PUGNÈRE Denis
CNRS / IN2P3 / IP2I

Introduction au stockage distribué

ANF User Support Tools 4 HPC : 19-23 juin 2023, Aussois

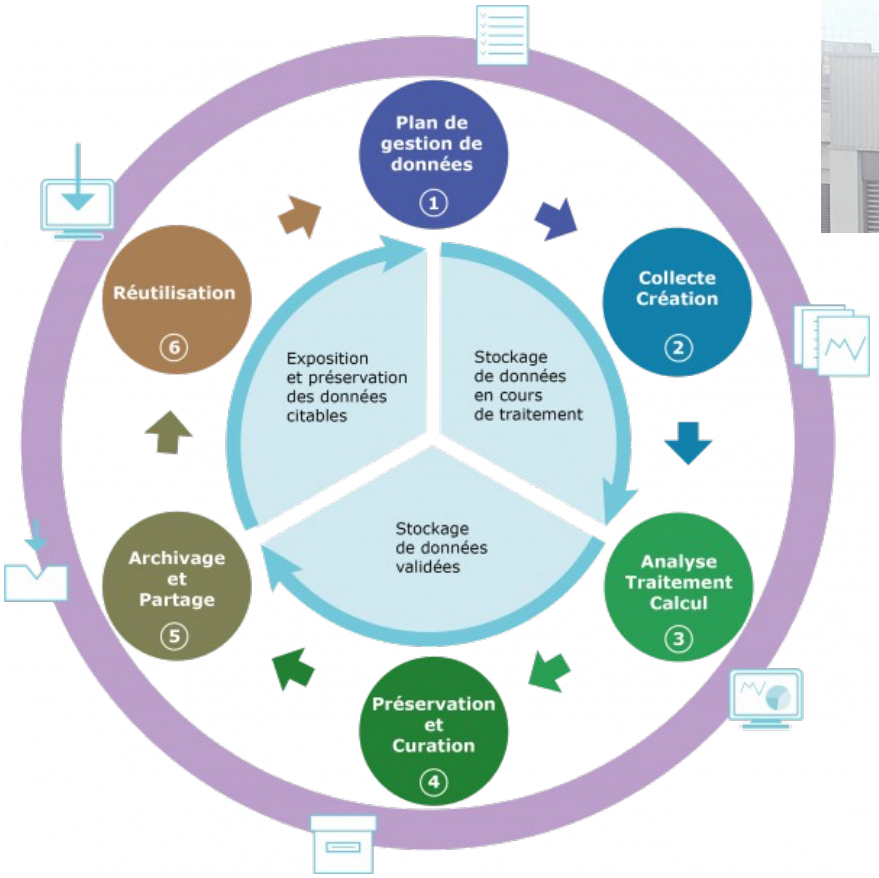
Qui suis-je ?

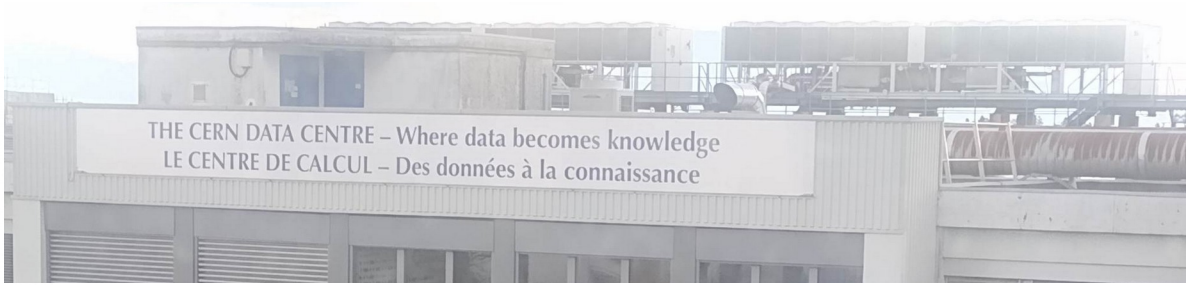
- Ingénieur système à l'Institut de Physique des 2 Infinis de Lyon (IP2I) : laboratoire CNRS/IN2P3 & Université Claude Bernard Lyon 1, ~260 personnes) : Chargé mission calcul & stockage
- Responsable technique d'un T3 LCG (LHC Computing Grid)
- Architecte du système de stockage et de traitement « Online » des données de l'expérience ProtoDUNE Dual Phase @ CERN

Plan

- Les données, les types de systèmes de stockage
- Critères de choix d’un système de stockage ou de traitement de données
- Fonctionnalités et impacts des fonctionnalités sur les performances
- Tableau comparatif de quelques systèmes de stockage
- Exemple de fonctionnement d'un SS distribué XrootD et Lustre
- Méthodologie de tests
- Problème de la migration des données
- Le modèle versus la réalité

Cycle de vie des données





Quand les données deviennent de la connaissance..

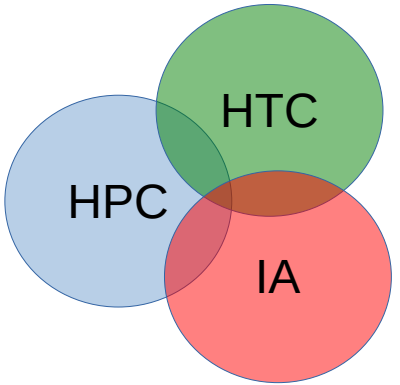
Les données représentent le bien le plus précieux

Selon le principe **FAIR** (Findable, Accessible, Interoperable, Reusable) définissent les fondements d'un partage de données **faciles à trouver**, **accessibles**, **interopérables** et **réutilisables**

Référence illustration : https://scienceouverte.univ-grenoble-alpes.fr/donnees/cycle-de-vie-principes-fair/

