Outils algorithmiques et logiciels pour le stockage distribué

Benoît Parrein, MCF-HDR
Université de Nantes (Polytech Nantes), laboratoire LS2N, UMR 6004

ANF Des données au BigData : exploitez le stockage distribué !
12/12/2016, Gif-sur-Yvette

UNIVERSITÉ DE NANTES

LS2N LABORATOIRE DES SCIENCES DU NUMÉRIQUE DE NANTES

POLYTECH NANTES

Who I am ?

- 2001 Doctorat de l'Université de Nantes (description multiple de l'information par **transformation Mojette**)
- 2004 MCF Université de Nantes, Polytech, **département Informatique**
- **01/01/2017** Responsable équipe **RIO** (Réseaux pour l'Internet des Objets), laboratoire **LS2N**, UMR 6004
- Projets majeurs :
  - **SEREADMO :** Sécurité des Réseaux Ad hoc par Transformée Mojette (Draft IETF)
  - **P2PWeb :** modèle hybride client-serveur – P2P pour la distribution multimédia (valorisation des travaux au sein de la société **Easybroadcast**)
  - **FEC4Cloud :** codes correcteurs pour le stockage sur une infrastructure Cloud (valorisation des travaux au sein de la société **Rozo Systems**)

UNIVERSITÉ DE NANTES

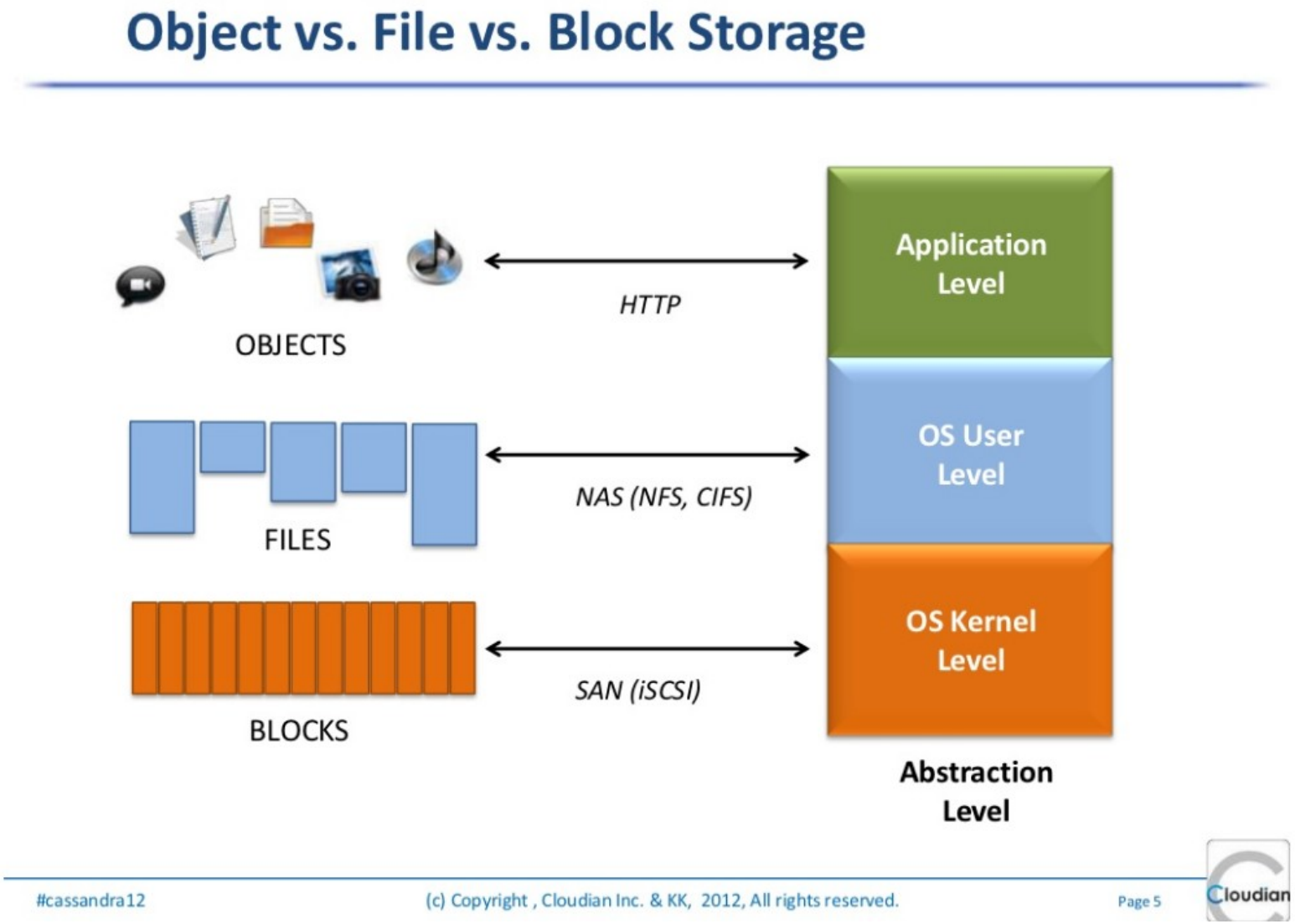
LS2N LABORATOIRE DES SCIENCES DU NUMÉRIQUE DE NANTES

POLYTECH NANTES

Sommaire

- Modes de stockage (objet, bloc, fichier)
- Théorème CAP
- Outils pour le stockage distribué
  - Fonction de hachage
  - CRUSH (avec un exemple jouet)
  - *Distributed Hash Table*
  - Filtres de Bloom
  - Déduplication
  - Codes correcteurs
  - Codes Mojette
  - RozoFS
- Tentative de classification des solutions *Software Defined Storage* (SDS)
- Architecture Fog pour l'IoT

Stockage par objet, fichier ou en mode bloc



http://www.slideshare.net/geminimobile/cloudiancassandra-summit-2012-13954057

Stockage distribué par objet

- **Caractéristiques :** méthodes d'accès via PUT et GET, méta-données distribuées, taille des blocs de 1 à 4 Mo en moyenne, scalabilité, cohérence des données...
- **Implémentations :** Cassandra, Rados, IPFS, Scality, WOS (DDN), Caringo, SwiftStack ...

Stockage distribué en mode bloc

- **Caractéristiques :** au plus près des disques, définition précise des LUN (*Logical Number Units*), peu extensible, hautes performances,...
- **Implémentations :** iSCSI, FCoE, ATAoE, Rados (mode bloc), RozoFS (en mode bloc)...

Stockage distribué par fichier

- **Caractéristiques :** haute disponibilité, tolérance aux pannes, POSIX, taille des blocs de 4 à 32K en moyenne, *Lock*, *SPoF* (le plus souvent),...
- **Implémentations :** GlusterFS, RozoFS, CephFS, ...

Stockage distribué par fichier (pour le HPC)

- **Caractéristiques :** très haute disponibilité, parallélisme des I/O, faible tolérance aux pannes, taille des blocs ...
- **Implémentations :** Lustre, GPFS, BeeGFS, ...

Stockage distribué par fichier (pour le Big Data)

- **Caractéristiques :** scalabilité, adapté/spécialisé aux algorithmes de type MapReduce, architecture Hadoop/Spark/Flink, communication de type RPC, taille des blocs (64Mo), pas entièrement POSIX ...
- **Implémentations (java):** HDFS

Théorème « CAP »

- **Conjecture de Brewer (PODC, 2000):**

  Il est impossible de garantir en même temps (de manière synchrone) les 3 contraintes suivantes :

  - **Cohérence** (ou consistance des données) (***C****onsistency* en anglais): tous les nœuds du système voient exactement les mêmes données au même moment ;
  - **Disponibilité** (***A****vailability* en anglais) : garantie que toutes les requêtes reçoivent une réponse;
  - **Tolérance au partitionnement** (***P****artition Tolerance* en anglais) : aucune panne moins importante qu'une coupure totale du réseau ne doit empêcher le système de répondre correctement (ou encore : en cas de morcellement en sous-réseaux, chacun doit pouvoir fonctionner de manière autonome).
- Preuve formelle de la conjecture de Brewer (Gilbert et al. 2002)
- Pour la cohérence, possibilité de consensus via Quorum (voir les protocoles Paxos)

[source : wikipedia]

Illustration du théorème CAP

- « Soit **A et B deux utilisateurs du système**, soit N1 et N2 deux nœuds du système. Si A modifie une valeur sur N1, alors pour que B voie cette valeur sur N2, **il faut attendre que N1 et N2 soient synchronisés**. Si N1 et N2 doivent toujours servir des valeurs cohérentes, alors **il y a un temps incompressible** entre le début de l'écriture, la synchronisation et la lecture suivante. Sur un système très chargé et très vaste, ce temps incompressible va considérablement **influencer la disponibilité et la résistance au morcellement**. Il existe bien évidemment des techniques pour optimiser ce temps mais plus le système est vaste, plus il est difficile à réduire. »

[source : wikipedia]

Sommaire

- Modes de stockage (objet, bloc, fichier)
- Théorème CAP
- **Outils pour le stockage distribué**
  - Fonction de hachage
  - CRUSH (avec un exemple jouet)
  - *Distributed Hash Table*
  - Filtres de Bloom
  - Déduplication
  - Codes correcteurs
  - Codes Mojette
  - RozoFS
- Tentative de classification des solutions *Software Defined Storage* (SDS)
- Architecture Fog pour l'IoT

Fonctions de hachage (définition cryptographique)

Une fonction de hachage à sens unique, ***H(M)***, opère sur un message de longueur arbitraire. Elle fournit une valeur de hachage de **longueur fixe *m***.

h = H(M), où *h* est de longueur *m*.

