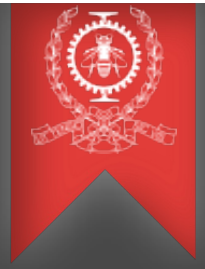


Services de fichiers

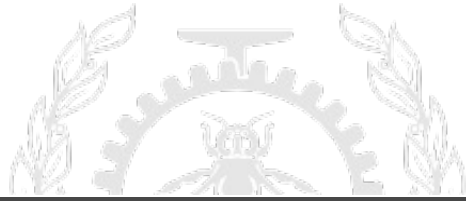
Module 6

INF8480 Systèmes répartis et infonuagique

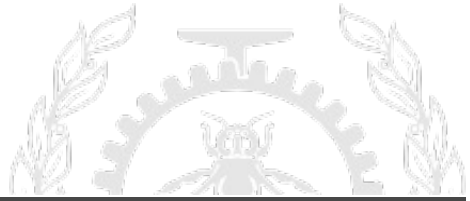
Michel Dagenais

École Polytechnique de Montréal
Département de génie informatique et génie logiciel

Sommaire

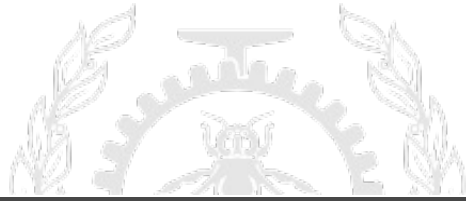


Services de fichiers



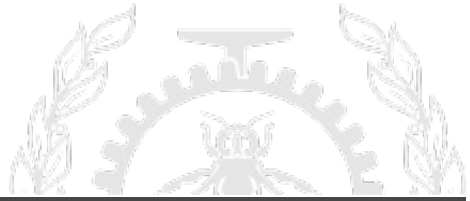
Echanges de fichiers poste-à-poste

- Souvent non structuré à l'avance.
- Ne cherche pas à reproduire la sémantique de l'accès à un fichier local (POSIX).
- Se préoccupe de la mise à l'échelle, de la performance globale et individuelle, et de l'équité des usagers (donner versus recevoir la bande passante) et parfois de l'anonymat.
- Pour des gros fichiers publics (e.g. torrent d'image ISO de logiciels libres), la contestation d'un régime dictatorial, ou l'échange de fichiers dont l'usage est restreint par droits d'auteur.

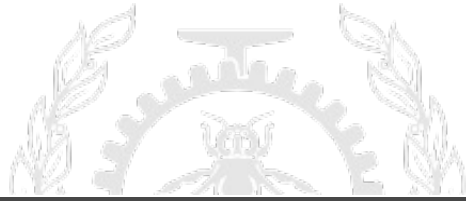


Services de fichiers conventionnels

- Eviter d'avoir à gérer localement le stockage et les copies de sécurité.
- Partager les fichiers au niveau d'un groupe, département, entreprise ou le monde entier.
- Transparent aux applications, reproduit la sémantique d'un accès local.

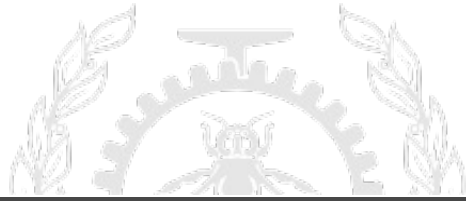


Services de fichiers



NAPSTER

- Serveur centralisé pour l'index (liste des pairs offrant un fichier donné).
- Un client se connecte à l'index pour avoir la liste des autres clients offrant un fichier qu'il recherche. Il offre ses propres fichiers qui sont ajoutés à l'index.
- Le client se connecte à un des autres clients pour obtenir le fichier recherché.
- Il se peut que d'autre clients se connectent au premier pour télécharger un des fichiers qu'il offre.



