


Introduction aux systèmes de fichiers parallèles

Philippe.Wautelet@idris.fr

CNRS - ***IDRIS***

Ecole « Optimisation »
Maison de la Simulation / 7 au 11 octobre 2013

Sommaire

Architecture des machines parallèles

Machine parallèle

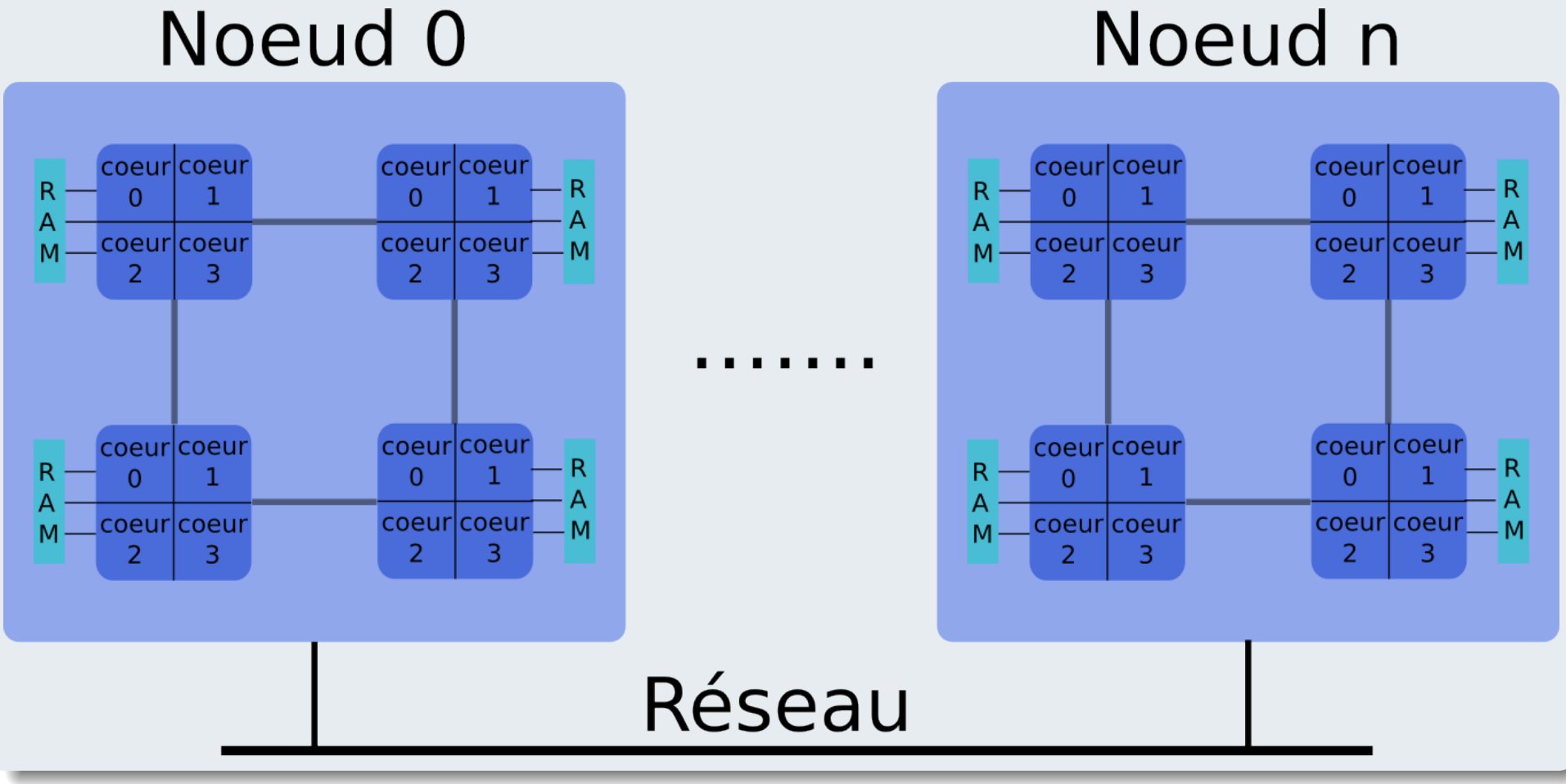
Un calculateur parallèle est constitué de :

- d'un ensemble de cœurs de calcul ayant accès à une mémoire locale et regroupés dans des nœuds ;
- d'un réseau d'interconnexion rapide et performant ;
- d'un système de stockage lui aussi performant.

Chaque nœud contient un certain nombre de cœurs de calcul éventuellement assistés d'accélérateurs (GPGPU, FPGA...).

- Tous les cœurs à l'intérieur d'un nœud ont accès à la mémoire (architecture à mémoire partagée).
- Par contre, généralement, les cœurs d'un nœud n'ont pas accès à la mémoire d'un autre nœud (architecture à mémoire distribuée).
- Il existe des machines dites à mémoire partagée dans lesquelles tous les cœurs peuvent accéder à la mémoire de n'importe quel nœud. Dans ce type de machine, l'accès à la mémoire est non-uniforme (NUMA) car selon où se trouve la mémoire par rapport à un cœur, les performances (débit et latence) seront variables. Ces performances peuvent également varier à l'intérieur d'un nœud, mais de façon beaucoup moins prononcée.