Propriétés

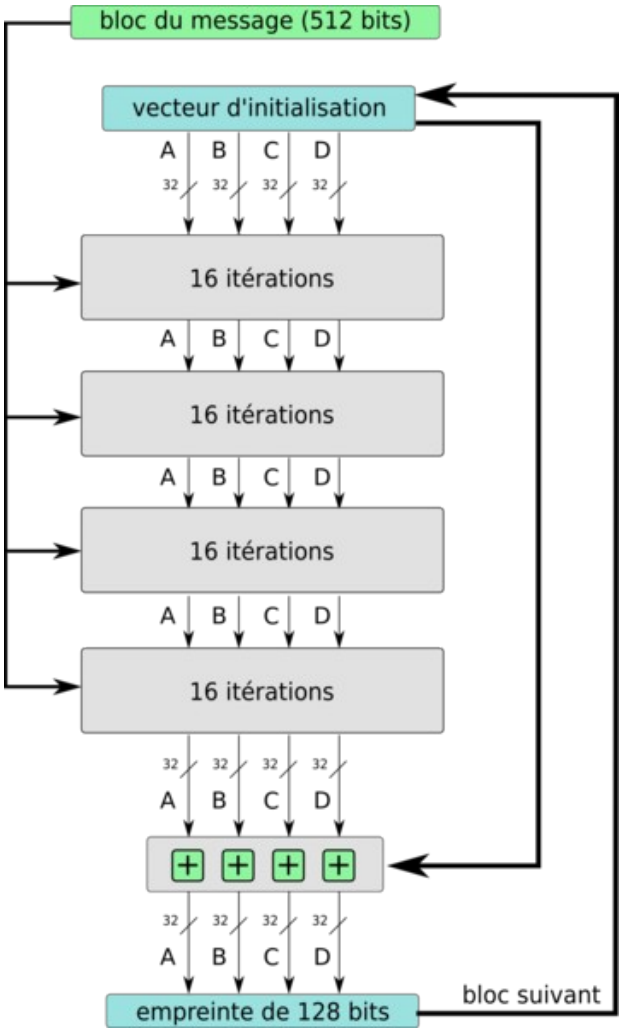
- étant donné *M*, il est facile de calculer *h*.
- étant donné *h*, il est difficile de calculer *M*.
- étant donné *M*, il est difficile de trouver un autre message *M'* tel que *H(M) = H(M') (résistance à la collision)*

Réalisations

- MD-5
- SHA-1
- SHA-256
- RIPE-MD
- ...

Principe d’une attaque :
rechercher des collisions (empreinte identique pour des messages distincts)

Vue générale du MD-5



4 rondes

4 variables de chaînage: A, B, C et D

16 exécutions d'une même opération non linéaire distincte à chaque ronde:

4 opérations non linéaires par ronde
ronde 1: F(X,Y,Z) = (X and Y) or (¬X and Z)
ronde 2: G(X,Y,Z) = (X and Z) or (Y and ¬Z)
ronde 3: H(X,Y,Z) = X xor Y xor Z
ronde 4: I(X,Y,Z) = Y xor (X or ¬Z).

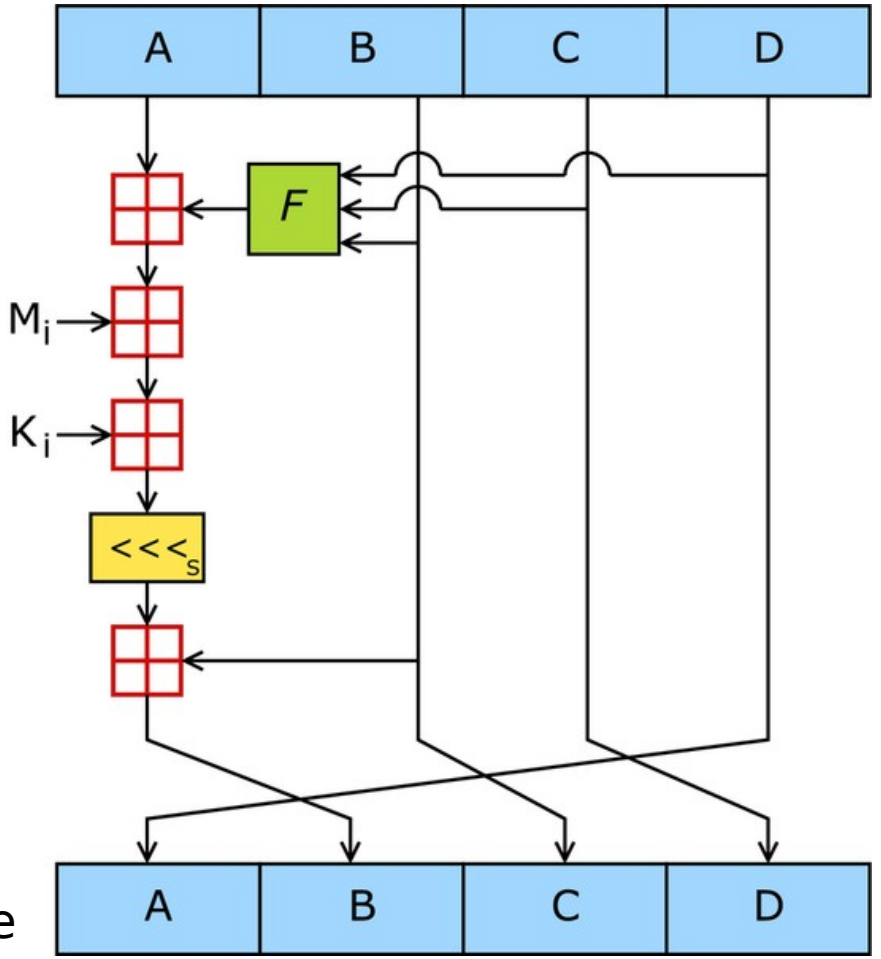
source: wikipedia

Une itération (parmi 16) du MD-5

opération XOR

1 sous-bloc message

1 constante
(par itération et par ronde)



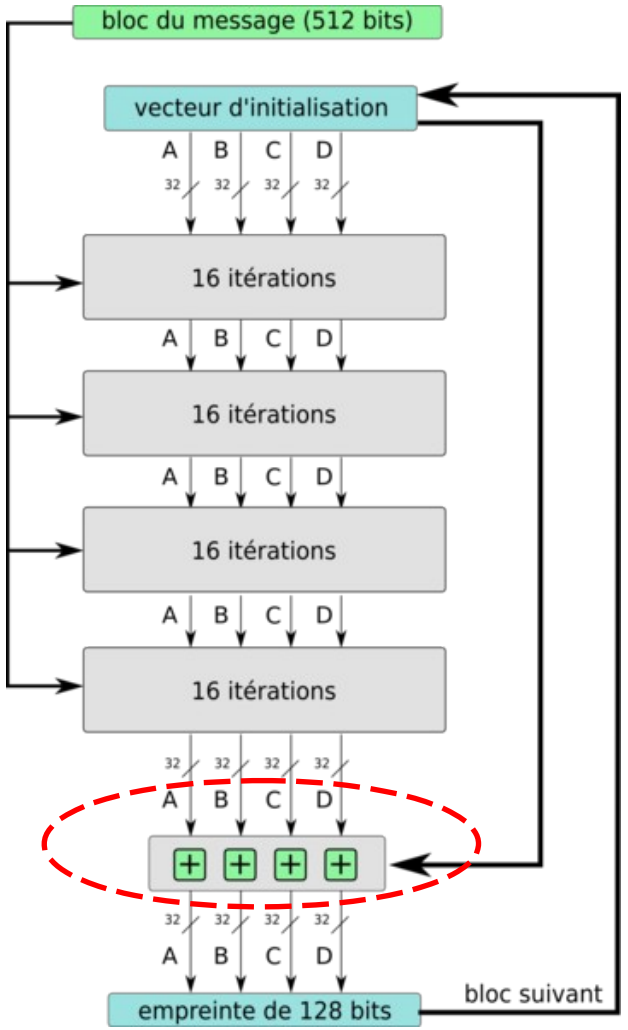
remplacement d'une des variables pour l'itération ou la ronde suivante

ronde 1: F(B,C, D)
ronde 2: G(B,C,D)
ronde 3: H(B,C,D)
ronde 4: I(B,C,D)

16 itérations x 4 rondes

source: wikipedia

Vue générale du MD-5



4 rondes

ne pas oublier!

chaînage des blocs consécutifs d'un message pour obtenir l'empreinte finale

source: wikipedia

Exercice

1 Message Digest 5 (MD5)

On cherche à calculer l'empreinte d'un message. Pour simplifier le calcul, on considérera ici 1 message de 16 bits, un vecteur d'initialisation de 16 bits (composé de 4 variables de chaînage A, B, C et D de 4 bits chacune), 1 ronde et 1 itération. La vue générale et la description d'une itération du MD5 sont données sur les supports de cours.

1.1 Empreinte d'un message

Le message est 1011000011000010. Pour $A = 1001$, $B = 0111$, $C = 1011$, $D = 0101$, la fonction $F(X, Y, Z) = (XandY)$ or $(\neg XorZ)$, le sous-bloc message $Mi = 1011$ (le premier), la constante $Ki = 1111$ et le décalage à gauche $s = 1$, calculer l'empreinte finale de 16 bits (on n'oubliera pas les 4 opérations XOR en sortie de la ronde avec les variables de chaînage du vecteur initialisation).

Extrait TD3, INFO4, Polytech Nantes

CRUSH (*Controlled Replication Under Scalable Hashing*) [Weil et al., 2006]

- CRUSH est une fonction prenant en entrée **un nom** (d'objet ou de groupe de placement), **un *clustermap*** ainsi qu'**une règle de placement** et produit en sortie **une liste de *devices*** (le vecteur i) sur lesquels les copies de l'objet seront stockés.
- CRUSH travaille en réalité sur **des groupes d'objets** (*placement groups*).
- L'idée est de calculer **un hash sur le nom de l'objet** de manière à déterminer l'identifiant de groupe. CRUSH est appliqué sur cet identifiant de groupe.

CRUSH (exemple jouet, *bucket* uniforme)

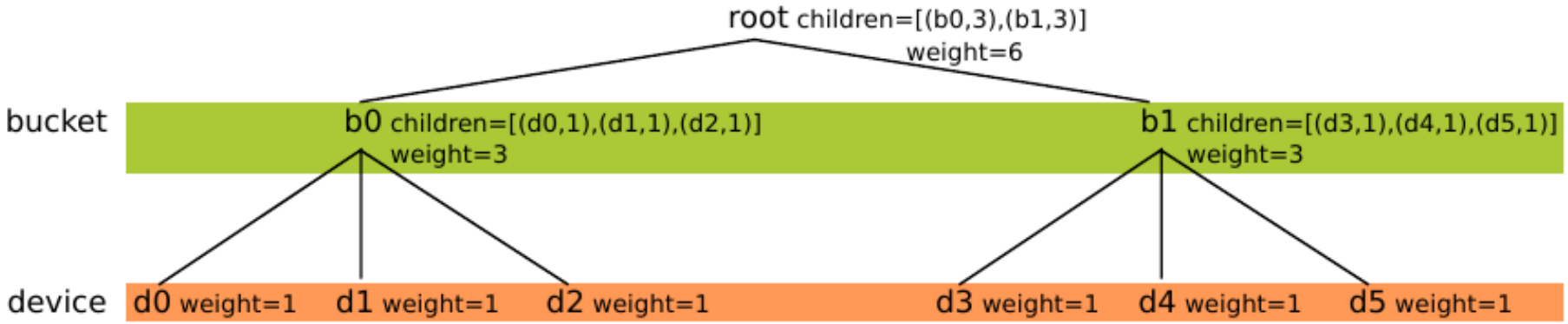


FIGURE 1 – Clustermap utilisé.