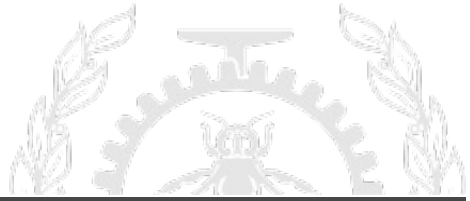
GNUTELLA

- Pas de serveur central, seulement des pairs parmi lesquels certains sont plus puissants et se distinguent (ultra-pairs).
- Graphe bien connecté avec chaque pair connecté à quelques ultra-pairs et les ultra-pairs connectés à des dizaines d'ultra-pairs.
- Chaque pair envoie la liste de ses fichiers aux ultra-pairs connectés (code de dispersement des mots contenus dans les titres).
- Les ultra-pairs font l'union des tables de leurs pairs et envoient cette information à leurs ultra-pairs.
- Une requête est envoyée de préférence à un nœud ayant le fichier, autrement à un ultra-pair qui a annoncé le fichier, et finalement à un ultra-pair à la fois.
- La requête contient l'adresse de l'ultra-pair qui a initié la recherche et le fichier trouvé lui est envoyé directement.

BITTORRENT

- Système réparti poste à poste spécialement conçu pour la propagation de gros fichiers (e.g. image iso Ubuntu, contenu de DVD...).
- Fichier .torrent donne le nom et la longueur du fichier, l'adresse du serveur central de suivi pour ce fichier, et une somme de contrôle.
- Le serveur maintient une liste des pairs qui ont une copie complète ou partielle du fichier.
- Un client obtient du serveur une liste de pairs possédant le fichier. Il les contacte pour obtenir les morceaux en parallèle dans n'importe quel ordre.
- Des statistiques sont maintenues aux 10s sur ce que chaque pair contribue. Chaque pair donne normalement priorité aux autres pairs qui contribuent à l'effort.
- Les morceaux de fichiers les plus rares sont offerts en priorité.

Services de fichiers



Module local d'accès aux fichiers

- Pilote d'interface: disque(s) ou partition de disque accessible sous la forme d'un vecteur de blocs.
- Gestion des blocs: tampons d'entrée-sortie, pré-lecture, écriture asynchrone, ordonnancement des accès.
- Système de fichiers avec inode (identificateur de fichier avec permissions d'accès et liste de blocs directs ou indirects qui le composent) et répertoire (fichier contenant une liste de noms versus identificateurs de fichiers permettant de constituer une structure hiérarchique).
- Table de mount : certains chemins mènent vers d'autres partitions ou disques.
- Cache de dentries : cache de chemins versus identificateur de fichier.

Accès de fichiers par réseau

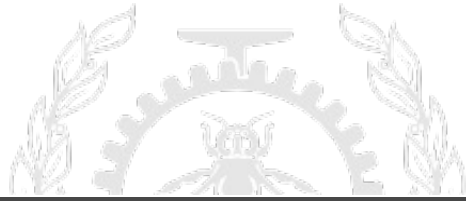
- Transparence d'accès: un programme peut lire un fichier local ou distant sans distinction.
- Transparence de localisation: le nom ne change pas selon l'ordinateur où se trouve le programme qui veut accéder le fichier.
- Transparence de concurrence: les accès concurrents peuvent se faire de la même manière qu'en local.
- Transparence de défectuosité (messages perdus, serveur réinitialisé, réseau partitionné...).
- Transparence de performance (performance similaire même avec un nombre croissant de clients).
- Disponible sur plusieurs plates-formes.
- Transparence même en cas de réplication et de migration.
- Sécurité

Modules d'un service de fichiers réparti

- Propriétés d'un fichier: longueur en octets, date de création, dernière lecture/écriture, décompte de référence, propriétaire, liste de contrôle des accès.
- Service de répertoire: convertir le chemin demandé en identificateur de fichier et vérifier les permissions d'accès. Retourner une clé appropriée.
- Service de fichiers de base: la clé permet de savoir le fichier à accéder, et les droits que possède le détenteur sur ce fichier. Opérations de lecture, écriture sur le fichier.
- Module client qui utilise les services de répertoires et de fichiers et qui utilise des opérations réapplicables (idempotentes) pour tolérer les défaillances (message retransmis). Il peut avoir à se connecter à plus d'un serveur.