Machine type



Machines parallèles : exemples

Tianhe-2 (National University of Defense Technology in Changsha, Chine) (#1 top 500 6/2013)

- TH-IVB-FEP Cluster sous Kylin Linux
- 54,9 Pflop/s crête, 33,9 Pflop/s sur benchmark LINPACK
- 3.120.000 cœurs et 1.375 TiB de mémoire vive (16.000 nœuds contenant 2 Xeon Ivy Bridge à 12 cœurs et 2,2 GHz avec 64 Gio de mémoire et 3 Xeon Phi 31S1P1 avec 8 Gio de mémoire)
- Réseau TH Express-2
- Système de fichier H^2FS de 12,4 Pio
- 17,6 MW (24 MW avec le refroidissement)

Titan (ORNL, USA) (#2 top 500 6/2013)

- Cray XK7 sous linux
- 27,1 Pflop/s crête, 17,6 Pflop/s sur benchmark LINPACK
- 560.640 cœurs et 710 TiB de mémoire vive (18.688 nœuds contenant 1 AMD Opteron 6274 à 16 cœurs à 2,2 GHz avec 32 Gio de mémoire et 1 Nvidia Tesla K20X avec 6 Gio de mémoire)
- Réseau GEMINI (Cray)
- Système de fichiers Lustre, 40 Pio de disques (1,4 Tio/s crête)
- 8,2 MW

Qu'est-ce qu'un système de fichiers ?

Qu'est-ce qu'un système de fichiers ?

Rôles principaux

Un système de fichiers a deux fonctions principales :

- Organiser et maintenir l'espace de noms des fichiers
- Stocker le contenu des fichiers et leurs attributs

Données

Elles correspondent aux contenus proprement dit des fichiers.

Métadonnées

Les métadonnées sont un ensemble d'informations sur le fichier. Elles contiennent par exemple :

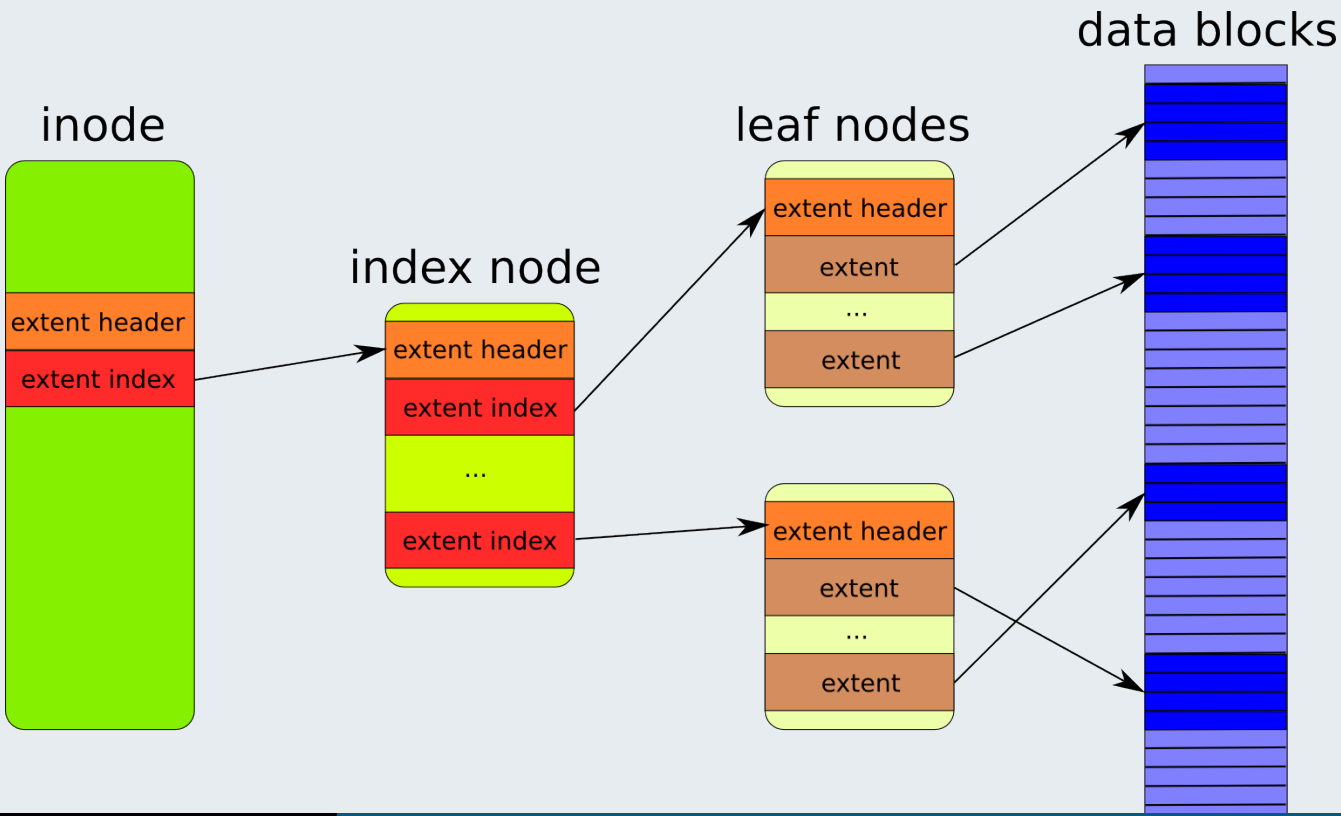
- La position des données sur le disque
- La taille du fichier
- Les dates de création, dernière modification et dernier accès
- Le propriétaire (UID et GID) et les droits d'accès
- ...