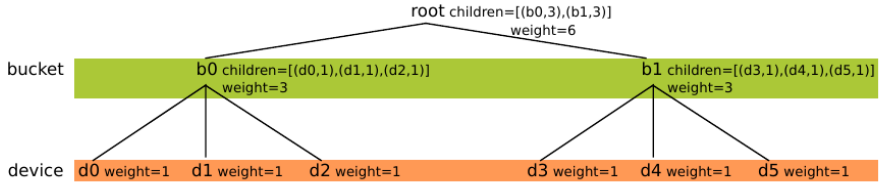
```
take(root)
select(2, bucket)
select(1, device)
emit
```

FIGURE 2 – Règle de placement utilisée.

Source : Bastien Confais, CNRS, Novembre, 2016

Exemple jouet CRUSH

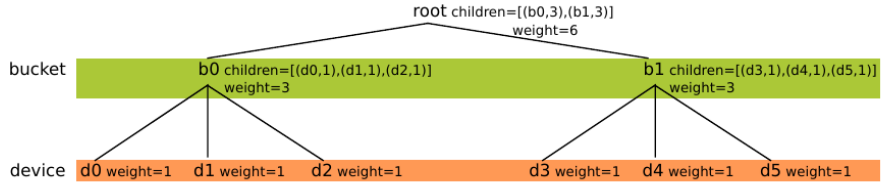
CRUSH(x, clustermap, placement_rules)



- Initialisation $\vec{i} = (root)$
- **Select(2, bucket)**
  - Application de la fonction sur tous les nœuds de i
  - Choisir 2 nœuds (de type **bucket**) dans le sous-arbre
    - Application de la fonction **c(r,x) = hash(x) + r.p mod m** | p un grand nombre premier et m le nombre de fils
    - A.N :
      - **c(1,x) = hash(x) + p mod 2** et **c(2,x) = hash(x) + 2p mod 2** soit c(1,x) = 1 et c(2,x) = 0.
    - $\vec{i} = (b1, b0)$

Exemple jouet CRUSH

CRUSH(x, clustermap, placement_rules)



- **Select (1, device)**
  - Application de la fonction sur tous les nœuds de I
  - Choisir 1 nœuds (de type **device**) dans le sous-arbre
    - Application de la fonction **c(r,x) = hash(x) + r.p mod m** | p un grand nombre premier et m le nombre de fils
    - A.N :
      - **c(1,x) = (hash(x) + p) mod 3** e.g c(1, x) = 2 (pour les 2 buckets)
        soit le 2ème *device* de chaque *bucket*
    - $\vec{i} = (d2, d5)$

Le réplicat primaire se situe au niveau de d2
(le secondaire au niveau de d5)

Collision du *hash*

- Deux fois la même valeur dans $\vec{i}$
- c(1,x) = c(2,x)
- Implique que les 2 réplicats sont placés sur le même périphérique
- CRUSH ajoute un constante à c(r,x) en collision

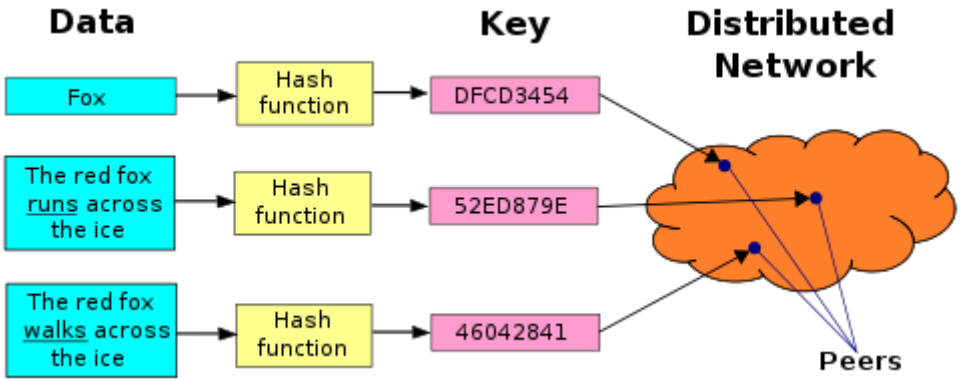
Autres types de *bucket*

- ***List :*** pris en compte des poids de l'arborescence des devices dans la sélection
- ***Tree :*** sélection du nœud fils à l'aide d'un arbre binaire
- ***Straw :*** maximisation d'une fonction de sélection. Permet de déplacer la plus petite quantité de données possibles lorsque le *clustermap* est modifié

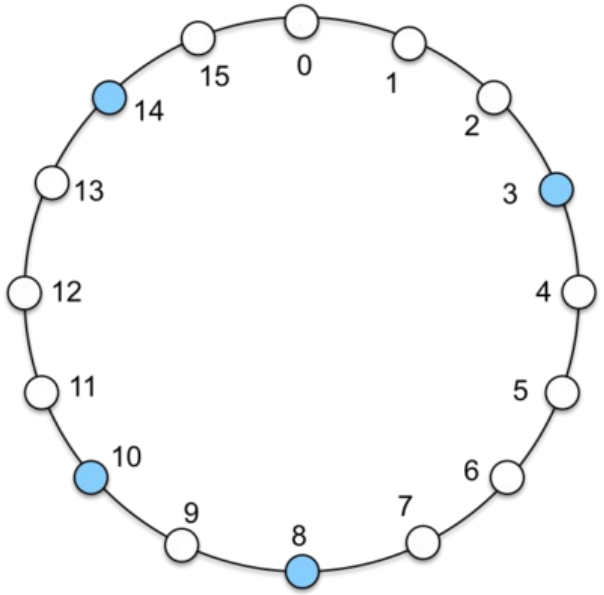
Distributed Hash Tables (DHT)

- Décentralisées
  - Chaque nœud (ou un sous ensemble) stocke une portion de l'espace des clés
- Structurées
  - Notion de nœuds successeurs qui stockent la clé (e.g SHA-1)
  - e.g 3 stocke les clés pour les nœuds 15, 0, 1, 2, 3 (qui stockent les data)
- Routage sur la clé
  - Les nœuds connaissent uniquement leur successeur. Les requêtes sont transmises aux successeurs.

Usage : Scality, IPFS, GlusterFS,...



Source : wikipedia



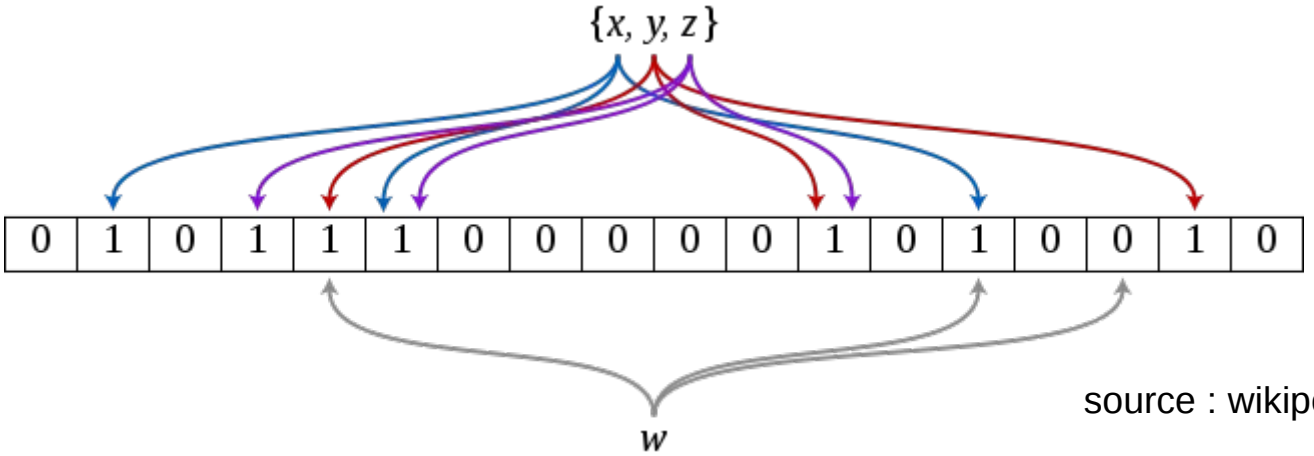
Anneau logique dans Chord

https://www.cs.rutgers.edu/~pxk/417/notes/23-lookup.html

Filtre de Bloom

Cette structure est probabiliste : lors du test de la présence d'un élément dans un ensemble, un filtre de Bloom permet de savoir :

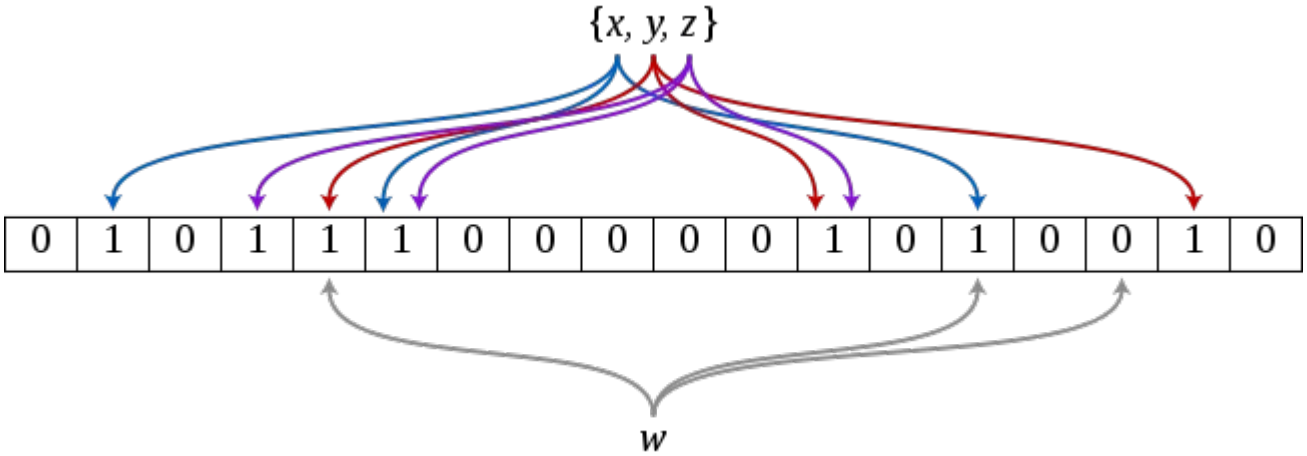
- avec **certitude** que l'élément est **absent** de l'ensemble (il ne peut pas y avoir de faux négatif) ;
- avec **une certaine probabilité** que l'élément peut être **présent** dans l'ensemble (il peut y avoir des faux positifs).