Définir le besoin de stockage, de traitement

- Stockage :
  - POSIX : $HOME ? /scratch ? <- les besoins sont différents
  - De fichiers non POSIX,
  - Stockage d'objets (key <-> value),
  - Stockage de blocks, images de VM, images de containers
  - Stockage massif : Acquisition de données ? Archivage ?
  - Distribution géographique des données ? PRA ?
- Selon les types de traitements à réaliser sur ces données :
  - HTC (High Thougput Computing), data mining, « analytics » : I/O dominant
  - HPC (High Performance Computing) : CPU dominant
  - IA : Machine Learning, Deep Learning : GPU dominant
  - Streaming : traitement des données au fil de l'eau, déclenchement de traitements sur événements ...
- Selon le pattern d'accès :
  - get / put, fetch / store / update
  - SQL query versus NoSQL
  - Map-reduce (traitement -> agrégation -> réduction) : pré-indexé dans les workers



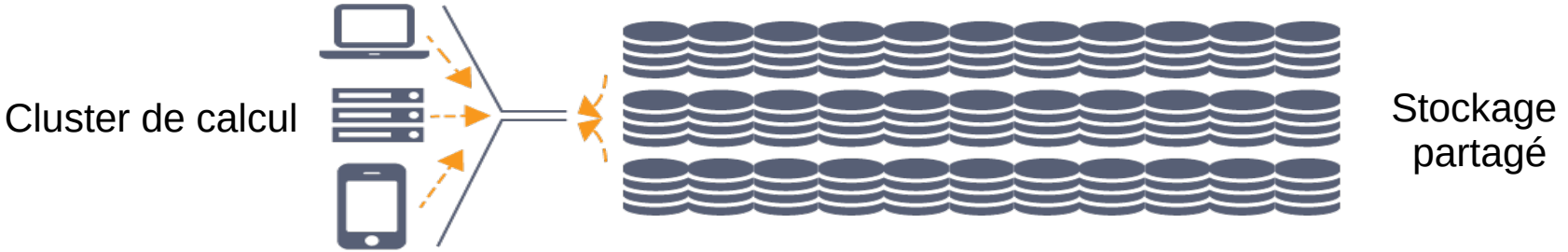
=> Tenir compte : Du type, du volume, de la criticité des données et du type d'accès à ces données

=> Décrire le workflow : construire la matrice de flux et les contraintes matérielles et temporelles

Paradigmes de traitement des données

À cause du problème “Moving Computation is Cheaper than Moving Data”

On passe du monde classique « **Compute centric paradigm »** : CPU + … + CPU <=> stockage + .. + stockage



Au mode « **Data centric paradigm** » : CPU + stockage intégré <=> CPU + stockage intégré <=> ...

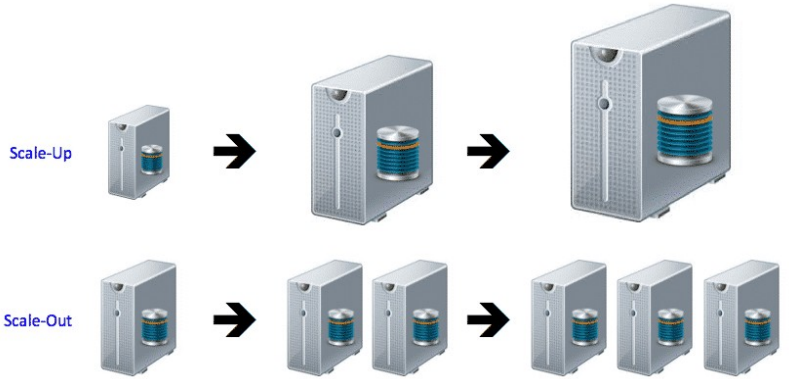
Référence illustrations : https://www.nextplatform.com/2016/10/06/emergence-data-centric-computing/

Évolutivité du stockage

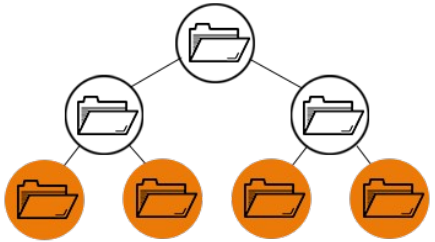
Évolutivité verticale (scale-up)

ou horizontale (scale-out)

du stockage ?

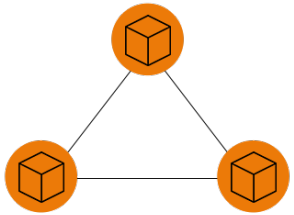


	Scale-up	Scale-out
Comment ?	Ajouter de capacité (disques / tiroirs de disques) à la solution de stockage (au serveur)	Ajouter des serveurs de disques supplémentaires) à la solution de stockage
Caractéristiques	Limites techniques, saturation du serveur d'I/O	Augmentation des performances I/O et réseau globales, permet agréger la bande-passante, complexité, redondance ?



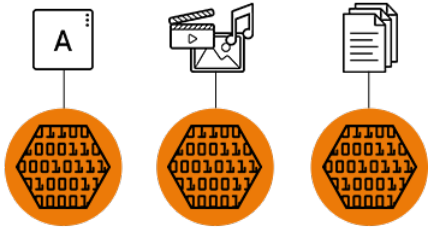
Stockage de type fichiers (POSIX)

- Caractéristiques :
  - Organisation hiérarchique des données
  - Chaque fichier est organisé dans des dossiers.
  - À chaque fichier sont associées des métadonnées (droits, heures de création / modification / accès...)
- Avantages : accès simple et natif aux données, standardisé, partage simple, verrouillage des fichiers, modifications possibles, scale-up
- Inconvénients : pas/peu adapté aux nombreux petits fichiers / objets, fragmentation des fichiers dans le temps, pas de scale-out natif (sauf NFSv4.1), performances se dégradent avec le nombre de fichiers



