stable. Le processus Paxos 2, dont nous allons décrire l'exécution à partir de la figure 4.14, a été élu maître par son module BPLEADERELECTOR par défaut. La durée du consensus a été allongée par le démarrage de nombreuses rondes supplémentaires. La phase de proposition, entre les lignes 11 et 13 et la phase d'annonce, entre les lignes 13 et 14, ont duré 612 millisecondes.

Analyse de l'exécution d'un processus Paxos assesseur

Sur la figure 4.13, nous présentons le déroulement du consensus sur le processus Paxos maître à partir de l'instant où le système est stabilisé. Une nouvelle ronde est initiée par le thread BPSTARTER. Le thread BPLEADER reçoit le message *Init* et envoie aux quatre processus impliqués un message *Collect*. Le thread BPAGENT local est le premier à le recevoir, ligne 2. Quatre réponses à la phase de collecte sont reçus par BPLEADER, lignes 3 à 10, mais aucune ne contient de valeur d'initialisation.

Sans valeur d'initialisation de la valeur décision, le processus BPLEADER est bloqué dans l'état `wait`. Nous procédons à la saisie manuelle d'une valeur d'initialisation, à la ligne 19. Aussitôt, ligne 22, le BPLEADER commence la phase de proposition en émettant un message *Begin*, reçu localement par le thread BPAGENT. La valeur est acceptée par les quatre processus Paxos. En fait le quorum d'acceptation contient déjà une majorité ($n = 4$ et $n/2 + 1 = 3$) dès la réception du message *Accept* du processus 2, ligne 26. La réponse du processus 3 en ligne 27 est ignorée par le BPLEADER qui informe et déclenche le thread BPSUCCESS.

La valeur de décision est diffusée aux processus Paxos distants (1, 2 et 3), lignes 30 à 32, et les acquittements sont reçus 6 centièmes de seconde plus tard. Une fois tous les messages d'acquittement traités, lignes 36 à 43, le thread BPSUCCESS retourne la valeur de décision au programme appelant, ligne 45.

À partir du moment la valeur de décision est initialisée, ligne 20, le consensus

```
 1   <9.373>  BPleader()  Newround from 0 (current round=13)
     <9.433>  BPagent()   Collect from 0 for round=13,0
     <9.433>  BPleader()  Last from 0 for round=13,0
                            with value -1,nil and r'=0,0
 5   <9.453>  BPleader()  Last from 1 for round=13,0
                            with value -1,nil and r'=0,0
     <9.473>  BPleader()  Last from 2 for round=13,0
                            with value -1,nil and r'=0,0
     <9.493>  BPleader()  Last from 3 for round=13,0
10                          with value -1,nil and r'=0,0
     <10.193> Detector()  Alive from 0
     <10.853> Detector()  Alive from 3
     <10.933> Detector()  Alive from 1
     <11.053> Detector()  Alive from 2
15   <12.233> Detector()  Alive from 0
     <12.853> Detector()  Alive from 3
     <12.933> Detector()  Alive from 1
     <13.093> Detector()  Alive from 2
     Processing i 4
20   <13.473> BPleader()  Init from 0 with value 4,val
     <13.493> BPagent()   Init from 0 with value 4,val
     <13.553> BPagent()   Begin from 0 for round=13,0
                            with value 4,val
     <13.553> BPleader()  Accept from 0 for round=13,0
25   <13.573> BPleader()  Accept from 1 for round=13,0
     <13.593> BPleader()  Accept from 2 for round=13,0
     <13.613> BPleader()  Accept from 3 for round=13,0
     <13.673> BPsuccess() RndSuccess from 0 with val=4
     <13.893> BPsuccess() alive=1 leader=1 allacked=0 decision=4,val
30   <13.893> BPsuccess() SendSuccess to 1 with decision 4
     <13.893> BPsuccess() SendSuccess to 2 with decision 4
     <13.893> BPsuccess() SendSuccess to 3 with decision 4
     <13.953> BPsuccess() Ack from 3
     <13.973> BPsuccess() Ack from 1
35   <13.993> BPsuccess() Ack from 2
     <14.133> BPsuccess() alive=1 leader=1 allacked=0 decision=4,val
     <14.133> BPsuccess() No SendSuccess
                            prevsend=13.892999 && decision=4,val
     ...
40   <14.313> BPsuccess() alive=1 leader=1 allacked=0 decision=4,val
     <14.313> BPsuccess() No SendSuccess
                            prevsend=13.892999 && decision=4,val
     <14.333> BPsuccess() alive=1 leader=1 allacked=0 decision=4,val
     <14.353> BPsuccess() ******* DECIDE ****** = 4
45   main 10753 get Decision paxos : 4
```

FIG. 4.13 – Trace de l'exécution de Paxos simple pour l'assesseur, le processus élu maître.

est réalisé et annoncé, ligne 44, en 880 millisecondes.