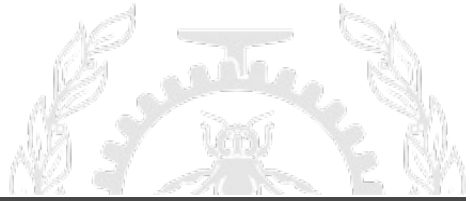
Interface pour le service de répertoire

- FID Lookup(FID dir, String name, Mode accessMode, UID user);
- AddName(FID dir, String name, FID file, UID user) {NameDuplicate};
- UnName(FID dir, String name) {NotFound};
- ReName(FID dir, String oldName, String newName) {NotFound, NameDuplicate};
- StringSequence GetNames(FID dir, String pattern);



Interface pour le service de fichiers

- CharSequence Read(FID file, unsigned pos, unsigned length) {BadPosition};
- Write(FID file, unsigned pos, CharSequence data) {BadPosition};
- FID Create();
- Truncate(FID file, unsigned length);
- Delete(FID file);
- AttributeSequence GetAttributes(FID file);
- SetAttributes(FID file, AttributeSequence attr);



Considérations

- Grouper les fichiers par sous-arbre ou autrement pour les répartir sur plus d'un serveur.
- La création de fichiers orphelins s'ils ne sont pas mis dans un répertoire (opération combinée Create/AddName).
- Le FID peut contenir: Identificateur de groupe (32 bits adresse IP + 16 bits temps), 32 bits numéro de fichier, 32 bits nombre aléatoire + 5 bits permission encryptés, 5 bits de permission non encryptés. Il identifie le fichier et les permissions mais est très difficile à contrefaire.
- Serveur de localisation de groupe: numéro de groupe versus ordinateur et numéro de port.
- Le serveur de fichiers utilise une cache comme à l'habitude. Les clients peuvent aussi avoir une cache mais cela pose un problème de cohérence sérieux.

Sun NFS

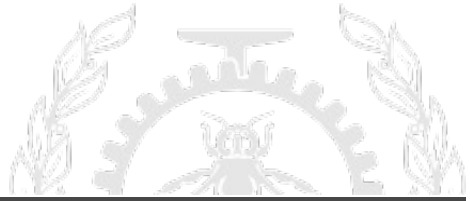
- Introduit en 1985, définition placée dans le domaine public en 1989. Un sous-arbre du serveur peut être monté localement.
- Transparence d'accès, et de localisation, migration, et réplication en lecture avec un peu d'effort via `autofs`.
- Le serveur peut se réinitialiser sans affecter l'accès du client autrement qu'en retardant son accès (stateless).
- Sur un réseau 10Mbits/s peu chargé, la performance pour lire des blocs en accès aléatoire est comparable à un accès local (accès disque de 10ms pour un bloc de 8K).
- Pas de support pour la réplication lecture/écriture, ce qui limite l'adaptabilité à un grand nombre de clients.
- Support mitigé pour le verrouillage des fichiers à des fins de synchronisation.

Implantation

- Le noyau maintient une couche VFS (Virtual File System) pour savoir quel sous-arbre vient de quel système de fichier.
- Le client NFS est un module du noyau qui est utilisé lors d'opérations sur les fichiers qui sont sur des serveurs NFS. Le processus `biod` s'occupe de pré-lecture et d'écriture asynchrone de blocs.
- Monté en dur, les accès NFS sont bloquants, autrement une expiration de délai cause le retour avec un code d'erreur des opérations sur un fichier. Certains programmes supposent que les accès aux fichiers ne peuvent causer d'erreur!
- La traduction du chemin doit se faire une composante à la fois afin de vérifier si on arrive à une sous-branche montée ailleurs.

Implantation

- Automount/autofs permet de monter les sous-arbres à la demande et de les démonter lorsqu'ils ne sont plus utilisés.
- Le serveur roule `portmap`, `mountd`, et `nfsd` qui peuvent être des processus en mode usager.
- Lors d'un accès pour un fichier, la date de dernière modification est vérifiée et utilisée pour invalider la cache du client. Autrement, on ne force une vérification de la date de dernier accès qu'une fois par 3 secondes pour les fichiers et 30 secondes pour les répertoires.
- Les opérations réseau utilisent des appels Sun RPC, avec optionnellement de l'encryption.
- Les écritures sont synchrones.