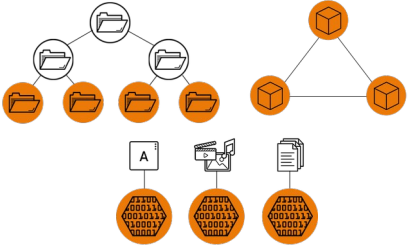
Stockage de type bloc

- Caractéristiques :
  - Chaque séquence d'informations (fichier, volume..) est subdivisé en une suite de « blocs » de taille fixe (256 Ko, 4 Mo...).
  - La taille du bloc élémentaire peut être ajustée pour correspondre à ce qui doit être stocké
  - Les blocs n'ont pas besoin d'être stockés au même endroit
  - Les blocs stockés au même endroit n'ont pas de liens entre eux
  - Chaque bloc est doté d'un identifiant unique qui permet de le distinguer des autres blocs
  - Les blocs ne contiennent aucune métadonnée en dehors de l'identifiant. Cela rend le stockage en mode bloc particulièrement efficace, puisque la quasi-totalité de la capacité de stockage du bloc est utilisée pour les données elles mêmes
- Avantages : performances (sur des gros volumes), fiabilité, faible latence, adapté pour le stockage d'éléments volumineux (images de VM, conteneurs...), modification incrémentales possibles
- Inconvénients : pas/peu de métadonnées, capacité limitée à traiter les données non structurées, Capacités de recherche et d'analyse limitées, pas adapté pour accès concurrents



Stockage de type objets

- Caractéristiques :
  - les données sont stockées sous forme d'unités discrètes appelées objets
  - Chaque unité a un identifiant unique ou une clé qui permet de la retrouver où qu'elle soit stockée dans un système distribué
  - Habituellement appelés : clé <=> valeur
  - Non hiérarchique
  - Requêtes HTTP et API REST
  - Stockage de grandes quantités de données non structurées
  - Possibilité d'associer des méta-données à chaque objet
- Avantages : Évolutif (pas de hiérarchie, capacité infinie), architecture simple, standardisé de fait (API S3), objets de toutes tailles, intégrité et disponibilité des données par la redondance (répliqua ou code à effacement), adapté pour un très grand nombre d'objets, scale-out natif, accès simultanés performants
- Inconvénients : pas de modification incrémentale des objets (les recréer), pas de verrouillage des fichiers, moins familier que le stockage de fichiers, pas compatible avec les bases de données plus traditionnelles (pas de transactionnel), latence plus importante



Stockage unifié

- Caractéristiques :
  - Un seul système de stockage offre les fonctionnalités de stockage de fichiers, de blocs et d'objets
  - Implémente les fonctionnalités au dessus d'un seul et même pool de stockage
  - Différentes applications
- Avantages : Gestion centralisée du stockage, réduction de la complexité (1 seul pool), stockage versatile et flexible, allocation dynamique du stockage aux applications, scale-out
- Inconvénients : complexité d'administration et de diagnostic, un type d'accès peut impacter l'ensemble des autres fonctionnalités

Les Software Defined Storage (SDS)

Dans le domaine du SDS POSIX & non POSIX (Stockage d'objets, key <-> value, NoSQL, map/reduce), on observe :

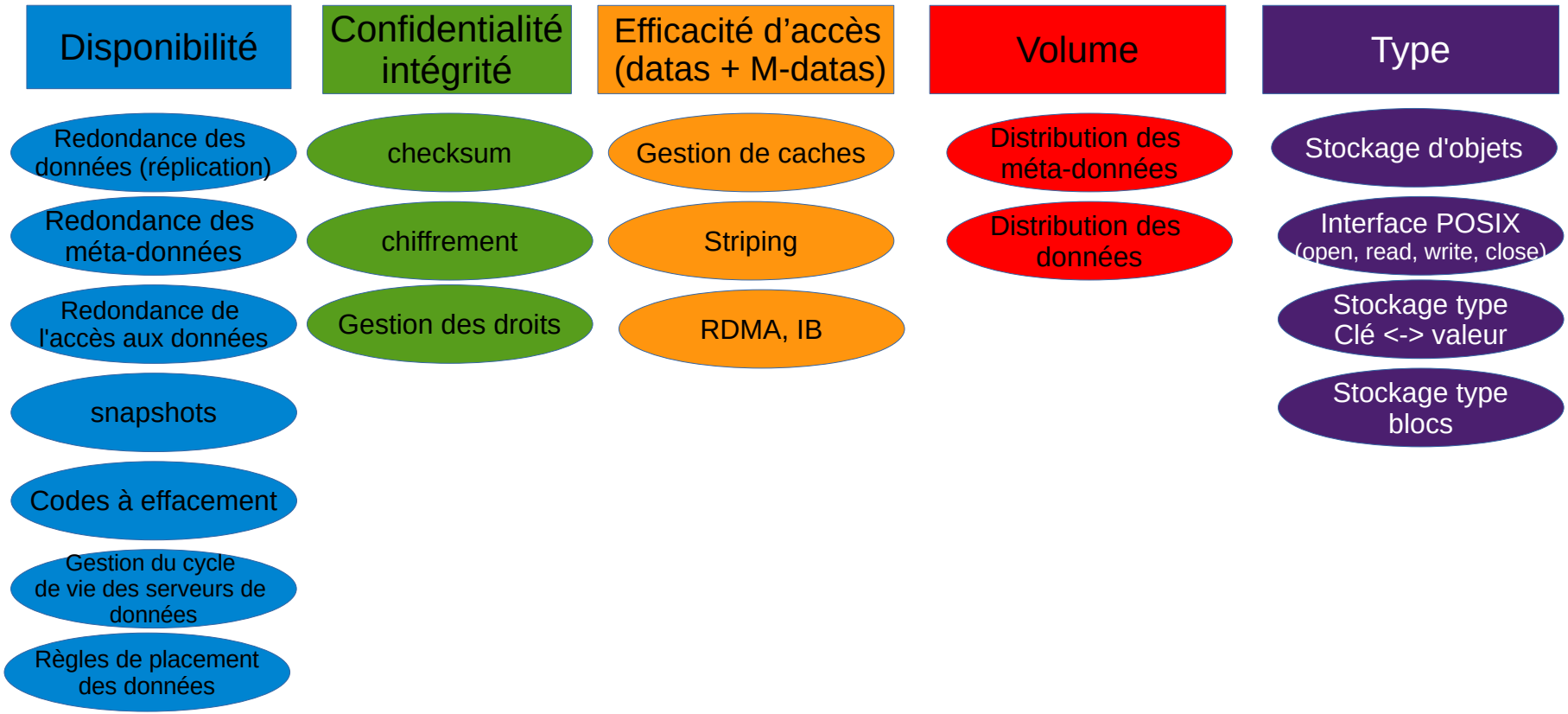
- Grande variété des caractéristiques des systèmes :
  - stockage seulement **v.s.** computing au plus près des données (Spark, Map/Reduce, OpenIO...)
- Grande entropie du comportement des systèmes. Car c'est une combinaison des :
  - profile I/O du SDS (nb replicas des objets versus code à effacement)
  - taille de chuncks, nb de data-servers,
  - quantité de meta-données, journal,
  - balancing, recovering (rebalancing)
- Profi I/O applicative (traitements local sur les données, types d'accès...)