Systèmes de fichiers séquentiels

Définition

- Un système de fichiers séquentiel local est un système de fichiers qui ne peut être accédé directement qu'en local.
- Seul 1 client peut y accéder (le système d'exploitation de la machine).
- Généralement, il n'y a pas de parallélisme (un seul accès simultané à la fois).

Exemple : structure d'un *gros* fichier sur système de fichiers ext4



Systèmes de fichiers parallèles

Définitions

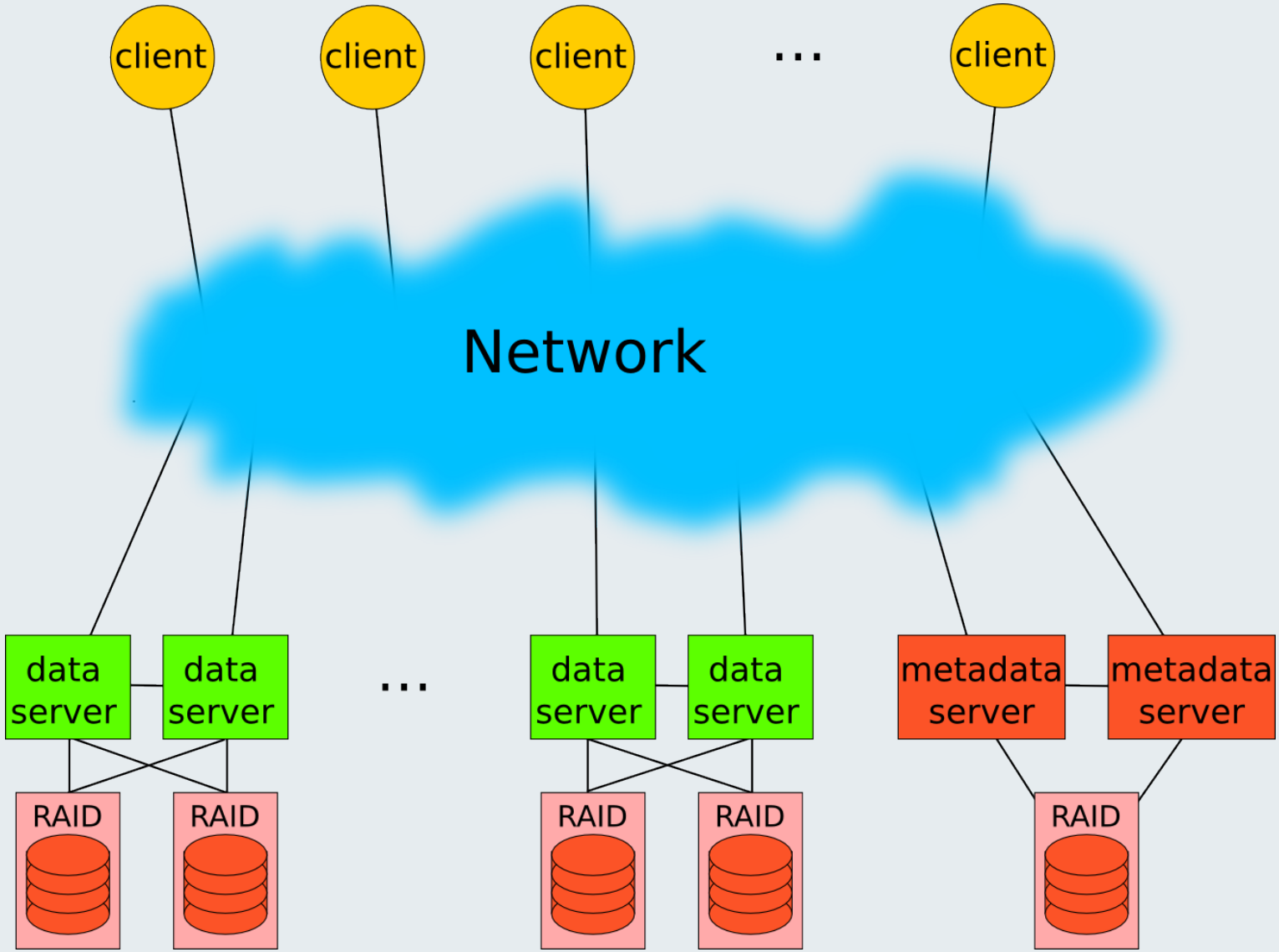
Un système de fichiers parallèle a pour but de permettre l'accès simultané à un système de fichiers à plusieurs clients. Ce qui le distingue d'un *simple* système de fichiers partagé est le parallélisme au niveau :

- des clients. Plusieurs clients peuvent lire et écrire simultanément et non pas chacun à son tour.
- de la répartition des données. Un client l'utilisant profitera de bonnes performances si les données sont réparties sur plusieurs serveurs de données.

Ce parallélisme se fait de façon transparente pour le client qui voit le système de fichiers comme s'il était local.