Analyse de l'exécution d'un processus Paxos assesseur puis électeur

```
 1   <0.412>  BPleader()  Newround from 0 (current round=1)
     <0.472>  BPagent()   Collect from 0 for round=1,0
     <0.472>  BPleader()  OldRound from 1 for round=1,0 and r'=11,0
     <0.492>  BPleader()  Last from 0 for round=1,0
 5                           with value -1,nil and r'=0,0
     <0.492>  BPagent()   Collect from 2 for round=12,0
     <0.512>  BPleader()  OldRound from 3 for round=1,0 and r'=11,0
     <0.532>  BPleader()  OldRound from 2 for round=1,0 and r'=11,0
     <1.272>  Detector()  Alive from 2
10   <1.952>  Detector()  Alive from 3
     ...
     <6.572>  BPagent()   Collect from 2 for round=13,0
     <7.312>  Detector()  Alive from 2
     <7.992>  Detector()  Alive from 3
15   <8.052>  Detector()  Alive from 1
     <8.192>  Detector()  Alive from 0
     <9.352>  Detector()  Alive from 2
     <9.992>  Detector()  Alive from 3
     <10.052> Detector()  Alive from 1
20   <10.232> Detector()  Alive from 0
     <10.692> BPagent()   Begin from 2 for round=13,0
                             with value 4,val
     <11.053> BPsuccess() Success from 2 (decision=4)
     <11.272> BPsuccess() ******* DECIDE ****** = 4
25   main 9281 get Decision paxos : 4
```

FIG. 4.14 – Trace de l'exécution de Paxos simple pour un processus électeur. Le processus a démarré avant les autres processus Paxos. Il est désigné assesseur puis électeur par son LEADERELECTOR.

Un exemple d'exécution, initialement instable est représentée sur la figure 4.14. Le processus Paxos est élu maître et initie une nouvelle ronde, initialement à 0 et incrémentée à 1, à la ligne 1. Son thread BPAGENT répond par un message *Collect* mais les processus 1, 2 et 3 retournent un message *OldRound*, ligne 3, 7 et 8, avec un numéro de ronde à 11. Avant même de recommencer une ronde avec un numéro de ronde supérieur à 12, le processus est informé du bon fonctionnement du processus 2. Le processus 2 est maître car la ligne 6 montre qu'il a organisé une collecte. Le processus est informé par BPLEADERELECTOR qu'il n'est plus `leader`. Le thread BPLEADER du processus 0 est inhibé.

Le processus 2 organise une collecte avec le numéro de ronde 13, ligne 12 et la phase de proposition commence à la ligne 21. Ces messages correspondent aux phases organisées par le processus maître de la figure 4.13, lignes 2 et 22. Le processus acquite la décision de la valeur 4, figure 4.14 ligne 23 et retourne la valeur au programme ligne 25.

4.3.2 Mesure de la charge de calcul

L'implémentation des automates LeaderElector, Success et Starter requiert la détection du passage du temps et le respect des temporisations. Nous décrivons notre approche pour appliquer ces contraintes tout en minimisant la charge de calcul des machines sur lesquelles est exécuté Paxos.

Nous avons réalisé une première implémentation de Paxos basique avec des verrous d'exclusions mutuelles, les *mutex*, et des sémaphores, pour l'accès aux queues de messages. Les lectures et les écritures sont dans ce cas bloquantes. Nous avons augmenté le programme d'un *thread* Alarm supplémentaire chargé d'envoyer des interruptions aux trois threads bpLeaderElector, bpStarter et bpSuccess.

Nous avons été confrontés à un problème de portabilité entre les systèmes *Linux*, *Solaris* de Sun et *Irix* de Silicon Graphics sur lesquels nous avons testés notre implantation de l'algorithme. La gestion des interruptions est différente sur les trois systèmes.