source : wikipedia

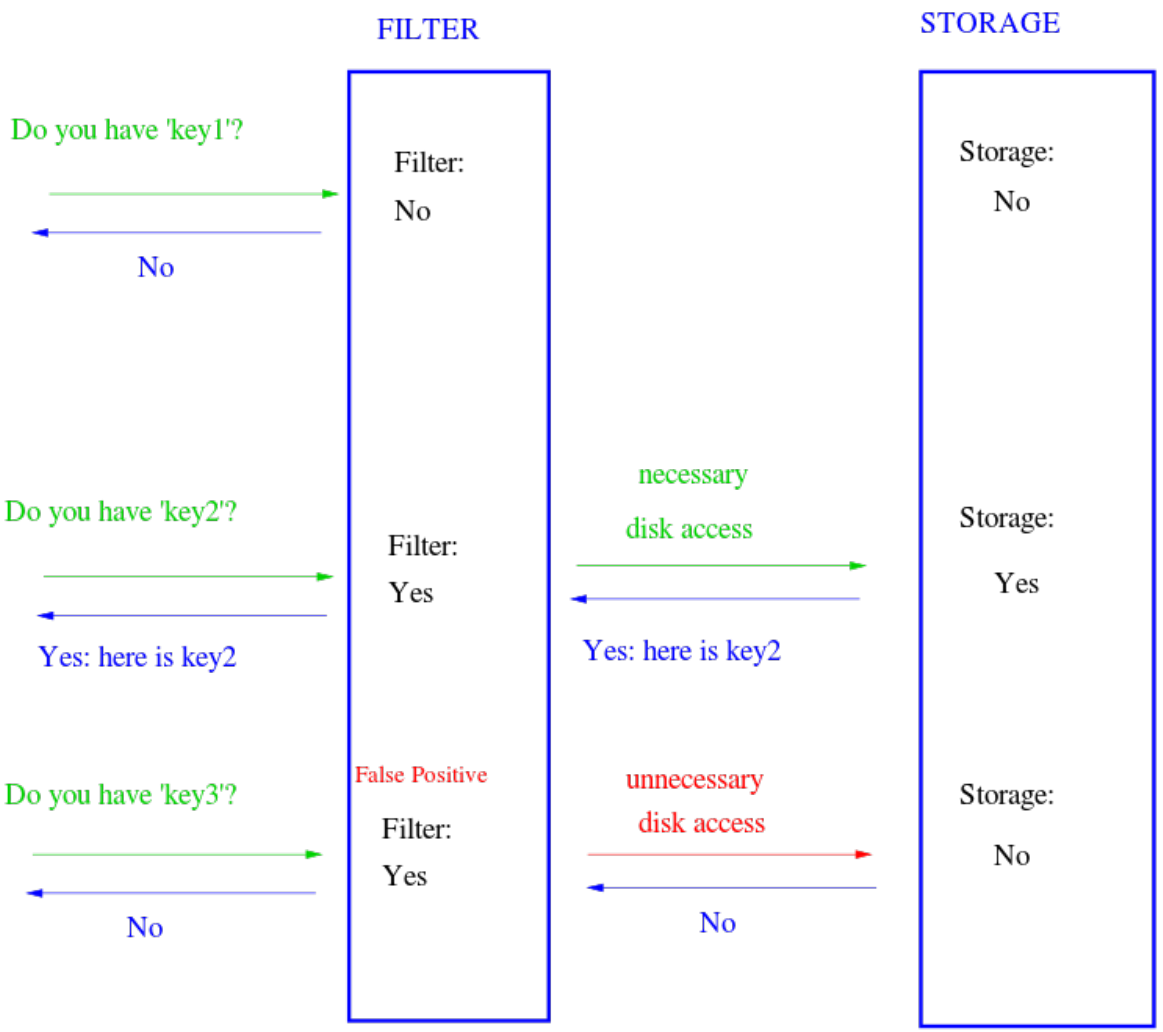
Filtre de Bloom : comment ça marche ?

- Un tableau binaire de **m bits** (ici m = 18)
- **k fonctions de hachage** différentes (ici k=3 | k << m)
- **Pour faire une requête**, appliquer les k fonctions de hachages (**sur un objet w**)
  - **si toutes retournent 1**, présence de l'objet (avec possibilité de faux positifs)
  - **si 1 bit à 0**, l'objet n'est pas dans l'ensemble



source : wikipedia

Filtre de Bloom et le stockage

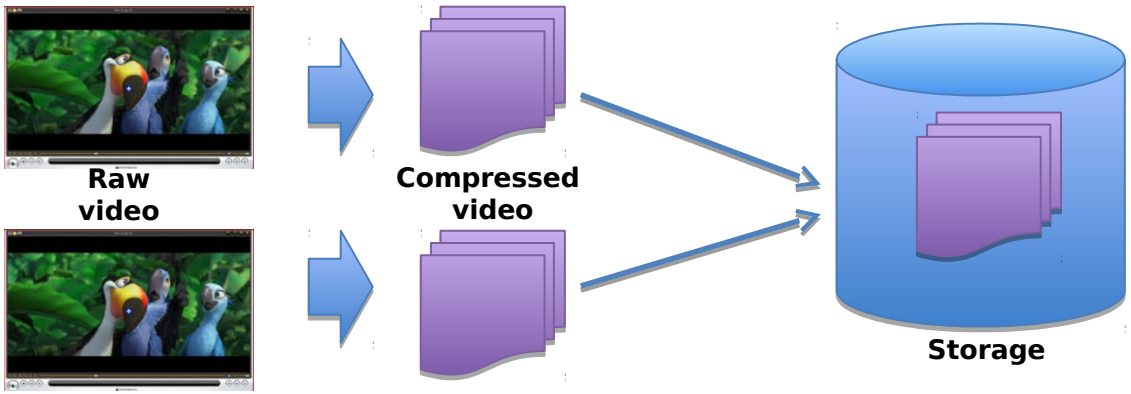


source : wikipedia

Usage : Cassandra, Akamai web servers, proxy http ...

Deduplication (from Dr. Suayb Arslan)

- Intelligent compression
- Single instance storage
  - Only one unique instance of the data is actually retained on storage media.
  - The redundant data is replaced with a pointer to the original unique copy.

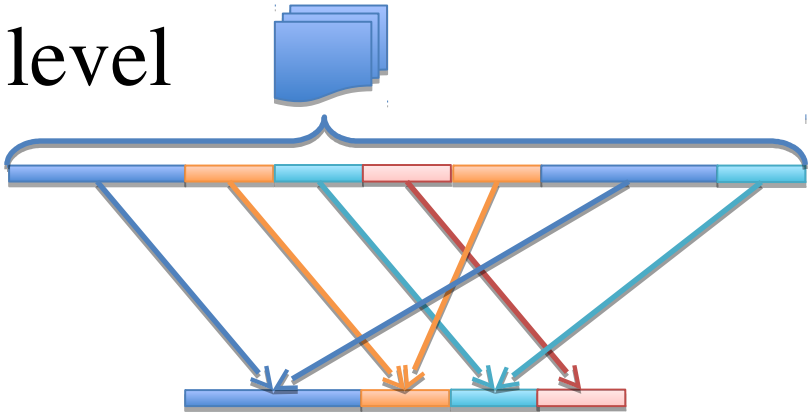


Deduplication techniques

- The file level

Store only the unique file copies

- Block level



File as a string of symbols
The string is divided into **fixed** or
variable length blocks.

How it works ?

0. blocks must be determined. Sometimes called “chunking”

1. files or blocks must be compared.

Bit by bit comparison is costly!

Hash based (predominant): Use hash functions.

For each block, a hash value is computed.

Unique data is identified based on the hash values.

Delta differencing (delta differential backup):

Similar to dalta coding in JPEG.

The baseline is a complete point-in-time copy of the data used to recreate other versions of the data.

2. need to keep an index

All the hash signatures are stored in an index stored usually in memory.New incoming files/blocks are deduplicated based on this index.

*Checksum is generated using MD5 on a 512x512 Lenna.tiff Image.

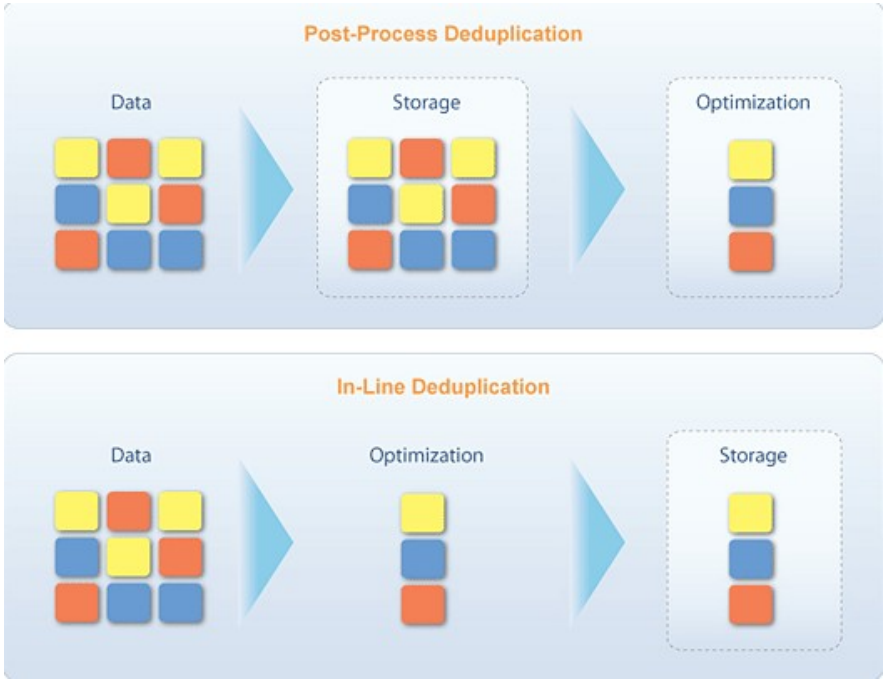
« Dedup » types

“In-line” versus “post-process”

(time of application)

In in-line deduplication, hash values are computed as the data enters the device in real time. Inaddition, hash comparisons are performed to identify duplicate chunks, all in real time.