Andrew File System (AFS)

- Version 1 en 1986, et 2 en 1989.
- Développé à l'université CMU, prévu pour des milliers de clients et des dizaines de serveurs.
- Lorsqu'un fichier est ouvert, il est copié sur le disque local du client (par morceaux de 64k ou le fichier complet) s'il ne s'y trouve pas déjà.
- Lors de la fermeture il est mis à jour sur le serveur.
- Efficace pour de nombreux petits fichiers qui sont soit lecture-seulement, soit modifiables par un seul usager, ce qui est typique d'un environnement de laboratoire informatique.
- Distribué commercialement au début puis relâché comme logiciel libre.

Implantation

- Les appels open et close sont interceptés pour effectuer les accès requis au serveur.
- Le client compte sur le serveur pour l'avertir de toute mise à jour sur les fichiers qu'il contient dans sa cache. Il revérifie après une réinitialisation, ou lorsqu'il ne reçoit rien du serveur pour quelques minutes.
- Conserve un état mais plus efficace, équivaut à 40% de la charge du même service par NFS.
- Le client a la garantie que lors d'un open il a la version la plus récente (à quelques minutes près), et que lors d'un close la mise à jour est propagée au serveur.
- Volumes (groupes de fichiers) lecture-seulement peuvent être répliqués sur plusieurs serveurs.
- Base de donnée de localisation de volumes répliquée sur chaque serveur.

CODA

- Développé à CMU à partir de 1990, première version vers 1995.
- Permettre la réplication de volumes lecture-écriture, et l'opération déconnectée.
- Semblable à AFS sauf pour open lire d'un seul serveur et pour close envoyer la modification à tous les serveurs.
- Une liste permet de spécifier les fichiers dont une copie locale devrait toujours exister. Au moment de la reconnexion, ou après une panne de réseau, la cache est revalidée.
- Lorsqu'un serveur redevient accessible, il faut vérifier ses vecteurs de numéros de version pour les fichiers modifiés. S'il manque des mises à jour, elles lui sont propagées. Si des mises à jour différentes ont été reçues de part et d'autre, il y a un conflit à résoudre.

Common Internet File System (CIFS)

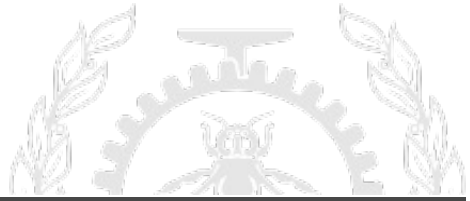
- Server Message Block (SMB) par-dessus NetBIOS, développé chez IBM pour DOS.
- Intégré à Lan-Manager (OS/2, Windows for Workgroups)
- En 1996, renommé CIFS.
- OpLock Exclusive, aucune autre copie sur un autre client, le détenteur n'a pas à propager chaque modification.
- OpLock Level 2, accès partagé, peut faire les lectures en cache mais doit propager toute modification.
- OpLock Batch, permission de retarder les close et de les annuler en cas de open qui subvient peu de temps après.
- OpLock Break, révocation de OpLock car le fichier vient d'être modifié par un autre client.

Global File System GFS

- Commencé à l'Université du Minnesota, 1995, Sistina Software, 1999, Red Hat 2003.
- Plusieurs serveurs peuvent se connecter au même SAN (disques réseau) par iSCSI, FibreChannel ou AoE (ATA over Ethernet), offrant une certaine redondance.
- Aussi possible de se connecter sur un disque répliqué par logiciel en Linux: GFS2 sur DRDB sur DM.
- Gestionnaire de verrou (Distributed Lock Manager DLM) assure que les accès sur des connexions différentes de SAN ou des réplicats de disque différents ne sont pas en conflit.
- Lorsqu'un nœud est considéré défaillant, il est désactivé par matériel (relais) pour éviter qu'il ne cause des conflits.

Gluster File System

- La compagnie Gluster (GNU Cluster) a été fondée en 2005 pour offrir un système de fichier libre et performant.
- Acheté en 2011 par Red Hat.
- Fait l'aggrégation de systèmes de fichiers natifs (e.g. xfs, btrfs) et les exporte sous plusieurs protocoles (NFS, CIFS, HTTP. . .).
- Gère la distribution et la redondance.
- Un des systèmes les plus utilisés avec OpenStack mais qui sera concurrencé par Ceph.