Notre deuxième approche consiste à rendre la lecture et l'écriture des files de messages non bloquantes. Dans chaque automate, la boucle d'événements de chaque processus vérifie d'abord que les transitions basées sur le passage du temps sont exécutées, puis interroge sa file d'attente de messages. Le test des conditions de temps dans la boucle, lorsqu'il est répété rapidemment, peut saturer les capacités de calcul de la machine. En fait, l'utilisation des *mutex* nécessite un appel au système. La charge de calcul engendrée par l'automate Paxos pour atteindre le consensus ne représente pas 1% de la charge nominale[7].

Nous présentons dans le tableau 4.15, la charge de calcul sur chacune des machines utilisées en précisant le système d'exploitation, le type de processeur et sa cadence d'horloge. Le nombre de processus Paxos impliqués dans le consensus est précisé. Ces résultats indiquent dans l'ensemble des valeurs de charge de calcul plus faibles que celle demandées par le serveur de disque virtuel. Les temps observés pour l'architecture Irix témoignent de la mauvaise implantation de la librairie *Pthread* sur ce système.

Système	Processeur	nombre de processus Paxos	Temps	
			Système	Utilisateur
Linux	alpha 266 Mhz	4	5 ms	10 ms
Linux	alpha 266 Mhz	8	8 ms	12 ms
SunOS 5.7	sparc 120 Mhz	4	0 = non mesurable	
Irix	R10000 120 Mhz	4	148ms	323ms
Linux	PIII i686 600 Mhz	4	0 = non mesurable	
Linux	AMD Athlon i686 1.5Ghz	4	0 = non mesurable	

Fig. 4.15 – Temps de calcul pour le déroulement du consensus sur 6 systèmes.

Notre implémention en language *C* représente 2500 lignes de code. Nous pensons

[7]Ce pourcentage est trop faible pour observer une variation entre les différentes machines utilisées.

que la décomposition proposée par Roberto De Prisco [PLL97] peut être reformulée pour obtenir une mise en œuvre plus simple.

4.3.3 Sécurité

Le consensus atteint par Paxos repose sur l'absence de messages volontairement destructifs. L. Lamport a montré [LLP82] que dans un système distribué, le problème n'a de solution que si plus de 2/3 des participants ont un comportement loyal[8]. Avec un système de signature numérique, le problème peut-être résolu quel que soit le nombre de participants loyaux et traitres.

Nous n'avons pas implémenté un tel système de signature dans notre protocole d'échange entre les processus Paxos. Néanmoins, l'ajout d'une négociation de clés numériques à l'initialisation et l'ajout des clés dans les messages échangés est envisageable.

[8]Le problème des généraux byzantins est une métaphore pour le problème de consensus dans un système distribué où certains membres corrompent les messages. Les généraux doivent décider de l'assaut ou de la retraite. Un général décide de la tactique adoptée et transmet ses ordres aux autres généraux mais certains falsifient le message. Les processus loyaux ont un comportement dit non-byzantin.

Chapitre 5

Conclusion et perspectives

Nous avons présenté notre architecture de disque virtuel, un volume de stockage distribué composé de serveurs disques répartis tolérant aux pannes. Cette architecture présente les caractéristiques demandées à un système de stockage moderne : disponibilité, fiabilité, transparence et extensibilité.

- La *disponibilité* est apportée par la parallélisation des accès au disque virtuel et la répartition des requêtes sur l'ensemble des serveurs qui le composent.
- La *fiabilité* repose sur l'intégration de l'algorithme de consensus distribué Paxos au serveur de disques virtuels.
- La *transparence* découle de la représentation d'un ensemble de serveurs de disques sous la forme d'un disque virtuel qui présente les même caractéristiques qu'un périphérique de stockage à accès direct.

La *sécurité* n'est pas une fonctionnalité laissée pour compte. Mais nous pensons que des mécanismes d'authentification et de cryptage peuvent être ajoutés sans modification notable de l'architecture[1].

5.1 État actuel de notre projet

Notre architecture est inspirée du système de disque propriétaire *Petal* [LT96] qui intègre l'algorithme de consensus distribué Paxos [Lam98] [Lam02] pour la tolérence aux pannes. Nous avons réalisé une implémentation de Paxos par processus légers.