In post-process deduplication, data is first stored on the storage device and then a process at a later time shall analyze the data for duplicates.



“source” versus “target” (point of application)

Source deduplication: occurs close to where data is created.

Target deduplication: occurs near where the data is stored.

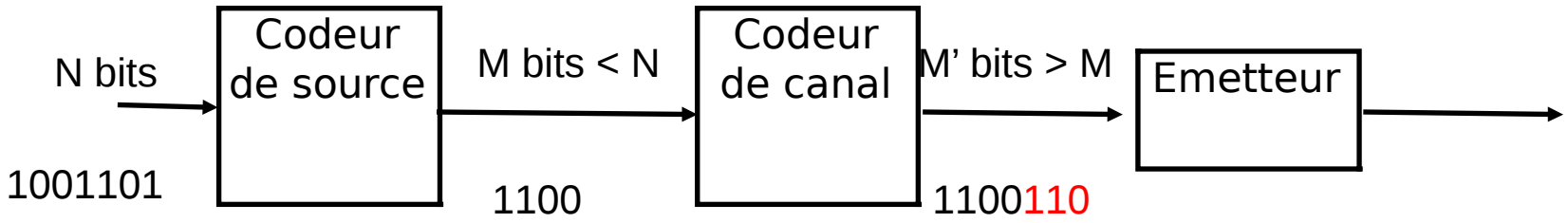
Plus d'infos en vidéos !

- By Dr. Suayb Arslan, MEF University, Istanbul

https://www.youtube.com/watch?v=bVxmQJuJaL4

https://www.youtube.com/watch?v=ZAgGeLh8i-s

Théorie de Shannon : codage source / codage canal



- **Codage source :** compression (réduction des redondance). Exemple : jpeg2000, H.265, zip,...
- **Codage canal :** protection (ajout d'une loi de détection et/ou correction d'erreur). Exemple : Hamming, Reed-Solomon, LDPC, Mojette,...

Code correcteur d'erreurs

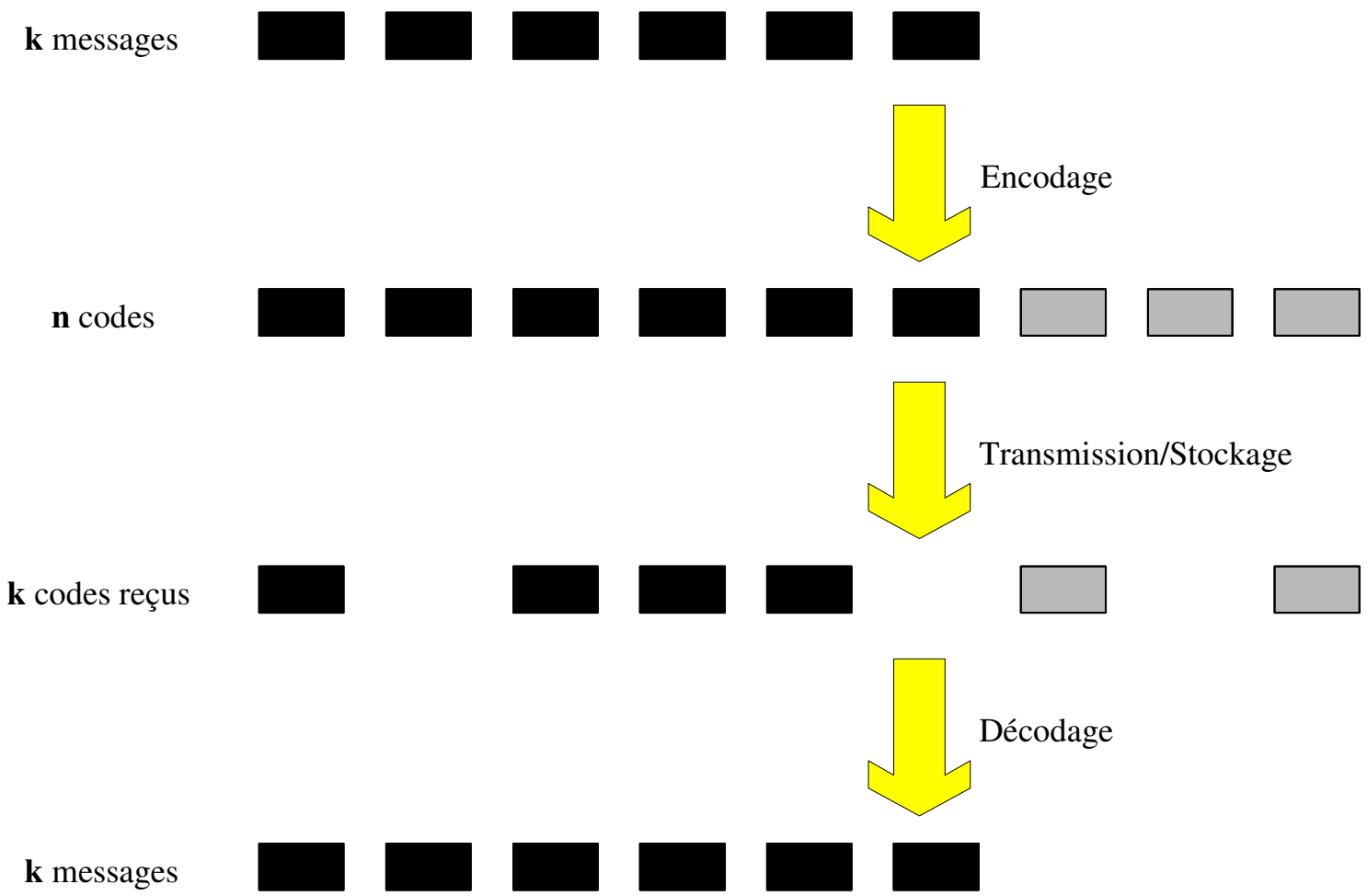
- Exemple du code linéaire Hamming (7,4)

```
>> [H,G] = hammgen(3)

H =

     1     0     0     1     0     1     1
     0     1     0     1     1     1     0
     0     0     1     0     1     1     1


G =

     1     1     0     1     0     0     0
     0     1     1     0     1     0     0
     1     1     1     0     0     1     0
     1     0     1     0     0     0     1
```

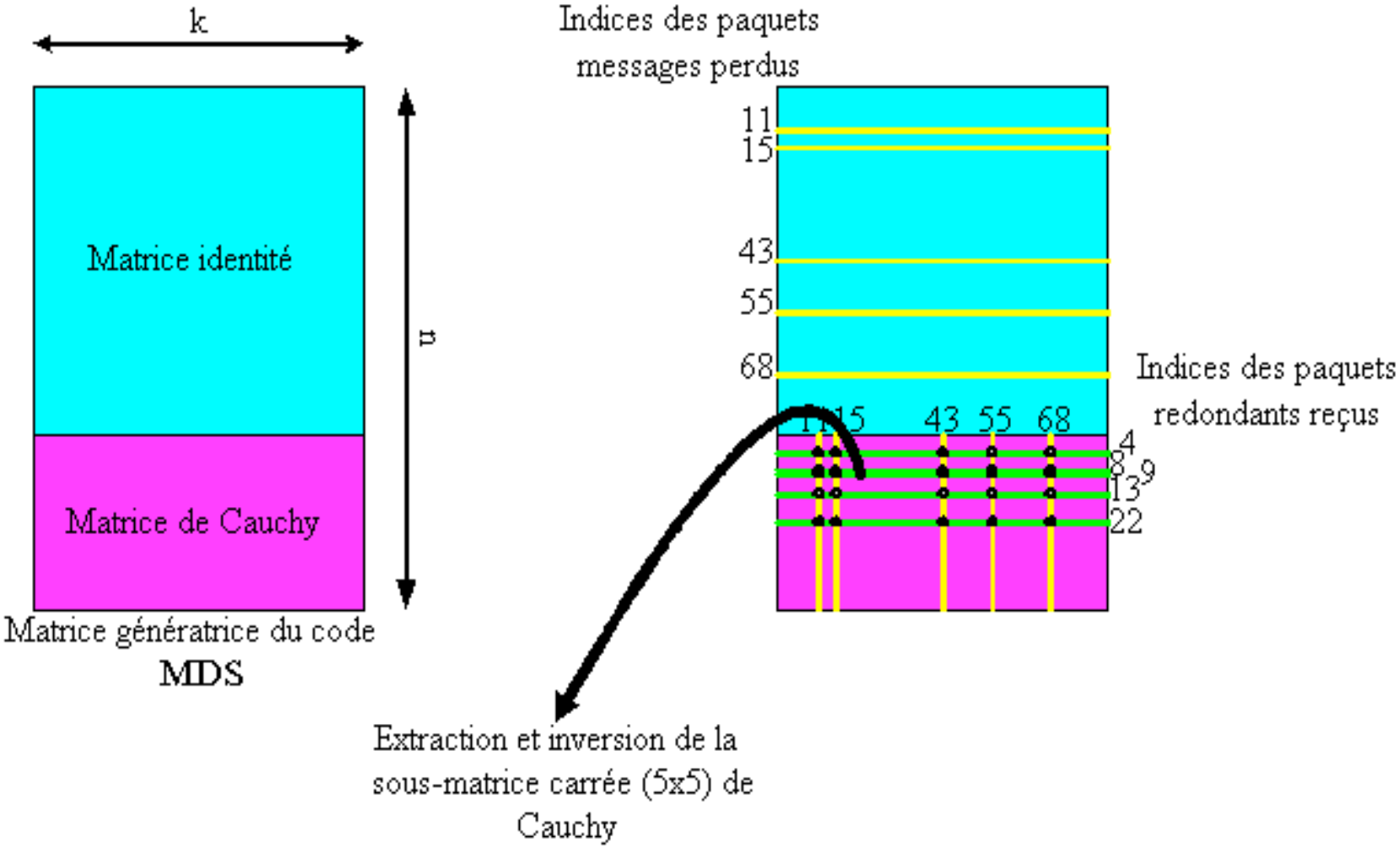
% parity check matrix

%the generator matrix

Code à effacement (en mode paquet)



Code Reed Solomon (par matrice de Cauchy)