En plus des fonctions d'un système de fichier local, un système de fichiers parallèle doit gérer efficacement les éventuels conflits entre les différents clients. L'approche privilégiée consiste à utiliser des verrous pour limiter/contrôler les accès simultanés à un fichier ou répertoire donné.

Architecture générale

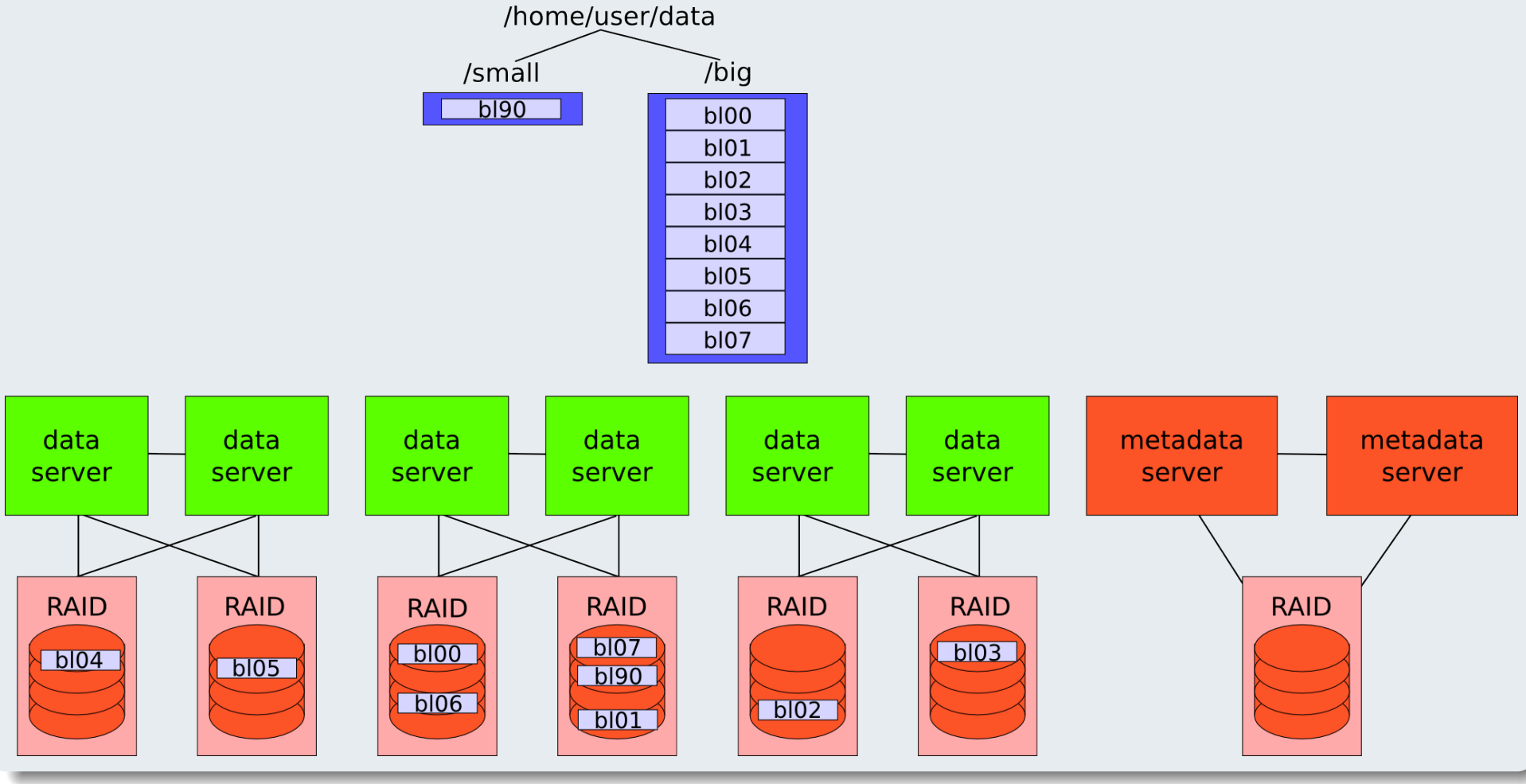


Architecture générale

Un système de fichiers parallèle est constitué de :

- des clients qui vont lire ou écrire des données sur le système de fichiers ;
- un ou plusieurs serveurs de métadonnées. Ceux-ci gèrent ces dernières et le placement des données sur les disques, ainsi que les verrous de contrôle des accès (par exemple pour éviter que 2 clients modifient la même partie d'un fichier en même temps) ;
- un certain nombre de serveurs de données. Ceux-ci stockent l'ensemble des données. Pour certains systèmes de fichiers parallèles, les données et les métadonnées peuvent être manipulées par les mêmes serveurs ;
- et d'un ou plusieurs réseaux (dédiés ou pas) pour interconnecter tous ces constituants.

Striping des fichiers



Découpage des fichiers

Un fichier sera généralement découpés en morceaux de taille fixe (appelés *stripes* ou *chunks*) et partagés entre les différents serveurs. Une lecture ou écriture de ce fichier se fera donc en parallèle sur les différents serveurs de fichiers et le débit de lecture ou écriture sera donc la somme des débits obtenus sur tous ces serveurs.