Le système de fichiers Lustre

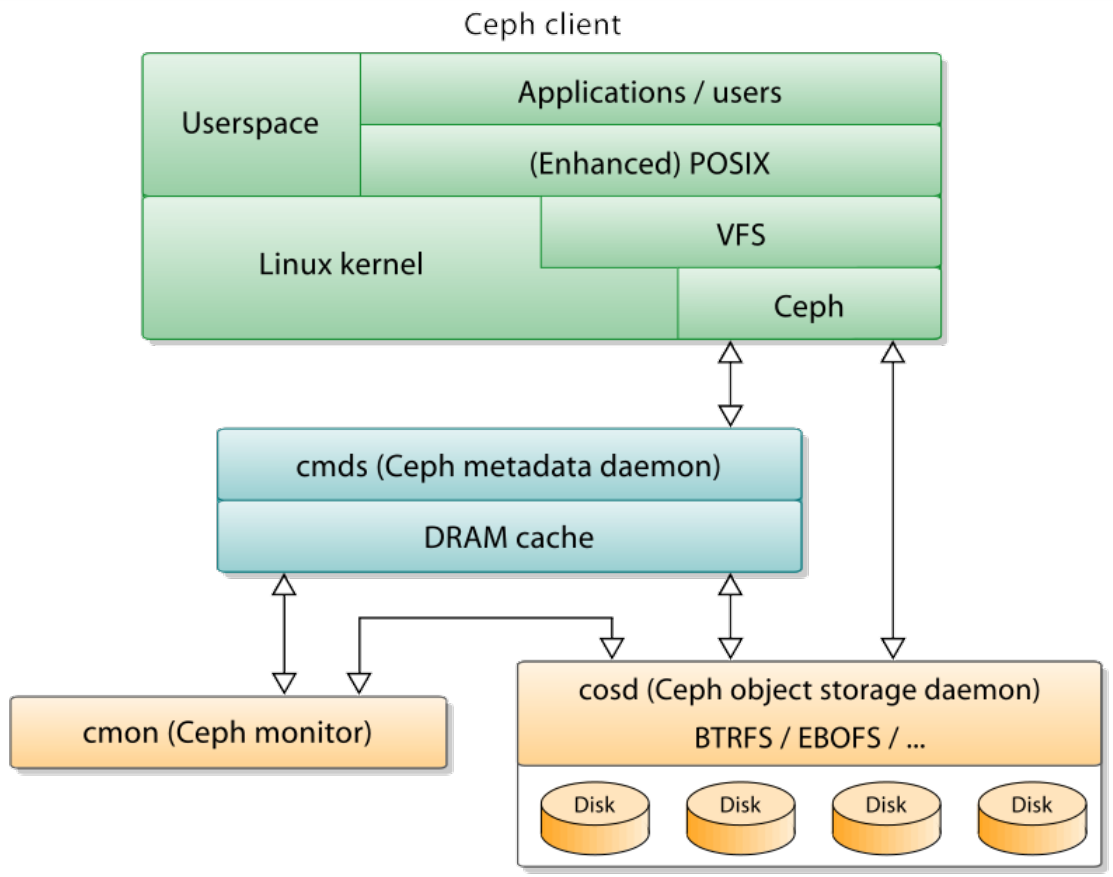
- Compagnie démarrée à CMU, achetée par SUN puis SUN par Oracle. Produit discontinué chez Oracle mais qui se poursuit en logiciel libre. Utilisé par 15 des 30 au sommet du Top500.
- Serveur de metadonnées, (Meta Data Server MDS); répertoires, liste des objets de stockage pour le fichier, propriétés des fichiers.
- Plusieurs serveurs d'objets de stockage (Object Storage Server) avec chacun quelques groupes de disques, e.g. 2 à 8, (Object Storage Target). Un fichier peut avoir des morceaux sur plusieurs OSS.
- Gestion des verrous (Distributed Lock Manager) lecture et écriture par sections de fichiers.
- Redondance Active/Passive pour MDS et Active/Active pour OSS.