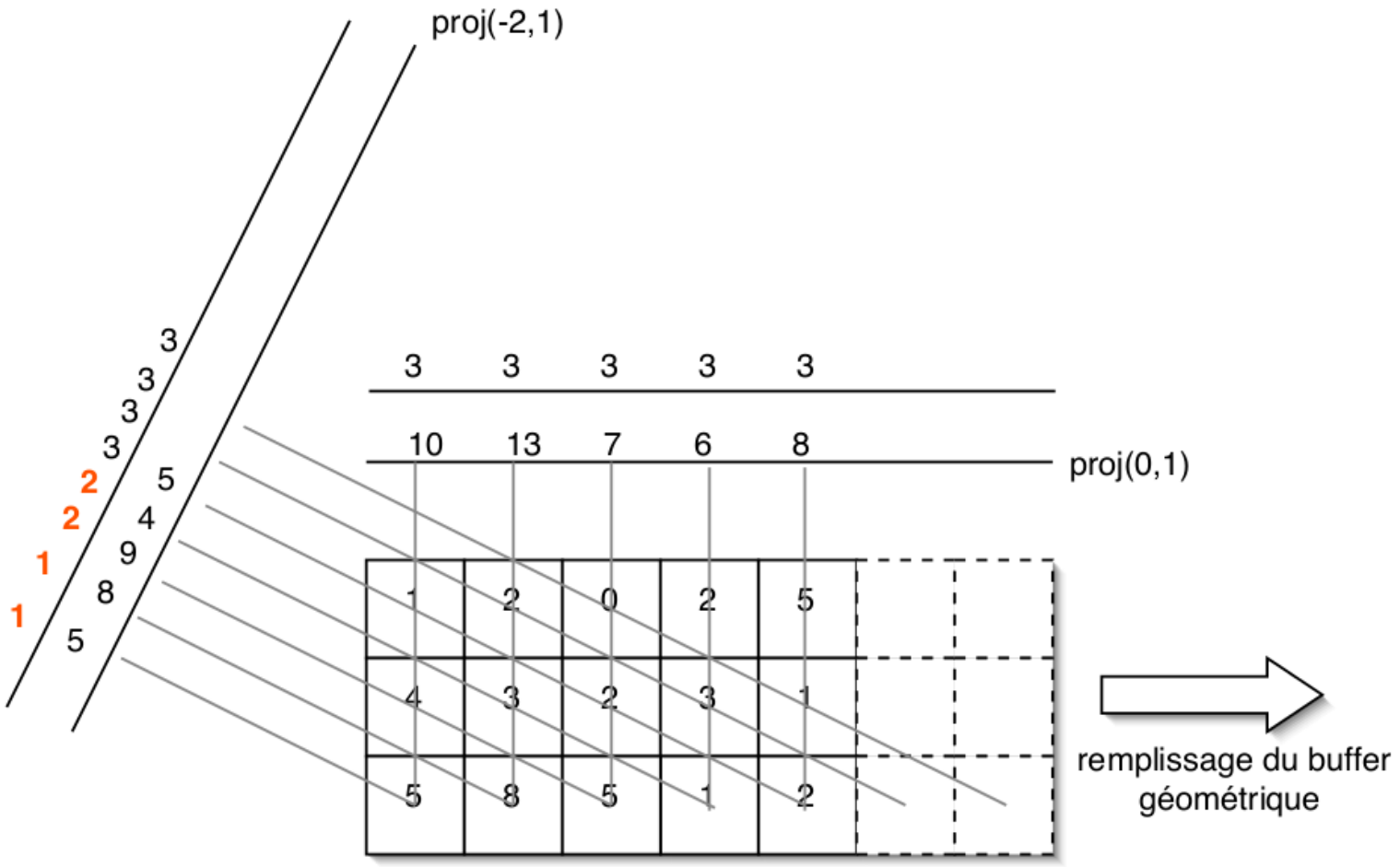


Code RS (par matrice de Vandermonde)

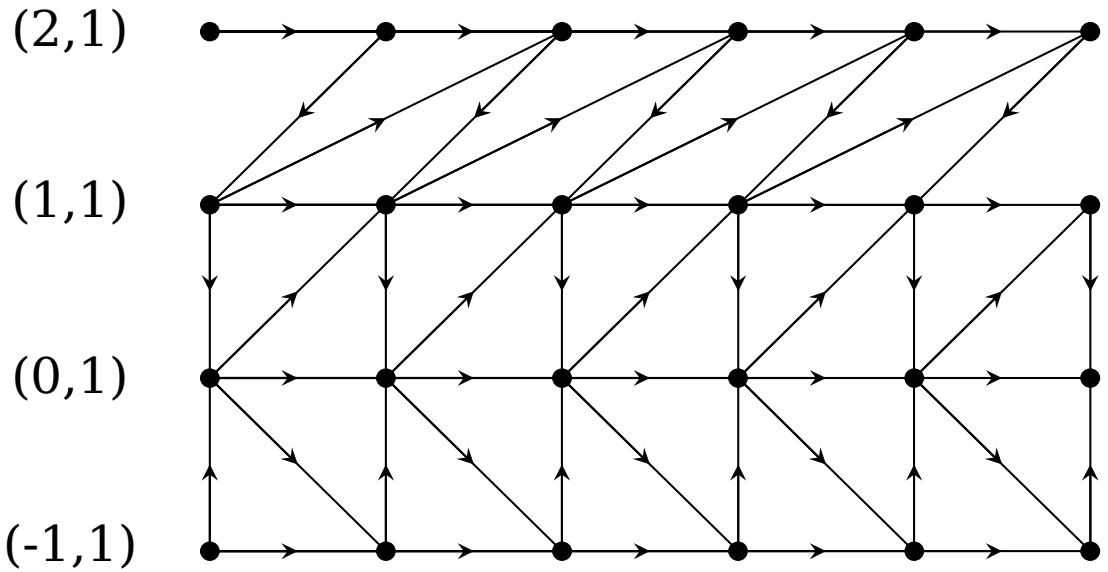
- RFC 5510
- Implémentations : ISA-L,...

$$V = \begin{pmatrix} 1 & \alpha_1 & {\alpha_1}^2 & \dots & {\alpha_1}^{n-1} \\ 1 & \alpha_2 & {\alpha_2}^2 & \dots & {\alpha_2}^{n-1} \\ 1 & \alpha_3 & {\alpha_3}^2 & \dots & {\alpha_3}^{n-1} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & \alpha_m & {\alpha_m}^2 & \dots & {\alpha_m}^{n-1} \end{pmatrix}$$

Code à effacement Mojette



Décodage (rapide) Mojette

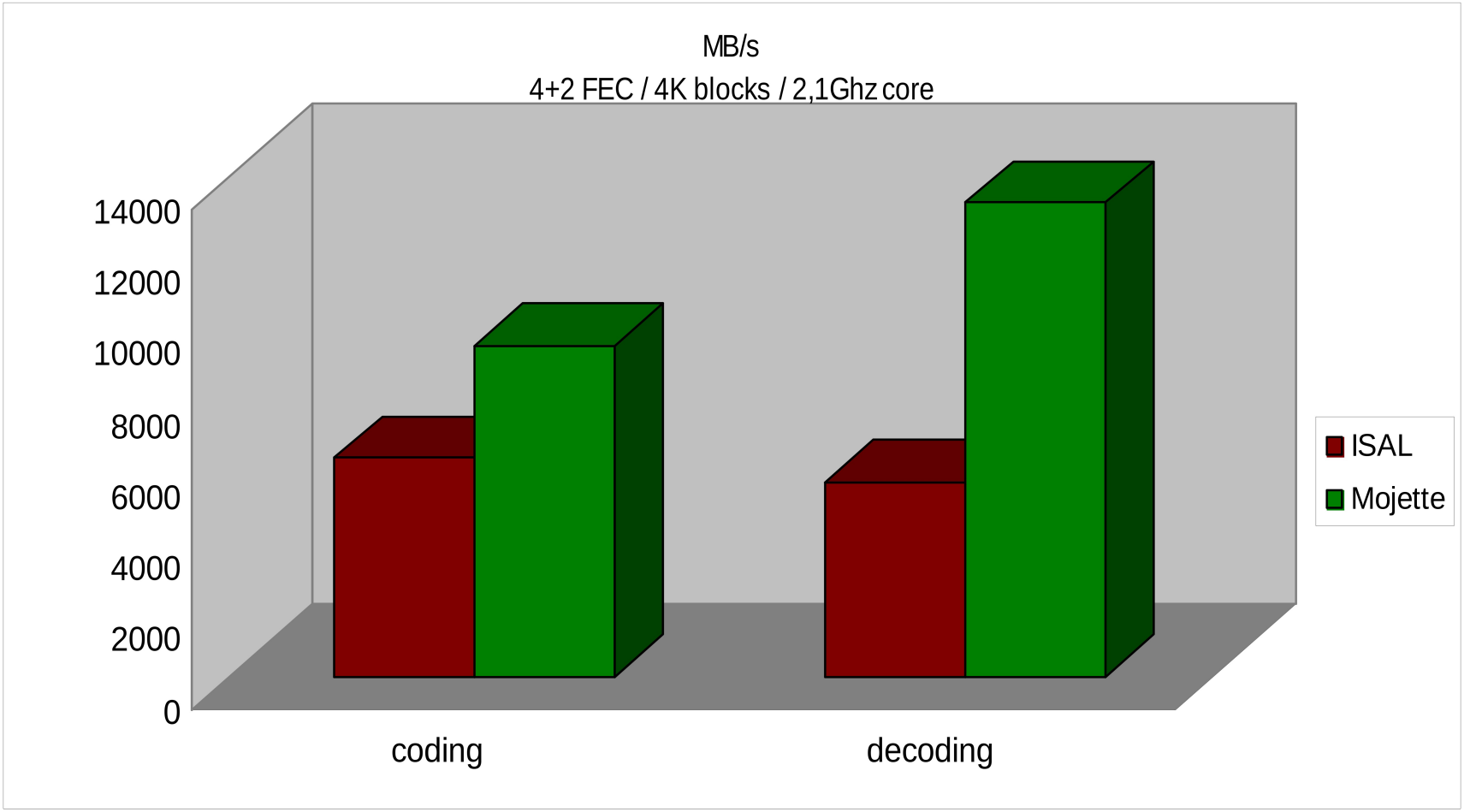


Parcours déterministes au sein des projections

(chaque nœud du graphe correspond à une rétroprojection

Exemple ici d'une grille 4xP avec 4 projections)