Intégrité des données et redondance

Le système de fichiers parallèle doit également garantir l'intégrité des données, ainsi que la redondance du système. Cela peut se faire à plusieurs niveaux :

- Chaque serveur de données et de métadonnées gère plusieurs disques qui utilisent un système de fichiers local avec un support RAID garantissant l'intégrité des données en cas de perte d'un ou plusieurs disques.
- Les données peuvent être répliquées à plusieurs endroits différents.
- Un serveur de données ou de métadonnées peut être capable de gérer les disques d'un autre serveur et de prendre la main sur celui-ci en cas de défaillance.
- Un cheminement alternatif pour les données peut exister (deux réseaux différents par exemple).

Verrous (*locks*) et accès concurrents : rôle

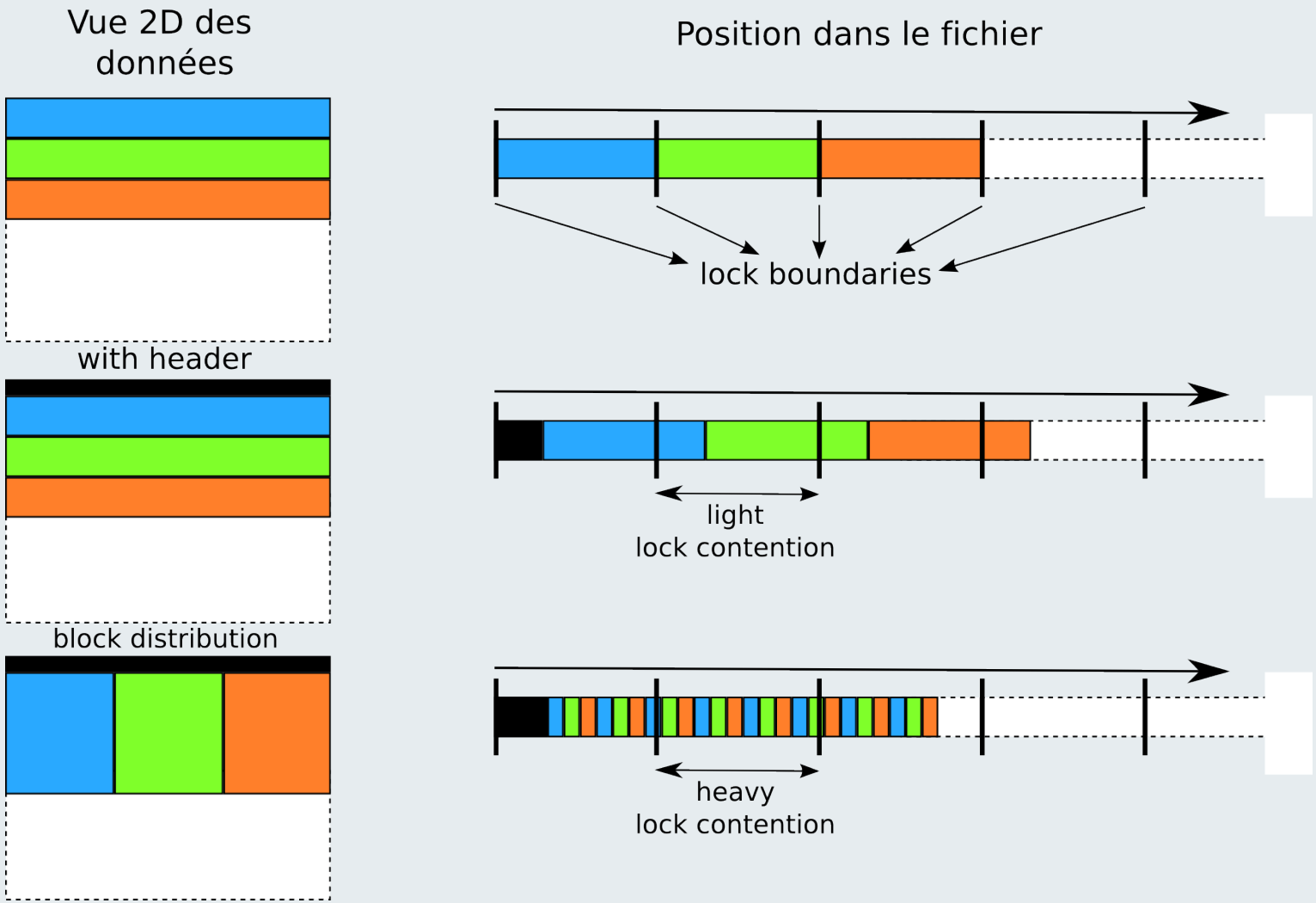
Pour garantir la cohérence des données et des métadonnées, les systèmes de fichiers parallèles utilisent généralement des verrous (*locks*) qui limitent les accès simultanés à ces informations. Cela permet, entre-autres, de garantir l'atomicité des opérations de lecture/écriture. Par exemple, un processus écrit un bloc de données et un autre veut le lire en même temps. L'utilisation d'un verrou garantira que le lecteur lira le bloc de données soit tel qu'il était avant la modification, soit après (selon qu'il obtient le verrou avant ou après l'écrivain), mais jamais un mélange des 2.