Google File System GFS

- Serveur primaire responsable des méta-données (attribut, emplacement des morceaux de fichiers avec réplicats).
- Journal des opérations du serveur primaire vers un autre serveur pour récupération en cas de panne.
- Serveurs (centaines) de morceaux (64MB) de fichiers, ChunkServers.
- Lors d'une modification à un fichier, un ChunkServer est identifié comme primaire pour chaque morceau.
- Le client envoie la modification à chaque réplicat puis confirme la modification au ChunkServer primaire qui détermine l'ordre, écrit, puis demande aux réplicats d'écrire dans le même ordre et confirme au client.

CEPH

- Nouveau service de blocs, fichiers, ou objets de stockage prévu pour la mise à l'échelle et l'infonuagique.
- Première version stable en 2016.



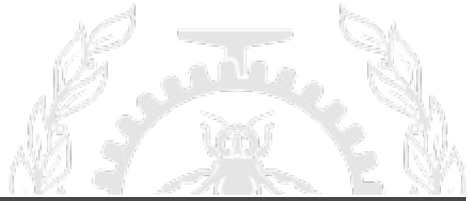
CEPH

Ceph-mon cluster monitor, vérifie l'état des serveurs de la grappe pour noter s'ils sont actifs ou en panne (au moins 2 pour tolérance aux pannes). Maintien une carte des serveurs avec consensus par Paxos.

Ceph-mds serveur de métadonnées (répertoires et inodes). Chacun couvre un sous-espace ajusté dynamiquement avec des recoupements pour fins de redondance. Les métadonnées sont stockées dans des fichiers.

Ceph-osd serveur pour le contenu des fichiers eux-mêmes, usuellement sur des partitions `xfs` ou `btrfs`. (Au moins 2, par défaut 3, pour réplication).

Ceph-rgw serveur RESTful pour offrir une interface compatible avec Amazon S3 et OpenStack Swift.

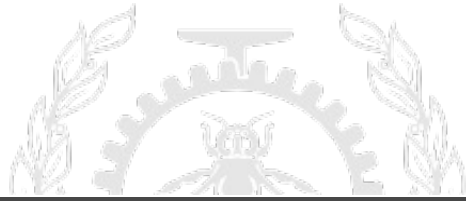


CEPH

- Un fichier est identifié par un inode number (INO) fourni par le serveur de métadonnées et décomposé selon sa taille en plusieurs objets (ONO) qui ont chacun un OID.
- Une fonction de hachage simple associe le OID à un groupe de placement (PGID). Le groupe s'occupe de la réplication.
- Le placement d'un groupe (PGID) sur des serveurs de fichiers (ceph-osd) est déterminé par un algorithme : Controlled Replication Under Scalable Hashing (CRUSH).
- Emphase sur la performance (morceaux de fichiers sur plusieurs serveurs), la mise à l'échelle (serveurs de métadonnées séparés, CRUSH) et la tolérance aux pannes.

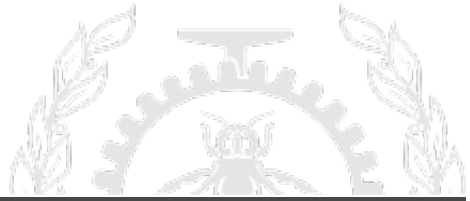
OrangeFS

- Nouvelle génération de PVFS (*Parallel Virtual File System*), qui était développé à Clemson University et ANL pour les grappes MPI.
- Services de métadonnées et de fichiers séparés.
- Les fichiers sont des objets avec des paires clé/valeur et une séquence d'octets (bytestream).
- Les clients peuvent accéder le service directement (Linux, Windows, Mac) ou par WebDAV, S3, Hadoop...
- La répartition des objets sur les serveurs est la partie importante et peut être décidée automatiquement ou spécifiée par l'utilisateur.



HDFS

- Stockage de fichiers (ne satisfait pas POSIX) afin de répartir de gros fichiers entre les nœuds d'une grappe.
- Au moins un name-node pour les métadonnées et une grappe de data-nodes pour la distribution et réplication des blocs de fichiers.
- Les nœuds de données peuvent communiquer entre eux pour équilibrer l'espace occupé et la charge.
- Optimisé pour de très gros fichiers, surtout en lecture, sur lesquels on veut faire du traitement réparti (HADOOP), ce qui permet de conserver les données près de là où elles seront utillisées.