Notre implémentation du disque virtuel

Nous avons réalisé une implémentation du serveur de disques virtuels et intégré le schéma de réplication appelé *dégroupement chaîné* [HD90]. Nos tests ont montré que les performances du serveur de disques virtuels augmentent linéairement avec le nombre de serveurs[2]. Ces tests montrent que l'utilisation de stations de travail

[1] Par exemple le gestionnaire de disque partagé NBD [BLA00] a intégré la librairie de socket sécurisée SSL avec moins de 30 lignes supplémentaires de code *C*.

[2] Nous avons utilisé douze stations de travail disponibles pour réaliser nos tests. Les performances obtenues dépendent surtout du réseau. En augmentant le nombre de stations et de switchs,

comme serveurs et clients est possible. Ce disque virtuel offre un débit crête en lecture de 65 Mo/s avec 8 machines, ce qui représente 93% de la bande passante de notre réseau Ethernet 100Mbits. Les modules d'état et de restauration qui assurent la fiabilité du système, nécessitent l'intégration de Paxos.

La version du système sur laquelle nous avons effectuée ces mesures utilise une version préliminaire de l'algorithme de consensus ; elle ne dispose donc pas de la reconfiguration dynamique des groupes de serveurs.

Notre implémentation de Paxos simple

Notre implémentation de l'algorithme Paxos simple par processus légers réalise le consensus par passages de messages en mode non-connecté UDP. L'arrêt de machines et la perte de messages n'altèrent pas le processus. Dans des conditions d'exécution réalistes[3], le consensus est établi en moins de 1 seconde pour 8 machines.

5.2 Perspectives

Nous présentons ici les principales évolutions que nous comptons apporter à notre système. Ces modifications concernent l'intégration des deux parties déjà réalisées : disque virtuel et algorithme de consensus, et d'autre part la réalisation d'un système de fichier spécifique qui exploite les caractéristiques particulières de notre disque virtuel.

5.2.1 Optimisation et intégration de MultiPaxos

Pour intégrer Paxos à notre serveur de disque virtuel, nous devons étendre notre réalisation à MultiPaxos qui réalise le consensus sur un ensemble de valeurs et l'appliquer à la réplication de données [GL00] [PLL97]. La réplication de données nécessite d'étendre le consensus, qui pour le moment se fait sur des entiers, à des types et à des structures complexes.

Notre décomposition de l'algorithme de consensus simplifie l'implémentation de cette extension en séparant les rôles des acteurs dans les étapes de collecte, de proposition et d'annonce du consensus.

Nous projetons de comparer l'efficacité de Paxos intégré au disque virtuel avec l'utilisation d'autres algorithmes de réplication tolérant aux pannes [AsC85] [LO88]. Ceux-ci sont moins généraux ce qui limite l'utilisation que nous pouvons en faire dans notre système, mais nous envisageons que leur spécialisation apporte des performances supérieures pour les cas les plus communs.

---

les performances continueront à augmenter. L'utilisation d'un réseau Gigabit-Ethernet, permettrait d'augmenter encore les débits, par contre le débit maximum des disques des stations de travail seraient une nouvelle limitation.

[3] À mesure que les défaillances du réseau ou des serveurs augmentent, le temps nécessaire pour l'obtention du consensus augmente également.

5.2.2 Système de fichiers distribué

Avec notre disque virtuel actuel, nous ne sommes en mesure de placer par dessus que des systèmes de fichiers classiques — type FFS BSD ou EXT2FS Linux. Ces systèmes ne peuvent pas exploiter les spécificités de notre système de stockage. Nous comptons donc réaliser un système de fichiers spécialisé, qui pourra exploiter au mieux les caractéristiques de notre disque virtuel.

Ce système de fichiers devra bien sur offrir une interface classique : traduire la position des fichiers en blocs sur le disque, stocker les attributs de protection et de propriéte des fichiers et permettre une organisation hiérarchique des fichiers.

D'un autre coté, notre système de fichiers s'adaptera au support de stockage particulier que constitue notre disque virtuel. Sa principale caractéristique est d'offrir un volume de stockage réel important et de permettre un adressage *virtuel* indépendant de l'adresse physique des blocs. L'espace d'adressage physique est de plus susceptible de varier au gré de l'ajout et du retrait de serveurs ou de disques.