Verrous (*locks*) et accès concurrents : fonctionnement

Selon le système de fichiers, les verrous sur les données se font au-niveau d'un fichier ou d'un *stripe*. Ils sont alignés sur certaines frontières (par exemple taille des pages mémoire pour Lustre et taille de bloc du système de fichiers pour GPFS).
Il y a 2 grands types de verrous :

- Les verrous exclusifs pour les écritures limitant l'accès à un intervalle à un seul client.
- Les verrous partagés pour les accès à un intervalle en lecture à n'importe quel nombre de lecteurs et empêchant les modifications/écritures simultanées.

Verrous (*locks*) et accès concurrents : fonctionnement



Les données représentent un tableau bidimensionnel dans l'application (données contiguës selon les lignes). Chaque couleur correspond à un processus/client.

Caches

Un cache est une copie locale proche de celui qui l'utilise. Son but est d'accélérer les performances. Leur influence peut être très importante.

Dans un système de fichiers parallèle, les caches se trouvent principalement au niveau :

- des serveurs de données. Les caches se trouvent avant les disques dans la mémoire vive (plus rapide) et peuvent être en lecture et en écriture (dans ce cas, la mémoire doit être alimentée par des batteries en cas de coupure électrique) ;
- des clients. La cohérence des données entre les différents clients doit être assurée. Cela se fait via les verrous. Par exemple, un client ayant les droits d'écriture devra *flusher* ses caches vers les serveurs de données si son verrou est révoqué. Autre cas, si des clients sont en lecture et qu'un autre se met à écrire dans la même zone, les caches en lecture devront être invalidés (càd que les données qui s'y trouvent ne pourront plus être utilisées) avant de pouvoir commencer à lire les données nouvellement écrites à partir des serveurs de données.

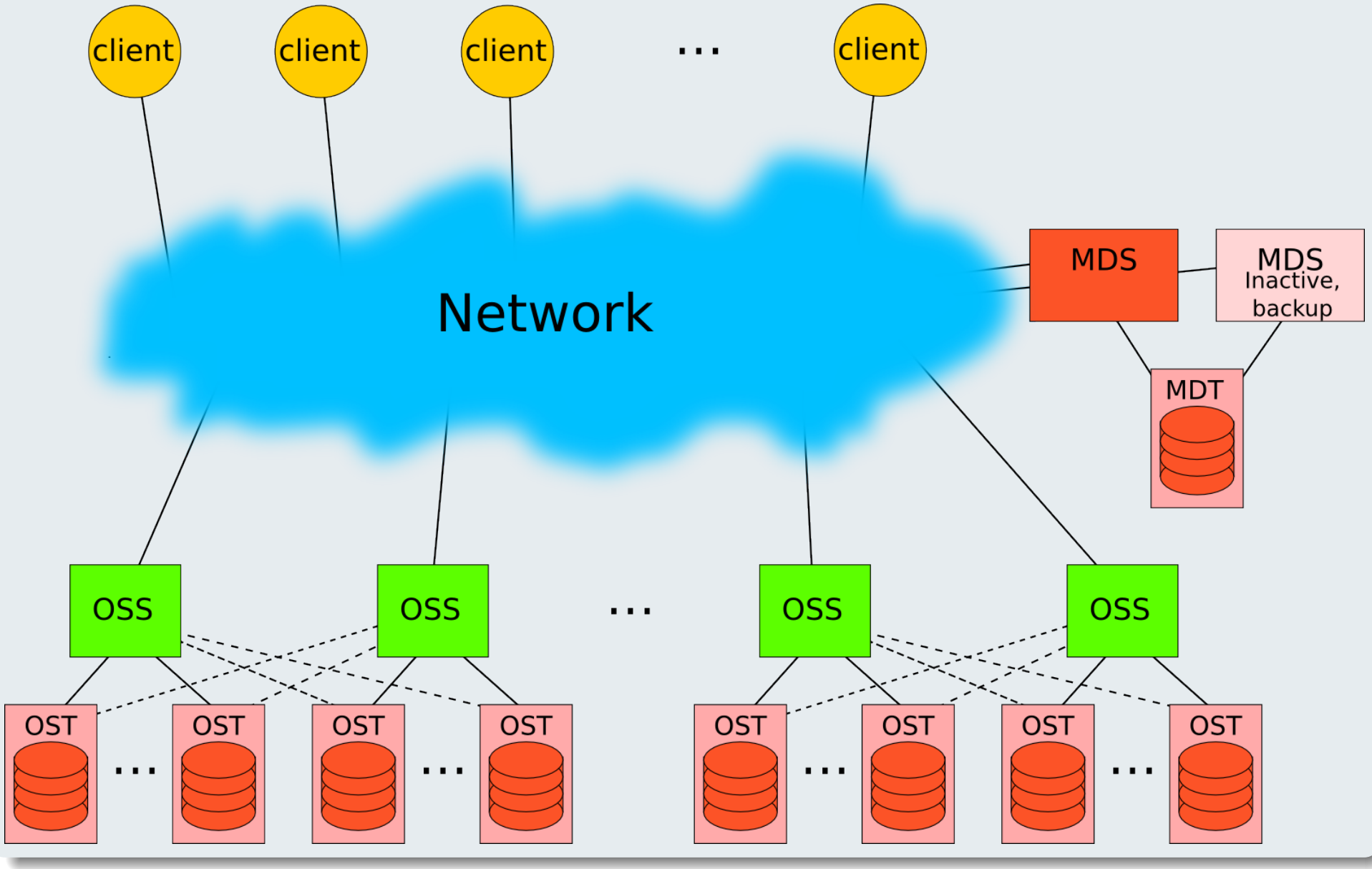
Chaque système de fichiers parallèle a sa façon de gérer les caches.