Performances Codes (débits)



source : rozosystems

Le stockage Mojette

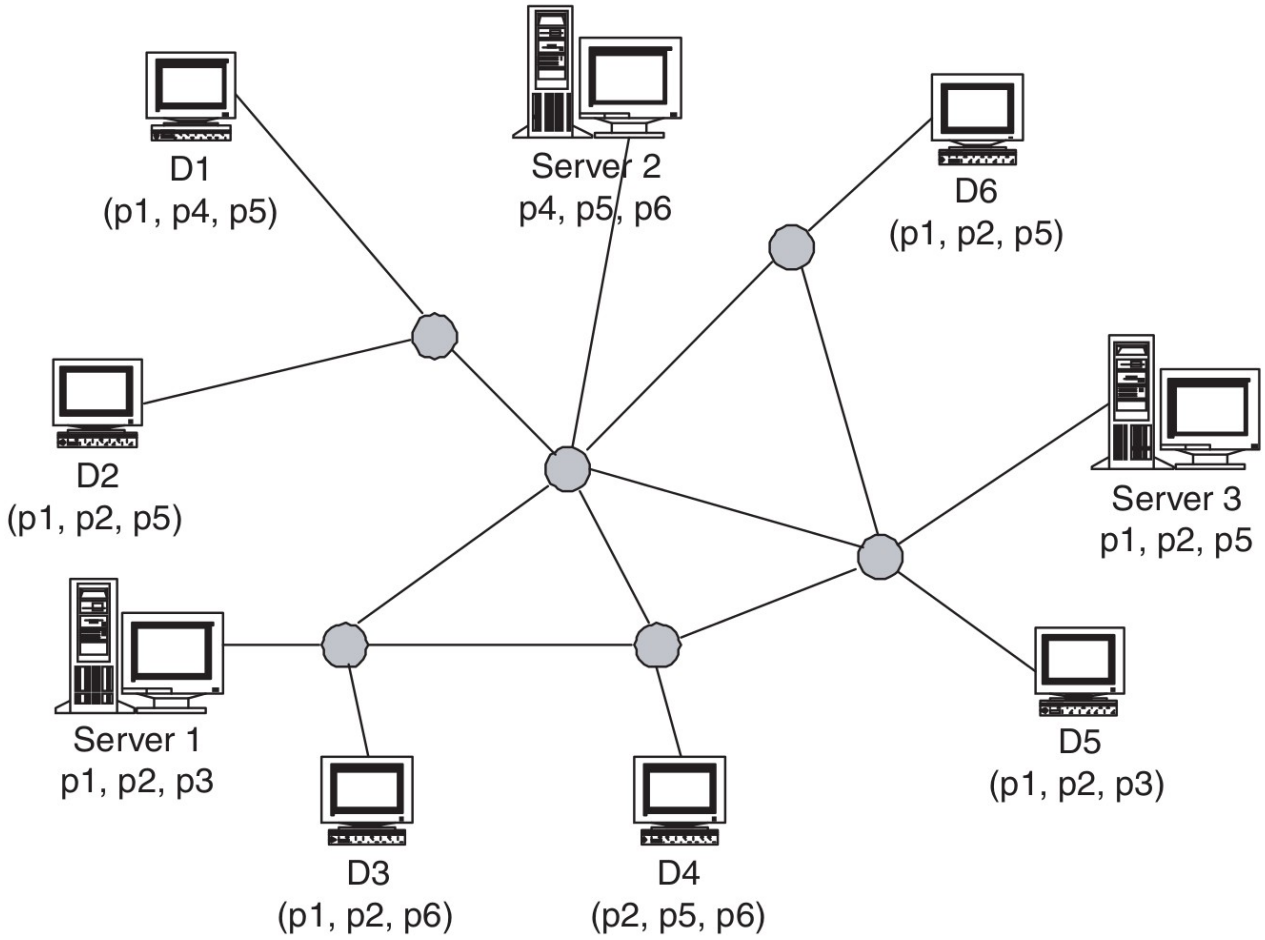


Fig. 2. Distributed Mojette Internet system with three servers and six clients.

GUÉDON, J.; PARREIN, B. & NORMAND, N.: Internet Distributed Image Information System. In: *Integrated Computer-Aided Engineering,* 8 (2001), Nr. 3, S. 205-214

RozoFS

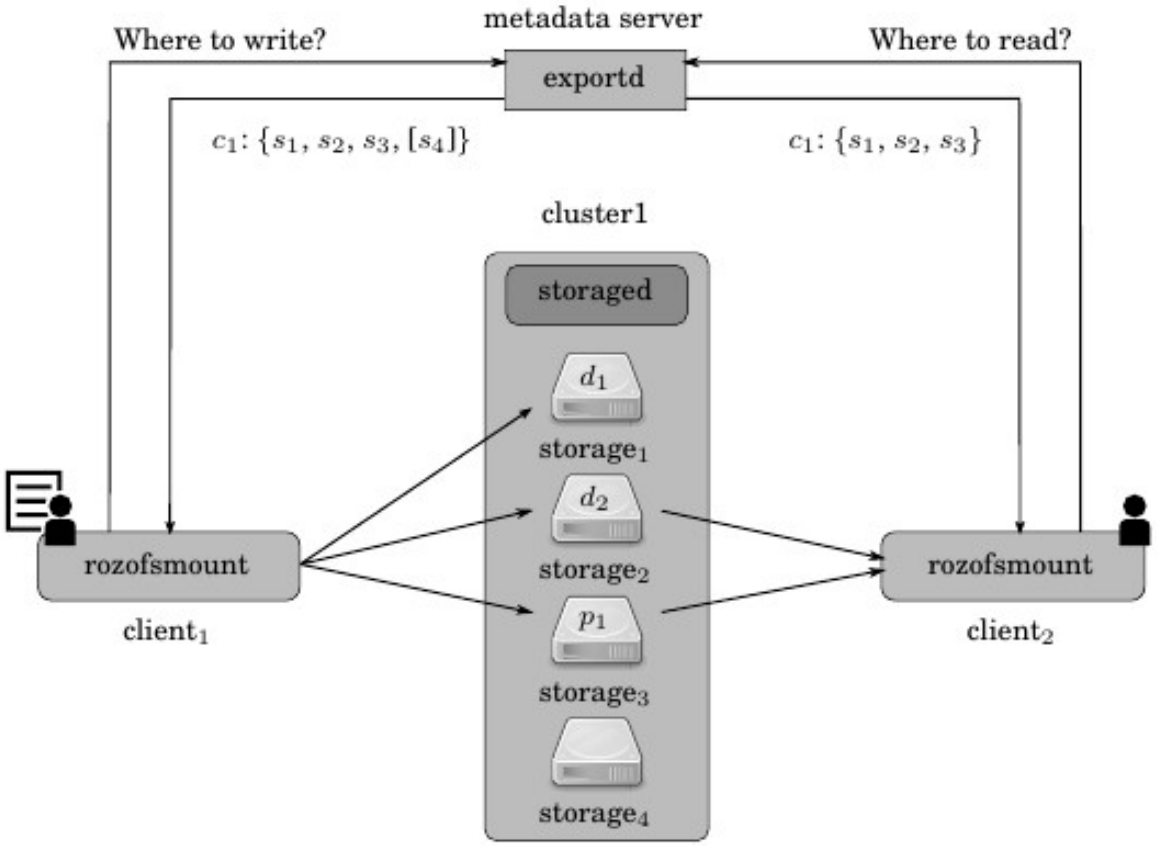
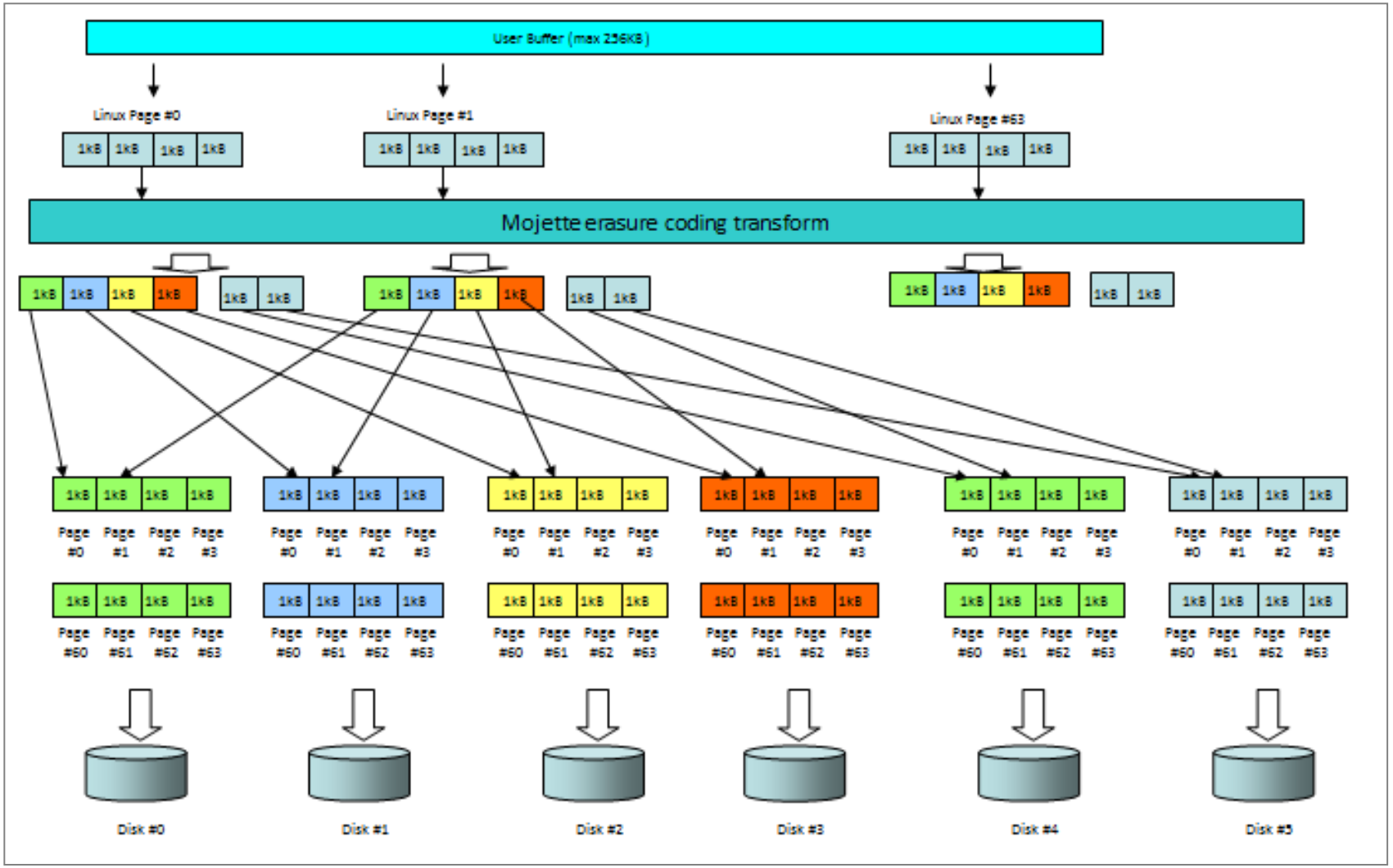


Fig. 8: RozoFS interactions during write and read operations in layout 0 (i.e. using $(3, 2)$ systematic Mojette erasure code). Given a specific request, the metadata server answers clients with a cluster id: c_j and a list of storage ids: s_i. The spare node id is depicted in [ ]. Data blocks correspond to d_k, while p_l are Mojette projections.