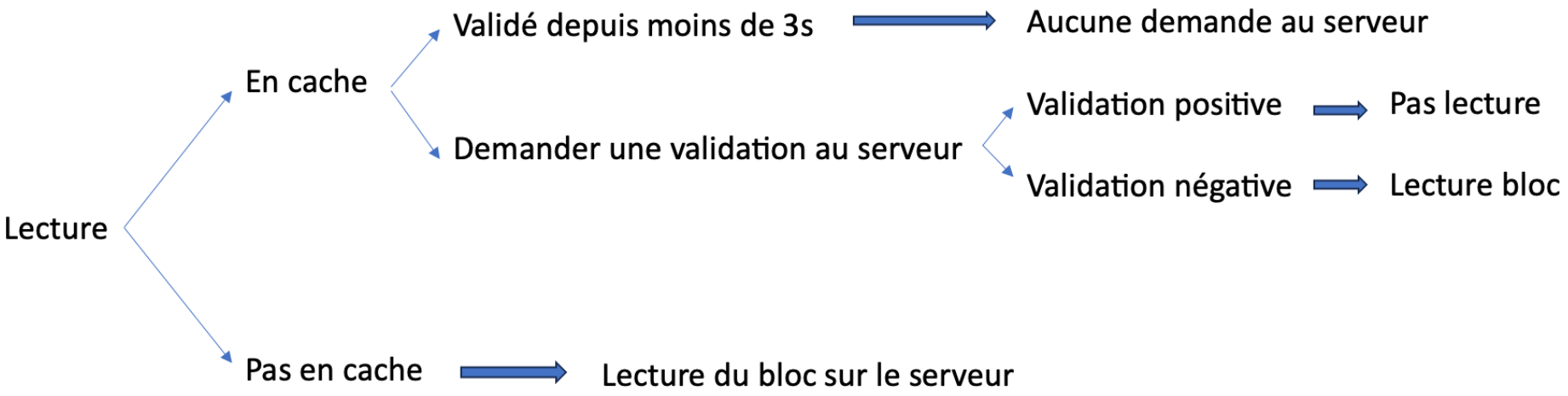
Fichiers – Ex. 6.1.

Un serveur NFS sert de nombreux clients. Les processus sur chaque client effectuent en moyenne 2 écritures et 8 lectures par seconde sur des blocs de fichiers venant de ce serveur. Les blocs accédés en lecture se trouvent en cache sur le client dans 85% des cas. Parmi les blocs en cache, 60% ont été validés depuis moins de 3 secondes. Les autres blocs en cache demandent une validation auprès du serveur. Parmi ces blocs qui demandent une validation, 40% ont été modifiés et nécessitent une lecture sur le serveur en plus, alors que 60% sont valides. L'écriture d'un bloc sur le serveur prend 20ms de disque. La lecture d'un bloc du serveur prend 15ms de disque dans 30% des cas, et est servie à partir du cache d'entrée-sortie en temps négligeable dans 70% des cas. Une validation d'un bloc du serveur prend 15ms de disque dans 10% des cas, et est servie à partir du cache d'entrée-sortie en temps négligeable dans 90% des cas. Quel est le nombre de clients maximal que peut soutenir le serveur sans être saturé, s'il contient 16 disques, que les cœurs de CPU ne sont pas un facteur significatif, et que les requêtes sont réparties uniformément entre les disques?

Fichiers – Ex. 6.1. – Solution

Écriture : Lorsqu'un processus sur un client effectue une écriture sur un bloc venant d'un serveur, ceci se traduit automatiquement en une écriture sur le serveur.

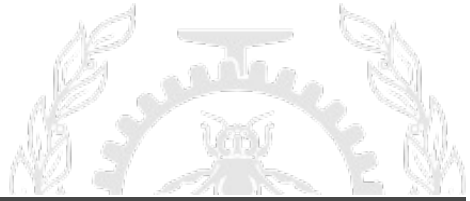
Lecture : Lorsqu'un processus sur un client effectue **une lecture** d'un bloc venant d'un serveur, cela se décompose comme suit:



Fichiers – Ex. 6.1. – Solution

ii. Les blocs accédés en lecture se trouvent en cache sur le client dans 85% des cas. Parmi les blocs en cache, 60% ont été validés depuis moins de 3 secondes. Les autres blocs en cache demandent une validation auprès du serveur.