Le système de fichiers profitera de l'adressage creux[4] du disque virtuel. Ce principe est actuellement utilisé sur le système de fichier Frangipani [TML97] : un immense disque virtuel est découpé en zones de plusieurs Téra-octets ; chaque zone est dévolue au stockage de fichiers de tailles distinctes ; Le système de fichier organise différement ses tables d'allocations dans chaque zone.

Une autre piste à explorer réside dans les systèmes de fichiers journalisés, et plus particulièrement les systèmes de fichiers du type de LFS [RO90] ; dans ces systèmes, le contenu d'un bloc n'est jamais modifié ; toutes les modifications des contenus des fichiers ou des méta-informations sont réalisées par l'écriture de nouveaux blocs ; une tâche en arrière plan est chargée de récupérer les blocs dont le contenu à été modifié, et est donc ré-écrit ailleurs ; elle recompacte également les blocs qui ne sont plus que partiellement utilisés, en les regroupant dans de nouveaux blocs. Ce système de fichier fut conçu à l'origine afin de permettre une écriture séquentielle des secteurs sur le disque, ce qui optimise l'exploitation du disque en minimisant les mouvements des bras pour les écritures[5], mais nous pensons qu'il serait bien adapté au mécanisme d'adressage virtuel des blocs de notre disque.

Nous pensons résoudre le problème posé par la concurrence des accès aux fichiers par un *gestionnaire de verrous* qui interagira avec le disque virtuel. Le verrouillage sera réalisé au niveau des blocs du disque virtuel ; ce verrouillage à grain fin permettra de réduire les attentes pour des modifications concurrentes[6].

[4] La correspondance entre les blocs virtuels et les blocs physiques est réalisée lors d'un accès en lecture ou en écriture à l'adresse virtuelle correspondant au bloc par allocation dynamique. C'est un espace de stockage à trou.

[5] Le recompactage des blocs peut également servir à optimiser les lectures séquentielles en réécrivant les contenus des fichiers dans des secteurs contigus.

[6] Naturellement, l'utilisation de ces caractéristiques reposera en définitive sur son utilisation par les applications ; il sera nécessaire d'avoir également des mécanismes de verrouillage plus grossiers dont la sémantique sera équivalente à celle que l'on trouve sur les systèmes de fichiers actuels, pour permettre aux applications existantes de continuer à fonctionner sans modification.

Bibliographie

[ABP+94] C. Attanasio, M. Butrico, C. Polyzois, S. Smith, and J. Peterson. Design and implementation of a recoverable virtual shared disk. Technical Report RC 19843, IBM Research, 1994.

[ADN+95] Thomas Anderson, Michael Dahlin, Jeanna Neefe, David Pat-terson, Drew Roselli, and Randolph Wang. Serverless network file systems. In *Proceedings of the 15th Symposium on Operating System Principles. ACM*, pages 109–126, Copper Mountain Resort, Colorado, décembre 1995.

[And99] Kenneth Preslan Andrew. A 64-bit, shared disk file system for linux. Technical report, Sistina Sodtware, Inc, 1999.

[AsC85] A. El Abbadi, D. skeen, and F. Cristian. An efficient fault-tolerent protocol for replicated data management. In *Proceedings of the 4th ACM SIGACT/SIGMOD Conference on principles of database Systems*, 1985.

[BDET00] William J. Bolosky, John R. Douceur, David Ely, and Marvin Theimer. Feasibility of a serverless distributed file system deployed on an existing set of desktop PCs. In *Measurement and Modeling of Computer Systems*, pages 34–43, 2000.

[BJPP+00] Francisco J. Ballesteros, Ricardo Jimenez-Peris, Marta Patinez, Fabio Kon, Sergio Arevalo, and Roy H. Campbell. Using interpreted compositecalls to improve operating system services. *SP&E*, 30(6) :589–615, 2000.

[BLA00] P. T. Breuer, A. Marn Lopez, and Arturo García Ares. The network block device. *Linux Journal*, 73, mai 2000.

[BN99] Peter Braam and Philip Nelson. Removing bottlenecks in distributed filesystems. In *Proceedings of the 5th Annual Linux Expo*, pages 131–139, Raleigh, North Carolina, mai 1999.

[Bra99] Peter J. Braam. File systems for clusters from a protocol perspective. In *Usenix 99*, School of Computer Science, Carnegie Mellon University, 1999.