Critères de choix d’un système de ~~stockage~~ traitement de données

- Un grand nombre de critères (souvent contradictoires) caractérisent les systèmes de stockage distribués :
  - Fonctionnalités,
  - Rapport coût / TB,
  - Rapport performance / coût,
  - Scalabilité de la solution,
  - Robustesse,
  - Complexité versus facilité de maintenance dans le temps,
  - Coût humain d'exploitation de la solution.
- Un grand nombre d'éléments matériels et logiciels sont en œuvre (y compris firmware)
- Ne pas se focaliser sur un test de benchmark sur un seul critère => mettre en œuvre plusieurs scénarios
- Quantifier le service rendu :
  - En fonction du besoin
  - En fonction des fonctionnalités **réellement utilisées**
- Ne pas négliger le coût de sortie de la solution

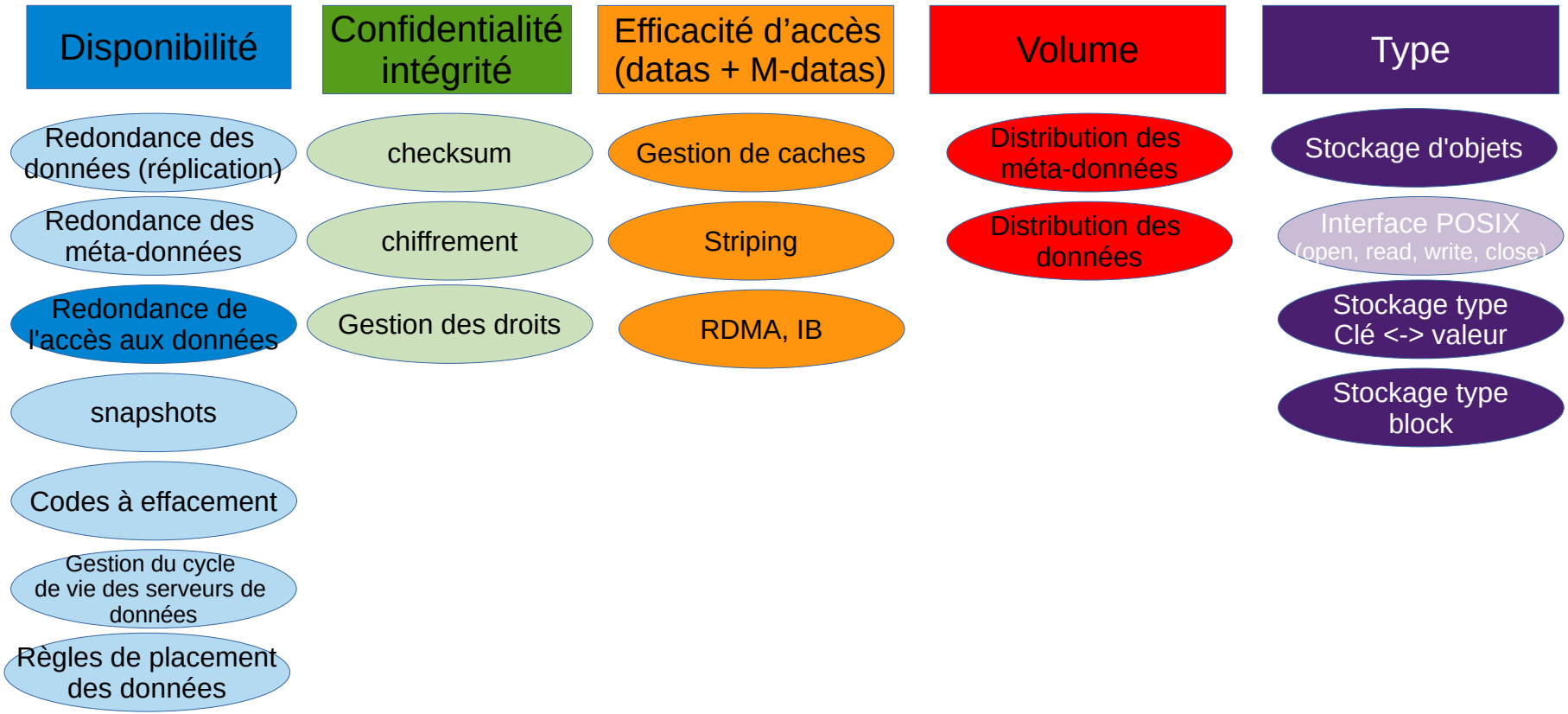
Différents types de fonctionnalités

Fonctionnalités des systèmes de fichiers/stockage distribués



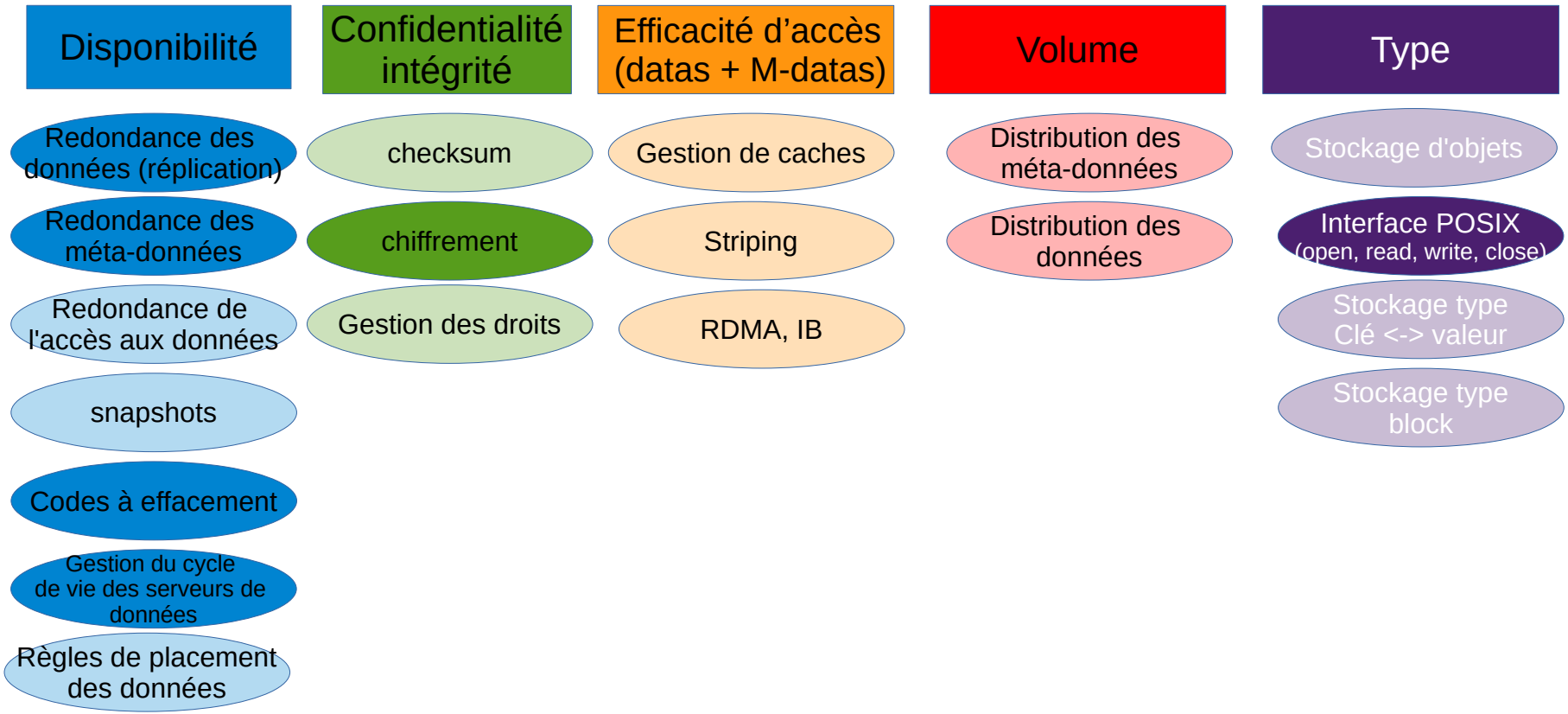
Impact des fonctionnalités sur les performances

Quelles fonctionnalités ont généralement un impact positif sur les performances ?



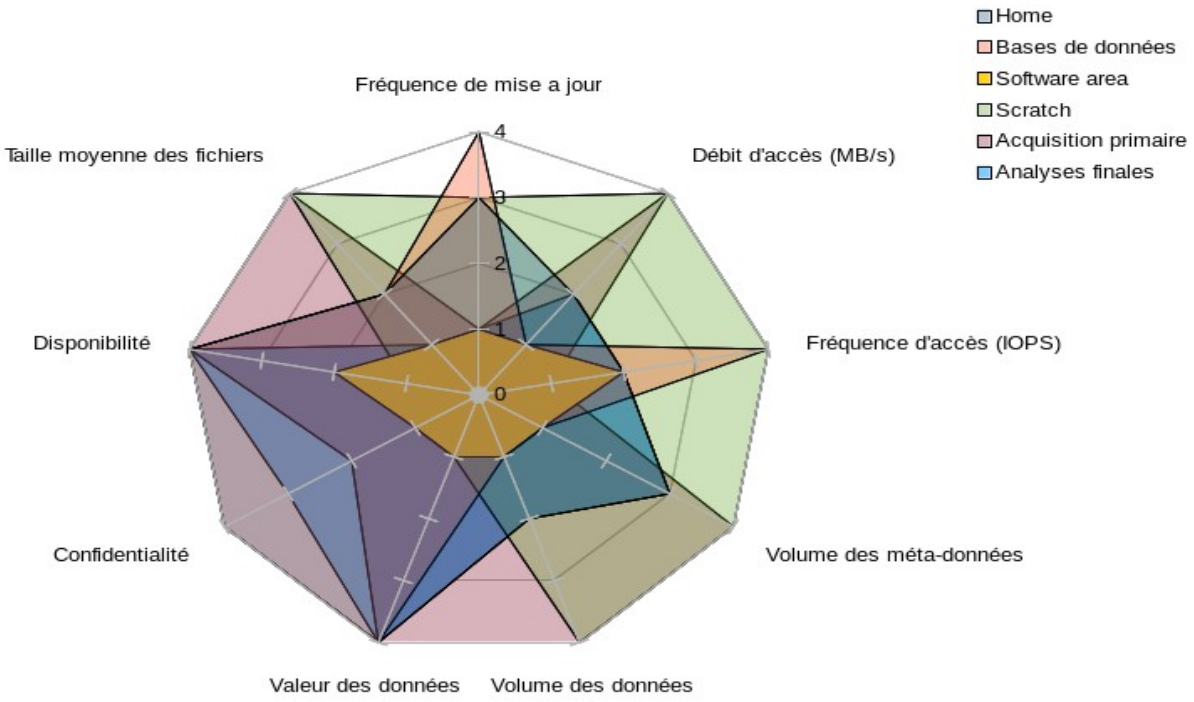
Impact des fonctionnalités sur les performances

Quelles fonctionnalités ont généralement un impact négatif sur les performances ?



Mais existe t-il un système de stockage idéal ?

Existe-il un système de stockage qui couvre tous nos besoins ?