Principaux systèmes de fichiers parallèles

Les systèmes de fichiers parallèles les plus couramment utilisés dans les supercalculateurs sont :

- Lustre
- GPFS
- PVFS2/OrangeFS
- PanFS

Schéma


client
client
client
...
client
Network
MDS
MDS
Inactive,
backup
MDT
OSS
OSS
...
OSS
OSS
OST
OST
OST
OST
OST
OST
OST
OST

Architecture de Lustre

Lustre est un système de fichiers parallèle sous licence *Open Source* utilisé par plus de la moitié des supercalculateurs du Top 500 (entre autres sur *Titan* (#2), *Sequoia* (#3) et *K computer* (#4)). Il fonctionne sur les principaux réseaux (InfiniBand, Myrinet, Quadrics, TCP/IP...).

Un système Lustre est constitué :

- d'un seul serveur de métadonnées MDS (*Meta Data Server*) qui gère un système de fichiers MDT (*Meta Data Target*) pour celles-ci,
- éventuellement un serveur de métadonnées MDS de backup qui peut prendre la main en cas de panne sur le MDS principal,
- d'un ensemble de serveurs de données OSS (*Object Storage Server*) qui gèrent chacun plusieurs OST (*Object Storage Target*)
- et des clients.

Les MDT et OST utilisent le système de fichiers ext4 modifié (*ldiskfs*) (ZFS pour la machine *Sequoia*) et peuvent utiliser LVM et du RAID.

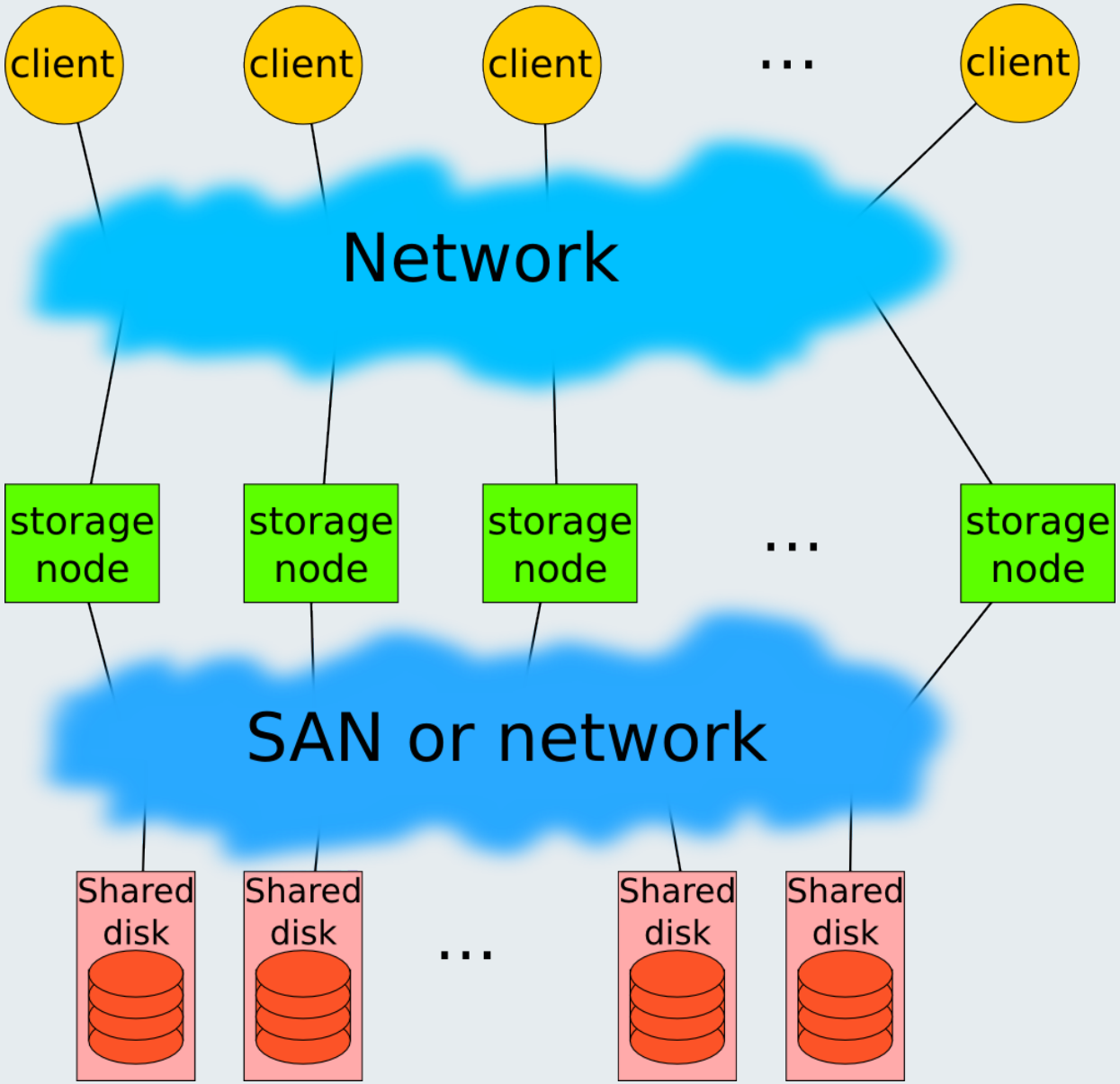
Fonctionnement de Lustre

Lorsqu'un client veut accéder à un fichier,

- il contacte le MDS qui lui fournit les informations sur les OST qui détiennent les données ou sur lesquels il va pouvoir écrire ;
- le MDS modifie les métadonnées si nécessaire ;
- ensuite, le client communique directement avec les OSS pour lire ou écrire les données.