[CLBHL92] J. S. Chase, H. M. Levy, M. Baker-Harvey, and E. D. Lazowska. How to use a 64-bit virtual address space. Technical Report TR-92-03-02, Department of Computer Science and Engineering, University of Washington, 1992.

[CM99] Dell Computer and Maxtor. The increasing demand for hard-disk drive capacity. *Dell Technology Brief*, septembre 1999.

[CSWH00] Ian Clarke, Oskar Sandberg, Brandon Wiley, and Theodore W. Hong. Freenet : A distributed anonymous information storage and retrieval system, 2000.

[dJKH93] Wiebren de Jonge, M. Frans Kaashoek, and Wilson C. Hsieh. The Logical Disk : a new approach to improving file systems. *j-OPER-SYS-REV*, 27(5) :15–28, décembre 1993.

[DKK+01] Frank Dabek, M. Frans Kaashoek, David Karger, Robert Morris, and Ion Stoica. Wide-area cooperative storage with CFS. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP '01)*, Chateau Lake Louise, Banff, Canada, October 2001.

[DST87] Dean S. Daniels, Alfred Z. Spector, and Dean S. Thompson. Distributed logging for transaction processing. In *ACM SIGMOD Annual Conference*, pages 82–96, mai 1987. ACM Special Interest Group on Management of Data.

[DW01] John R. Douceur and Roger Wattenhofer. Modeling replica placement in a distributed file system : Narrowing the gap between analysis and simulation. In *European Symposium on Algorithms*, pages 356–367, 2001.

[FP85] M. J. Fischer and M. S. Paterson. Impossibility of distributed consensus with one faulty process. In *Journal of the ACM*, volume 32, pages 374–382, 1985.

[GL00] Eli Gafni and Leslie Lamport. Disk paxos. In *International Symposium on Distributed Computing*, pages 330–344, 2000.

[GPRA98] Douglas P. Ghormley, David Petrou, Steven H. Rodrigues, and Thomas E. Anderson. SLIC : An extensibility system for commodity operating systems. In *In USENIX 1998 Annual Technical Conference*, pages 39–52, juin 1998.

[GSC+95] Garth A. Gibson, Daniel Stodolsky, Pay W. Chang, William V. Courtright II, Chris G. Demetriou, Eka Ginting, Mark Holland, Qingming Ma, LeAnn Neal, R. Hugo Patterson, Jiawen Su, Rachad Youssef, and Jim Zelenka. The Scotch parallel storage systems. In *Proceedings of 40th IEEE Computer Society International Conference (COMPCON 95)*, pages 403–410, San Francisco, printemps 1995.

[GWP99] Bjorn Grönvall, Assar Westerlund, and Stephen Pink. The design of a multicast-based distributed file system. In *Proceedings of the 3rd Symposium on Operating Systems Design and Implementation*, 1999.

[HD90] Hui-I Hsiao and David DeWitt. Chained Declustering : A new availability strategy for multiprocessor database machines. In *Proceedings of 6th International Data Engineering Conference*, pages 456–465, 1990.

[HO95] John H. Hartman and John K. Ousterhout. The Zebra striped network file system. *ACM Transactions on Computer Systems*, 13(3) :274–310, 1995.

[JF99] Minwen Ji and Edward W. Felten. Design and implementation of an island-based file system. Technical Report TR-610-99, Department of Computer Science, Priceton University, octobre 1999.

[KBC+00] John Kubiatowicz, David Bindel, Yan Chen, Patrick Eaton, Dennis Geels, Ramakrishna Gummadi, Sean Rhea, Hakim Weatherspoon, Westly Weimer, Christopher Wells, and Ben Zhao. Oceanstore : An architecture for global-scale persistent storage. In *Proceedings of ACM ASPLOS*. ACM, novembre 2000.

[Kle86] Steve R. Kleiman. Vnodes : An architecture for multiple file system types in sun UNIX. In *USENIX Summer*, pages 238–247, 1986.

[KS92] J. Kistler and M. Satyanarayanan. Disconnected operation in the coda file system, 1992.

[KS95] Puneet Kumar and M. Satyanarayanan. Flexible and safe resolution of file conflicts. In *Usenix Tech. Conf.*, New Orleans LA (USA), 1995.