Exemple illustratif non représentatif d'un cas général

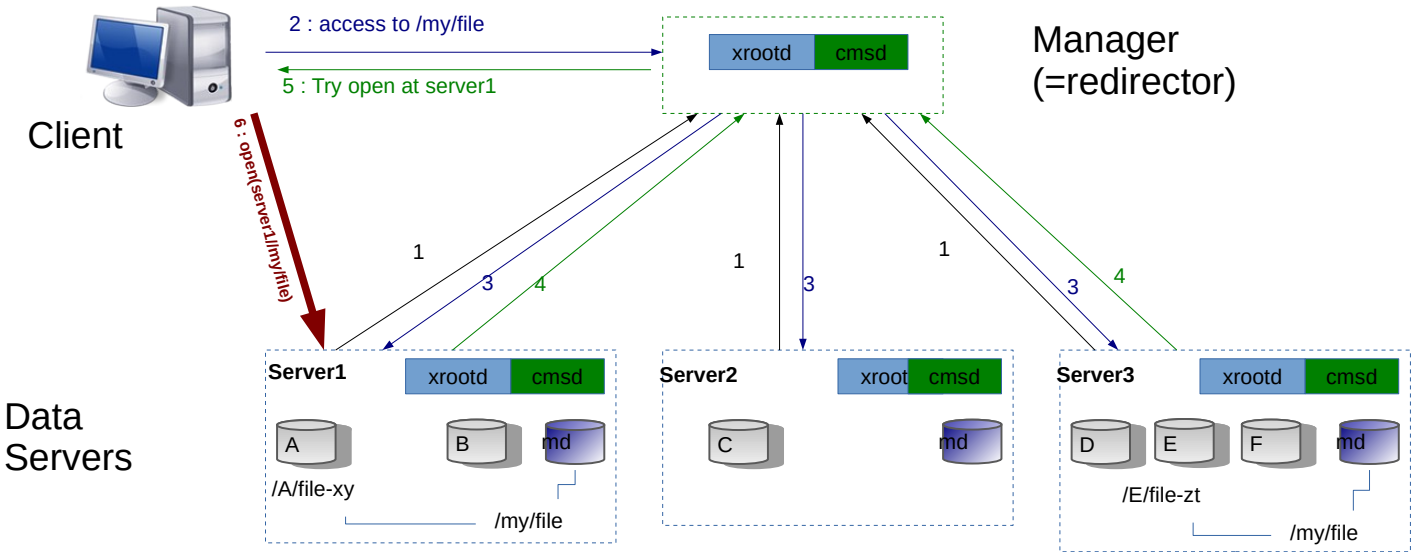
Systèmes + versions \ Caractéristiques	Lustre 2.15.x	BeeGFS 7.3.x	IBM GPFS 5.1.x	GlusterFS 11.x	Irods 4.x	Standford XrootD	CERN EOS 5.1	Ceph Quincy (17.x)	HDFS 3.3.x	S3 (Minio / OpenIO) MinIO = 2023.06
Type de stockage	fichiers	fichiers	fichiers	fichiers	fichiers	fichiers	fichiers	unifié	objets	objets
POSIX filesystem semantic	OUI	OUI	OUI	OUI	NON	OUI via FUSEX	OUI via FUSEX	OUI (via CephFS)	OUI (via HDFS NFS Gateway)	OpenIO = OUI via OIO-FS (licence)
Open Source	OUI	Client=oui,Serveur=EULA	NON	OUI	OUI	OUI	OUI	OUI	OUI	OUI
Serveur Métadata nécessaire ?	OUI	OUI + Manager	NON	NON	OUI	OUI	OUI	OUI pour CephFS	OUI	NON
Support RDMA / Infiniband ?	OUI	OUI	OUI	RDMA	NON	NON	NON	RDMA	NON	NON
Striping	OUI	OUI	OUI	OUI	NON	OUI	EC	OUI	OUI	EC
Failover (1)	M+D	DR	M+D	M+ (DR| DEC)	M+D	NON	M+DR+ DEC	M + (DR| DEC)	M+ (DR| DEC)	M+ (DR|DEC)
Quota	OUI	OUI (licence)	OUI	OUI	OUI	NON	OUI	OUI	OUI	OUI
Snapshot	NON	NON	OUI	OUI	NON	NON	NON	OUI	OUI	OpenIO (versioning)
Migration des données intégré ?	OUI	OUI	OUI	OUI	OUI	NON	OUI	Natif	OUI	Natif

(1) : M=Metadata, D=Data, M+D=Metadata+Data, DR=Data Replication, DEC=Data Erasure Coding

Exemple : Workflow du protocole XRootD

```
$ xrdcp root://manager/my/file /tmp/fichier
```

Se traduit par :



1 → Data server to manager : Hi, I'm alive, I have $x TB free, my CPU load is 0.00...

2 → Client to manager : I want to access to /my/file = open('root://manager//my/file')

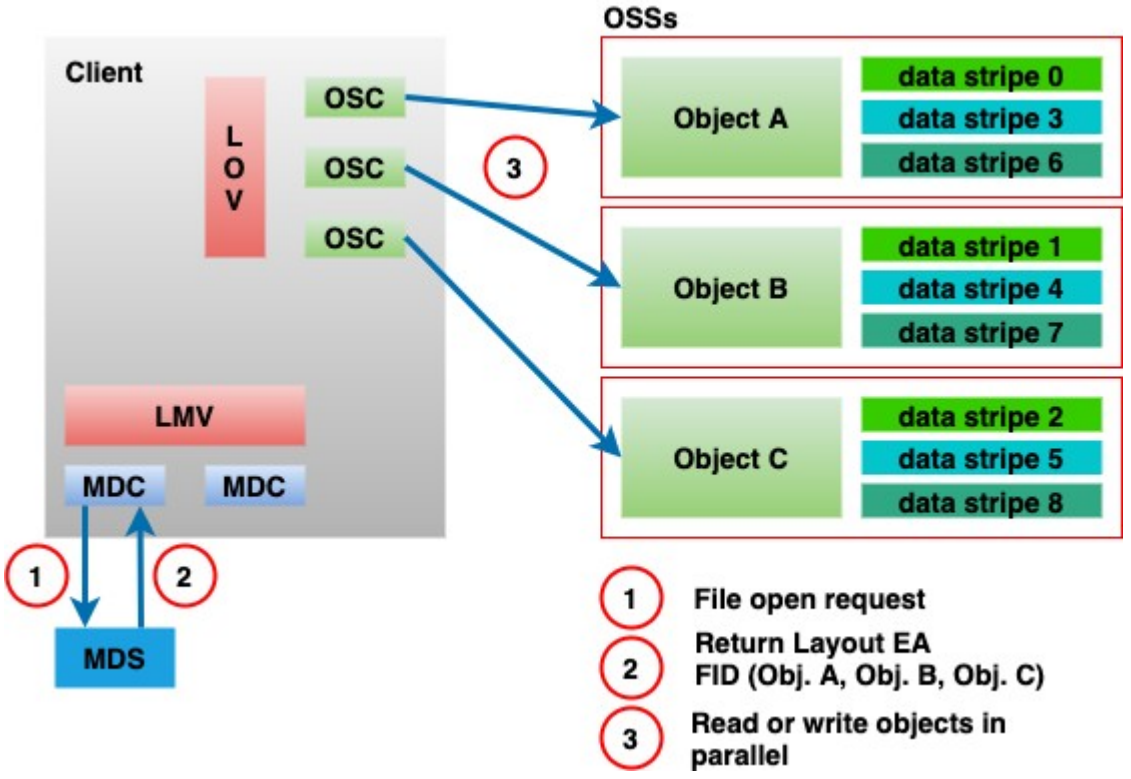
3 → Manager to data servers : who has /my/file ?