Les verrous se font par OST sur des intervalles d'octets et sont gérés par les OSS.

Schéma


client
client
client
...
client
Network
storage node
storage node
storage node
...
storage node
SAN or network
Shared disk
Shared disk
...
Shared disk
Shared disk

Architecture de GPFS

GPFS est un système de fichiers parallèle développé par IBM sous licence commerciale et utilisé sur de nombreux supercalculateurs (entre autres sur le #5 du Top 500 *Mira* et à l'IDRIS).

Un système GPFS est constitué :

- d'un ensemble de serveurs de stockage qui s'occupent des données et des métadonnées (qui peuvent être séparées ou pas),
- d'un ensemble de disques partagés (*SAN-attached* ou *network block device*) et accessibles par n'importe quel serveur de stockage
- et des clients.

Les métadonnées sont distribuées sur les différents serveurs de stockage avec un seul nœud responsable des métadonnées d'un fichier donné.

Les verrous sur un fichier (sur des intervalles d'octets) sont selon les circonstances soit distribués entre les différents nœuds, soit gérés par un nœud précis.

Architecture et fonctionnement de PVFS2/OrangeFS

PVFS2 et OrangeFS sont des systèmes de fichiers parallèles sous licence *Open Source*. Ces 2 systèmes sont très proches et ne varient que dans les détails. Ils fonctionnent sur les principaux réseaux (InfiniBand, Myrinet, Portals, TCP/IP...).

Un système PVFS2 ou OrangeFS est constitué :

- d'un ou plusieurs serveurs de métadonnées,
- d'un ensemble de serveurs de données
- et des clients.

Ils ont quelques caractéristiques particulières :

- Optimisés pour MPI avec support des types dérivés ;
- Approche sans verrous (*lockless* ou *stateless*). Cela simplifie fortement les choses, mais, par exemple, pas de garantie d'atomicité si 2 clients écrivent au même endroit en même temps (on n'aura pas soit l'un, soit l'autre, mais peut être un mélange des 2 !) ;
- Ne respecte pas la sémantique POSIX.

Comme pour Lustre, lorsqu'un client veut accéder à un fichier, il contacte un serveur de métadonnées qui lui fournit les informations sur les serveurs de données à utiliser et, ensuite, le client communique directement avec ces serveurs pour lire ou écrire.

Architecture de PanFS

PanFS est un système de fichiers parallèle développé par Panasas sous licence commerciale. Il s'agit d'une solution clé en main avec tout le matériel et le logiciel intégrés. *Cielo* #22 au Top 500 l'utilise.

Un système PanFS est constitué :

- d'un ensemble de serveurs de métadonnées (*DirectorBlades*). Chaque serveur de métadonnées ne gère qu'un seul *volume* qui correspond à un répertoire du système de fichiers. Ils s'occupent également de la gestion des verrous ;
- d'un ensemble de serveurs de données (*StorageBlades*)
- et des clients.

Les données peuvent être automatiquement migrées entre les différents serveurs de données pour équilibrer la charge.

Plutôt que d'utiliser du RAID classique, PanFS utilise de l'*object RAID*. Chaque fichier est découpé en objets (comme pour la plupart des autres systèmes de fichiers parallèles). Et ces objets sont protégés individuellement par l'utilisation de l'algorithme RAID-5 (configurable par l'utilisateur). Les sommes de parités sont calculées directement par les clients (qui peuvent ainsi vérifier l'intégrité des données lors de la lecture).

Sources de pertes de performances

- Saturation du/des serveur(s) de métadonnées
- Conflits d'accès sur la même portion de fichier (verrous) et *false sharing*
- Contention avec d'autres I/O
- Accès aléatoires
- Tailles des accès trop petites
- Effet *Read-Modify-Write*

Comment tirer parti des systèmes de fichiers parallèles ?

Pistes à explorer

- Utiliser le parallélisme (plusieurs accès simultanés, bibliothèques d'I/O parallèles telles que MPI-IO, HDF5, Parallel-NetCDF...)
- Limiter le nombre de fichiers (moins de métadonnées)
- Eviter les accès aléatoires
- Favoriser les accès à des blocs contigus de données les plus gros possibles
- Si plusieurs accès simultanés dans le même fichier, essayer de les espacer à l'intérieur de celui-ci (sauf si utilisation des opérations MPI-IO collectives en 2 phases qui regroupent des accès discontigus au niveau des processus en accès contigus au niveau du fichier)
- Utiliser les *hints* MPI-IO (utiles aussi pour les bibliothèques basées sur MPI-IO)
- Aligner les accès sur les frontières des *stripes* et n'avoir qu'un processus par *stripe*