- En cache et validé depuis moins de 3s: 0.85 x 0.6 = 0.51, rien à faire
- Pas en cache, 0.15, demande lecture

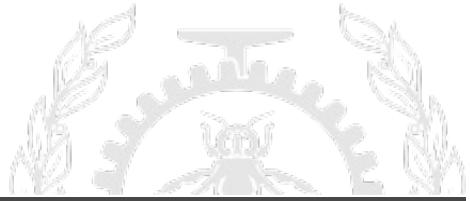


Fichiers – Ex. 6.1. – Solution

iii. Parmi ces blocs qui demandent une validation, 40% ont été modifiés et nécessitent une lecture sur le serveur en plus, alors que 60% sont valides.

- En cache, **demande validation, validation positive**: 0.85 x 0.4 x 0.6 = 0.204, demande de validation, **pas lecture**

- En cache, **demande validation**, **validation négative**: 0.85 x 0.4 x 0.4 = 0.136, demande validation et **lecture**



Fichiers – Ex. 6.1. – Solution

Lecture : Lorsqu'un processus sur un client effectue **une lecture** d'un bloc venant d'un serveur, cela se décompose comme suit:

- **Total demande validation** = (demande validation, validation positive**)** + (demande validation, validation négative) = 0.204 + 0.136 = 0.34 validation par lecture

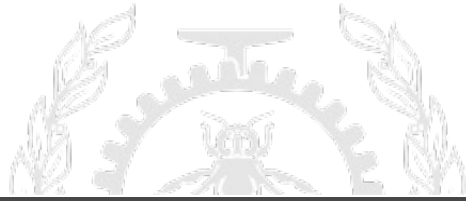
- **Total demande lecture** = (demande validation, validation négative) + (Pas en cache, demande lecture) = 0.136 + 0.15 = 0.286 lecture bloc sur disque-serveur par de lecture du client

Fichiers – Ex. 6.1. – Solution

Temps moyenne par écriture

iv. L'écriture d'un bloc sur le serveur prend 20ms de disque.

Une écriture prend 20ms.



Fichiers – Ex. 6.1. – Solution

Temps moyenne par lecture

v. La lecture d'un bloc du serveur prend 15ms de disque dans 30% des cas, et est servie à partir du cache d'entrée-sortie en temps négligeable dans 70% des cas

- La lecture d'un bloc sur le serveur prend 0.3 x 15ms = 4.5ms de disque en moyenne.

vi. Une validation d'un bloc du serveur prend 15ms de disque dans 10% des cas, et est servie à partir du cache d'entrée-sortie en temps négligeable dans 90% des cas

- Une validation prend 0.1 x 15ms = 1.5ms de disque en moyenne.

Fichiers – Ex. 6.1. – Solution

Chaque client, à chaque seconde, demande donc au serveur :

- 2 écritures,
- 8 lectures x 0.34 demande de validations = 2.72 demandes de validations,
- 8 lectures x 0.286 = 2.288 demandes de lectures sur disque-serveur

Temps moyenne par client

- Un client prend donc par seconde en moyenne 2 écritures x 20ms + 2.72 validations x 1.5ms + 2.286 lectures x 4.5ms = 54.367ms.

Total clients

Avec 16 disques, il peut donc soutenir jusqu'à 16 x 1000ms/s / 54.367ms/client = 294 clients.

Conclusion : services de fichiers pour le nuage

- OpenStack utilise principalement NFS, CIFS, GlusterFS, HDFS et bientôt Ceph.
- NFS et CIFS sont les solutions classiques, connues, faciles à installer.
- GlusterFS est la solution la plus établie pour les grandes installations qui nécessitent une certaine mise à l'échelle. Ceph sera un compétiteur sérieux, plus avancé technologiquement.
- HDFS peut être intéressant pour des applications de type Hadoop.
- Plusieurs systèmes de fichiers propriétaires sont aussi mis de l'avant par leurs fournisseurs respectifs (e.g., VMWare).