4 → Data server to manager : I have /my/file

5 → Manager to client : Try open /my/file at server1

6 → Client to data server1 : open('root :server1//my/file')

Exemple : Workflow du client Lustre



Référence illustration : https://wiki.lustre.org/Understanding_Lustre_Internals

Quel est l'objectif des tests ?

- Monter un PoC ?
- Réaliser des tests fonctionnels ?
- Comparer différents systèmes / technologies ?
- Résoudre un problème de performance ?
  Comprendre les I/O bottlenecks dans le service :
  - revient à chercher quelle est la cause de l'I/O WAIT !
  - donne des pistes pour tourner le bon bouton

Méthodologie : Connaître la capacité de son infrastructure

Principe des tests de base :

- Commencer par le plus bas niveau

- µ-benchmarks sur chaque élément du système : client(s), serveur(s), réseau

- **Tests individuels de chaque client :**
  - I/O : comportement en R, R/W, W, sequentiel, random, mixed
  - Réseau : receive, transmit, protocole (UDP, TCP), % drop, % retrans
    - si nécessaire en utilisant **+sieurs serveurs de stockage** pour tester 1 client
- **Tests individuel de chaque serveur de stockage :**
  - Configuration : JBOD, RAID type (RAID 0, 1, 5, 6, software, hardware...), block size, filesystem (EXTx, XFS, ZFS...)
  - I/O : comportement en R, R/W, W, sequentiel, random, mixed
  - Réseau : receive, transmit, protocole (UDP, TCP), %drop, % retrans,
    - si nécessaire en utilisant **+sieurs clients** pour tester 1 serveur de stockage
- **Tests réseau :**
  - Test de saturation de l'infrastructure réseau : tous clients <-> tous serveurs
- **Tests de charge système :**
  - charge (CPU, interruptions, IOPS, IOWAIT…) du|des **client**(s) : en fonction nb threads / client
  - charge (CPU, interruptions, IOPS, IOWAIT…) du|des **serveur**(s) : en fonction nb clients / serveur

Méthodologie adoptée

1 : Tests réseau

+

2 : Tests des clients

+

3 : Tests du stockage

+

4 : Tests de la chaine complète

Tests principalement protocolaires :

- Protocoles TCP / UDP (outils utilisés : iperf, nuttcp...)
- Saturation des interfaces : Algorithmes de contrôle de congestion : cubic, reno, bic, htcp...
- UDP : % de perte de paquets, TCP : nombre de retransmissions, Drop de paquets
- Puis reste des tests en TCP, flux en écriture

Connaître précisément quel flux peut générer le client :

Test initiaux => optimisations => caractérisation

- Optimisations :
  - Bonding réseau : test des algorithmes LACP (IEEE 802.3ad) , balance-alb, balance-tlb
  - Optimisation buffers réseau : modif /etc/sysctl.conf
  - Utilisation de Jumbo frames (MTU 9216)
  - Charge CPU : répartition des IRQ sur tous les coeurs
    - chkconfig irqbalance off ; service irqbalance stop
    - Mellanox : set_irq_affinity.sh p2p1

Tests individuels des éléments de stockage :

- benchmark du système de fichier local (outils utilisés : Iozone, fio, dd)

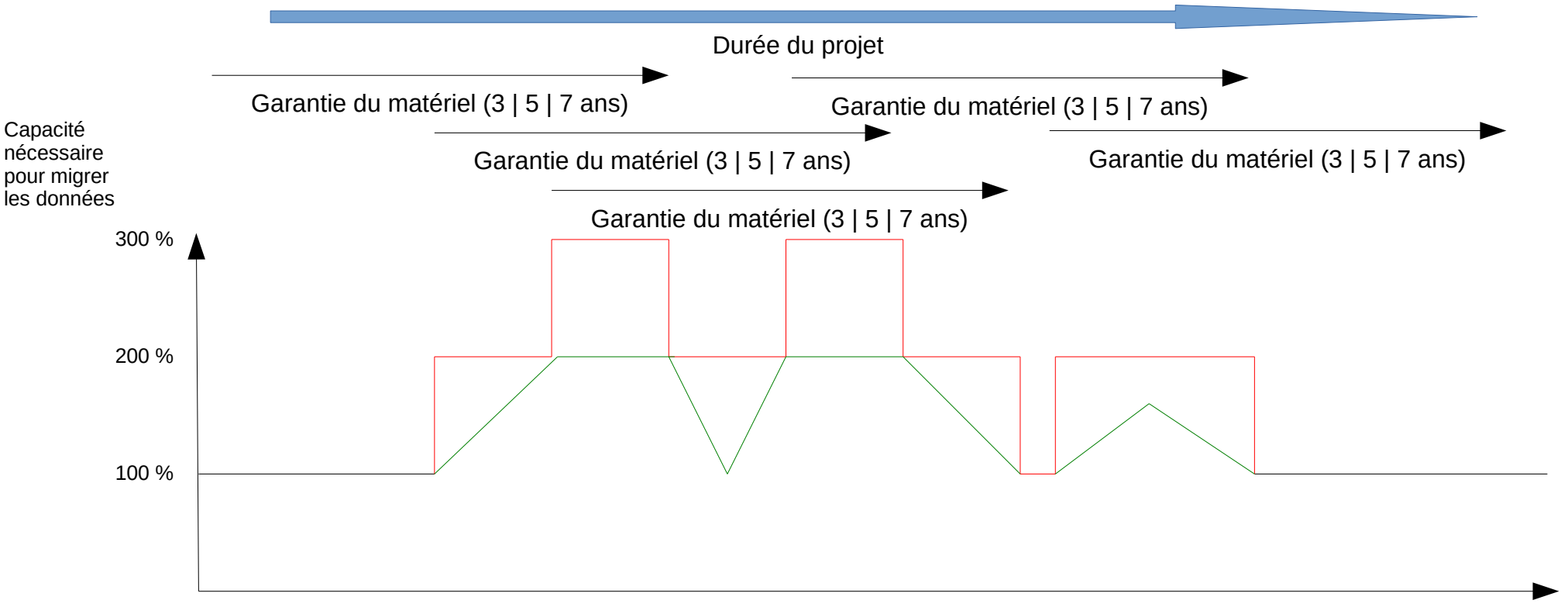
Tests de la chaine complète :