[Lab00] AT&T Bell Laboratories. Plan 9 *from Bell Labs : Programmer's Manual*. Computing science Research Center, Murray hill, New Jersey, third edition, 2000.

[Lam78] Leslie Lamport. Time, clocks, and the ordering of events in distributed systems. *Communications of ACM*, 21(7), 1978.

[Lam87] Leslie Lamport. mail message sent to a DEC SRC bulletin board, mai 1987. A distributed system is one in which the failure of a computer you didn't even know existed can render your own computer unusable.

[Lam98] Leslie Lamport. The part-time parliament. *ACM Transactions on Computer Systems*, 16(2) :133–169, 1998.

[Lam02] Leslie Lamport. Paxos made simple. 2002.

[LD02] Pierre Lombard and Yves Denneulin. nfsp : A distributed nfs server for clusters of workstation. submitted to International Parallel and Distributed Processing Symposium 2002, 2002.

[Lee95] Edward K. Lee. Highly-available, scalable network storage. In *In Proceedings of CompCon'95*, mars 1995.

[LLP82] R. Shostak L. Lamport and M. Pease. The byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, 4(3), juillet 1982.

[LLS99] Yui-Wah Lee, Kwong-Sak Leung, and Mahadev Satyanarayanan. Operation-based update propagation in a mobile file system. In *Proceeding of the USENIC Annual Technical Reference*, Monterrey, California, USA, juin 1999.

[LO88] B. Liskov and B. Oki. Viewstamped replication : A new primary copy method to support highly-available distributed systems. In *Proceedings of the 7th Annual ACM syposioum on principle of ditributed computing*, pages 8–17, août 1988.

[LT96] Edward K. Lee and Chandramohan A. Thekkath. Petal : Distributed virtual disks. In *Proceedings of the Seventh International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-VII)*, pages pp. 84–92, Cambridge MA, octobre 1-5 1996.

[Mac94] Rick Macklem. Not quite nfs, soft cache consistency for nfs. In *Winter USENIX Conference Proceedings, USENIX Association*, Berkeley, CA, janvier 1994.

[Man94] U. Manber. Finding similar files in a large file system. In *Proceedings of the USENIX Winter 1994 Technical Conference*, pages 1–10, San Fransisco, CA, USA, décembre 17-21 1994.

[MBH+94] Timothy Mann, Andrew Birrell, Andy Hisgen, Charles Jerian, and Garret Swart. A coherent distributed file cache with directory write-behind. *ACM Transactions on Computer Systems*, 12(2) :123–164, 1994.

[MCM01] Athicha Muthitacharoen, Benjie Chen, and David Mazières. A low-bandwidth network file system. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP '01)*, Chateau Lake Louise, Banff, Canada, octobre 2001.

[Mic02] Microsoft Corporation. DFS introduction : Building a single, hierarchical view of multiple file servers and file server shares on a network, 2002. `http://www.microsoft.com/ntserver/techresources/fileprint/dfs/dfssummary.asp`.

[Mog94] Jeffrey C. Mogul. Recovery in spritely NFS. *Computing Systems*, 7(2) :201–262, 1994.

[NR94] B. Clifford Neuman and Santosh Rao. The Prospero Resource Manager : A scalable framework for processor allocation in distributed systems. *Concurrency : Practice and Experience*, 6(4) :339–355, 1994.

[PJS+94] Brian Pawlowski, Chet Juszczak, Peter Staubach, Carl Smith, Diane Lebel, and Dave Hitz. NFS version 3 : Design and implementation. In *USENIX Summer*, pages 137–152, 1994.

[PLL97] Roberto De Prisco, Butler W. Lampson, and Nancy A. Lynch. Revisiting the paxos algorithm. In *Workshop on Distributed Algorithms*, pages 111–125, 1997.

[PPT+92] Rob Pike, Dave Presotto, Ken Thompson, Howard Trickey, and Phil Winterbottom. The use of name spaces in plan 9. In *Operating Systems Review (reprinted from Proceedings of the 5th ACM SIGOPS European Workshop)*, volume Vol. 27, Number 2, pages 72–76, Mont Saint-Michel, 1992.

[PSB+00] Brian Pawlowski, Spencer Shepler, Carl Beame, Brent Callaghan, Michael Eisler, David Noveck, David Robinson, and Robert Thurlow. Nfs version 4 protocol specification. Technical report, IETF, 2000.