- Sur le client
  - Stockage : Iozone, fio, dd, xrdcp, icp
  - Réseau / système : dstat
- Sur les éléments de stockage : dstat

Problématique de la migration des données

- Les besoins évoluent, les capacités aussi
- La durée de vie des projets est différente de la durée de vie des systèmes de stockage
- Migration des données = transfert des données d'un système de stockage à un autre, ou d'une partie du système de stockage vers l'autre partie
  - Mises à niveau capacité, changement de versions, maintenances, déplacement des matériels …
- Impacts sur le temps de transfert ou sur la capacité
- Migration des données = processus à prendre en compte

Cycle de vie du matériel de stockage versus durée du projet : migration des données



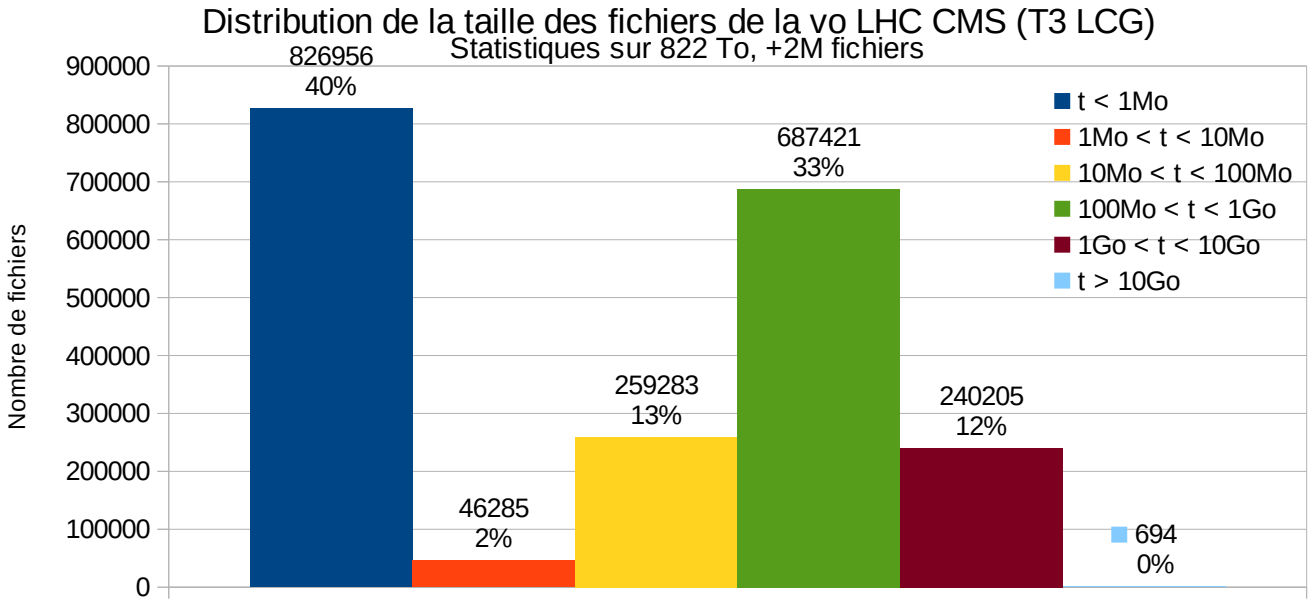
-- Sans outil intégré d'aide à la migration des données
-- Avec outil intégré d'aide à la migration des données

Les consignes de stockage données aux utilisateurs <=> la réalité

Exemple de consigne[1] pour les sites CMS : « The nominal CMS file size is 5-10GB ».

Dans les faits, au bout de plusieurs années d'exploitation sur le site WLCG CMS T3_FR_IPNL :

* 88 % des fichiers ont une taille < 1Go
* **seulement 12 % de fichiers > 1Go**



[1] : https://twiki.cern.ch/twiki/bin/view/CMSPublic/SiteOperationProcedures

Références

Outils de tests et de monitoring :

- Iozone : http://www.iozone.org
- nuttcp : http://nuttcp.net/nuttcp/nuttcp.html
- iperf : http://software.es.net/iperf/
- dstat : http://dag.wiee.rs/home-made/dstat/
- fio : http://git.kernel.dk/?p=fio.git;a=summary

Systèmes de stockage et d'accès :

- Xrootd : http://xrootd.org/papers/Scalla-Intro.pdf
- EOS : http://eos.web.cern.ch/
- IRODS : https://irods.org/
- BeeGFS : https://www.beegfs.io
- Lustre : http://lustre.org/
- GPFS (General Parallel File System → IBM Spectrum Scale) : https://www.ibm.com/support/knowledgecenter/en/SSFKCN/gpfs_welcome.html
- GlusterFS : https://www.gluster.org
- Ceph : https://ceph.com/en/
- MinIO : https://min.io/
- HDFS : https://hadoop.apache.org

Systèmes de traitement de données :

- Apache Spark : https://spark.apache.org
- Apache Hadoop : https://hadoop.apache.org