[Qui91] Sean Quinlan. A cached WORM file system. *Software - Practice and Experience*, 21(12) :pages 1289–1299, 1991.

[Rab81] M. O. Rabin. *Fingerprinting by Random Polynomials*. Center for Research in Computing Technology, Harvard University, 1981.

[RAS96] Edward W. Felten Robert A. Shillner. Simplifying distributed file systems using a shared logical disk. Technical Report TR-524-96, Princeton University CS Department, 1996.

[Rei00] Hans Reiser. Reiser fs v3 white paper. Technical report, NameSys, 2000.

[RG96] E. Riedel and G. Gibson. Understanding customer dissatisfaction with underutilized distributed file servers. In *5th NASA Goddard Space Flight Center Conference on Mass Storage Systems and Technologies*, septembre 1996.

[RO90] Mendel Rosenblum and John K. Ousterhout. The LFS storage manager. In *Summer'90 USENIX Technical Conference*, Anaheim, California, juin 1990.

[RP93] Herman C. Rao and Larry L. Peterson. Accessing files in an internet : The Jade File System. *IEEE Transactions on Software Engineering*, 19(6) :29, juin 1993.

[Sat81] M. Satyanarayanan. A study of file sizes and functionnal lifetimes. In *8th proceedings on Operating System Principle*, pages 96–108. ACM, 1981.

[Sch94] Bruce Schneier. *Applied Cryptography*. John Wiley & Sons, 1994.

[SDH+96] Adam Sweeney, Doug Doucette, Wei Hu, Curtis Anderson, Mike Nishimoto, and Geoff Peck. Scalability in the xfs file system. In *Proceedings of the USENIX Annual Technical Conference*, pages 1–14, San Diego, CA, USA, janvier 1996.

[SFH+99] Douglas S. Santry, Michael J. Feeley, Norman C. Hutchinson, Alistair C. Veitch, Ross W. Carton, and Jacob Ofir. Deciding when to forget in the elephant file system. In *Symposium on Operating Systems Principles*, pages 110–123, 1999.

[SK90] M. Satyanarayanan and J. Kistler. Coda : a highly available file system for a distributed workstation environment, 1990.

[SM89] V. Srinivasan and J. C. Mogul. Spritely NFS : experiments with cache-consistency protocols. In *Proceedings of the 12th ACM Symposium on Operating Systems Principles (SOSP)*, volume 23 (5), pages 45–57, 1989.

[SON99] Nader Salehi, Katia Obraczka, and Clifford Neuman. The performance of a reliable, request-response transport protocol. In *Proceedings of the Fourth IEEE Symposium on Computers and Communications*, juillet 1999.

[Ste97] David C. Steere. Exploiting the non-determinism and asynchrony of set iterators to reduce aggregate file I/O latency. In *Proceedings of the the 15th ACM Symposium on Operating Systems Principles (SOSP)*, volume 27, pages 252–263, 1997.

[Tan94] Andrew Tannenbaum. *Systèmes d'exploitation : systèmes centralisés - systèmes distribués*. Dunod, 1994.

[TL00] Douglas Thian and Miron Livny. Bypass : A tool for building split execution systems. In *the Ninth IEEE Symposium on High Performance Distributed Computing (HPDC9)*, pages 79–85, Pittsburgh, Pennsylvania, août 2000.

[TM96] Andrew Tridgell and Paul Mackerras. The rsync algorithm, juin 1996. Joint Computer science techncal Report series.

[TML97] C. Thekkath, T. Mann, and E. Lee. Frangipani : A scalable distributed file system. In *Proceedings of the Sixteenth ACM Symposium on Operating Systems Principles (SOSP)*, pages pp. 224–237, Saint-Malo, France, octobre 5-8 1997.

[Ts'97] Theodore Ts'o. Microsoft "embraces and extends" kerberos v5. Site Web USENIX News `http://www.usenix.org/publications/login/1997-11/embraces.html`, novembre 1997.

[Twe98] Stephen Tweedie. Journaling the Linux ext2fs filesystem. In *LinuxExpo'98*, 1998.

[WAD98] Randolph Y. Wang, Thomas E. Anderson, and Michael D. Dahlin. Experience with a distributed file system implementation with adaptive methods global address-based communication in large-scale parallel machines. Technical Report CSD-98-986, University of California at Berkeley, 26, 1998.