Pertin et. al., 2016, ACM TOS, en révision.

RozoFS



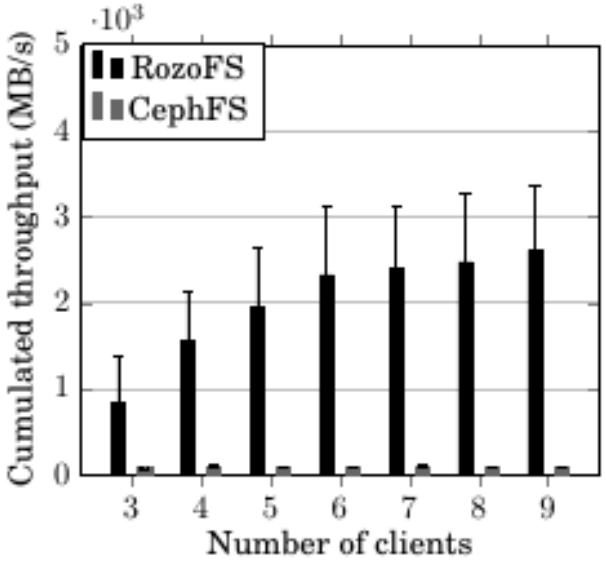
source : rozosystems

Réplication vs codes à effacement

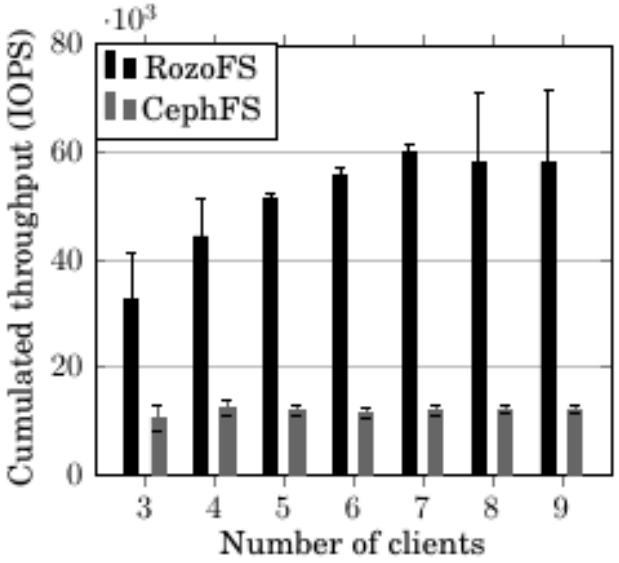
	Mojette	Reed Solomon	replica
Low disk space and network bandwidth	O	O	X
Low processing overhead (small blocks)	O	X	O
Low latency (fast algorithm)	O	X	O
parallelism	O	O	X

source : rozosystems

Performances Ceph vs Rozo



(a) Sequential write throughput (MB/s).



(b) Random write throughput (IOPS).

Fig. 9: Sequential and random write benchmarks. Performance is depicted as the cumulated throughput recorded given a growing number of clients. The workload corresponds to the writing of 100 MB files per client, with accesses of 64 KB and 8 KB for sequential and random tests, respectively.

Test réalisé sur le cluster `Econome` de Grid5000 (22 nœuds)
Disques 7200 SATA, Interco. à 10 GbE
Pertin et. al., 2016, en révision, ACM TOS

Performances Gluster (avec EC [1]) vs Rozo

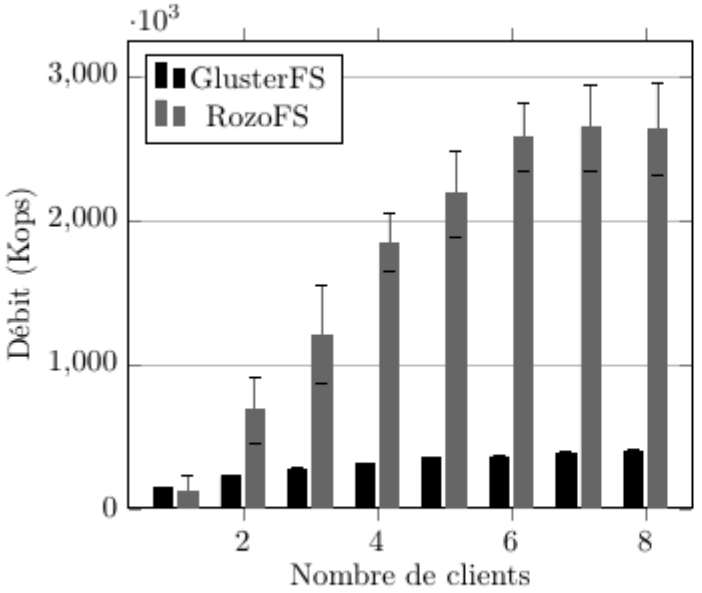


FIGURE 1 – Écriture séquentielle

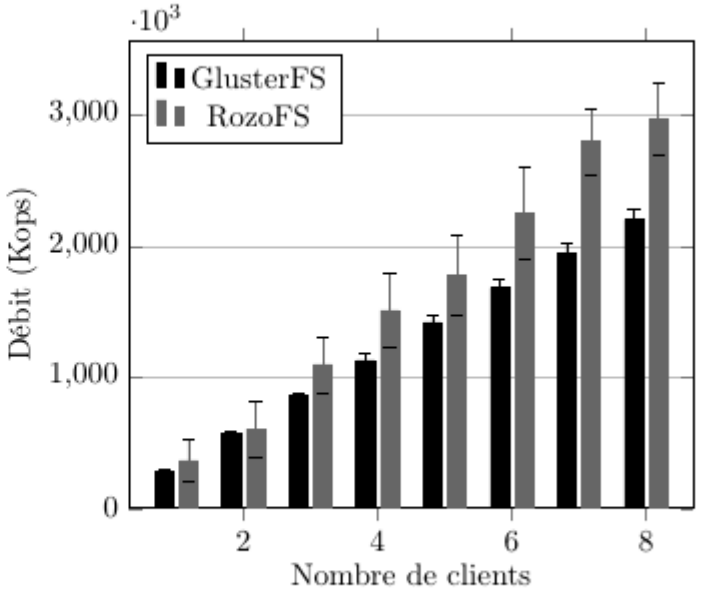


FIGURE 3 – Lecture séquentielle

Test réalisé sur le cluster `Paravance` de Grid5000 (8 nœuds)
Accès sequentiel (block size 64k)
Pertin et. al., 2016, en révision, ACM TOS

[1] https://github.com/gluster/glusterfs/tree/master/xlators/cluster/ec/src

Performances Gluster (avec EC [1]) vs Rozo

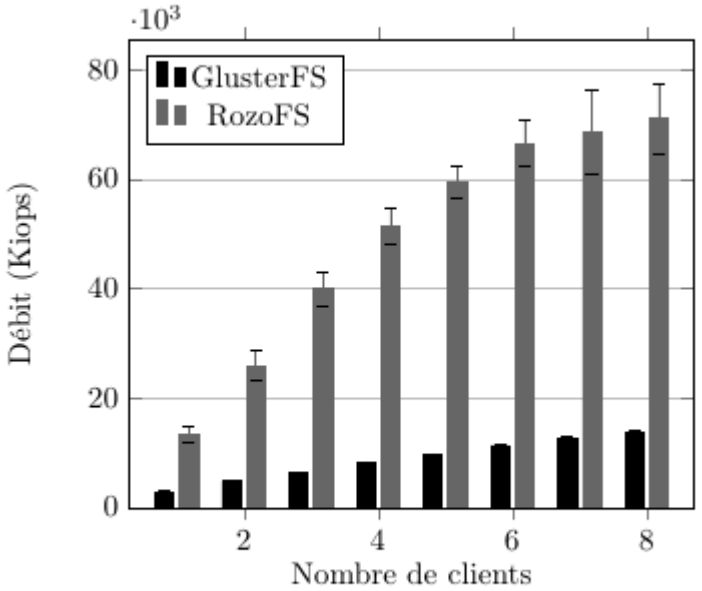


FIGURE 2 – Écriture aléatoire

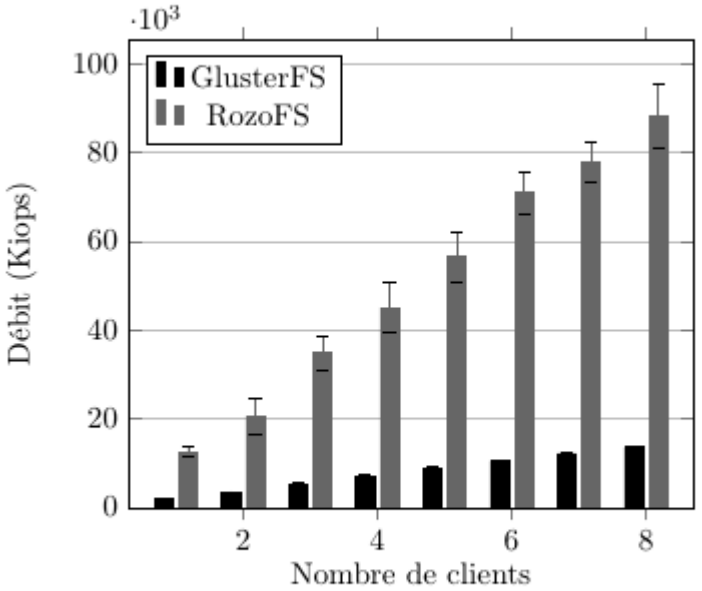


FIGURE 4 – Lecture aléatoire

Test réalisé sur le cluster `Paravance` de Grid5000 (8 nœuds)
Accès aléatoire (block size 8k)
Pertin et. al., 2016, en révision, ACM TOS

[1] https://github.com/gluster/glusterfs/tree/master/xlators/cluster/ec/src

Intérêt des codes correcteurs

(pour 27 GB de données et tolérance à 2 pannes)

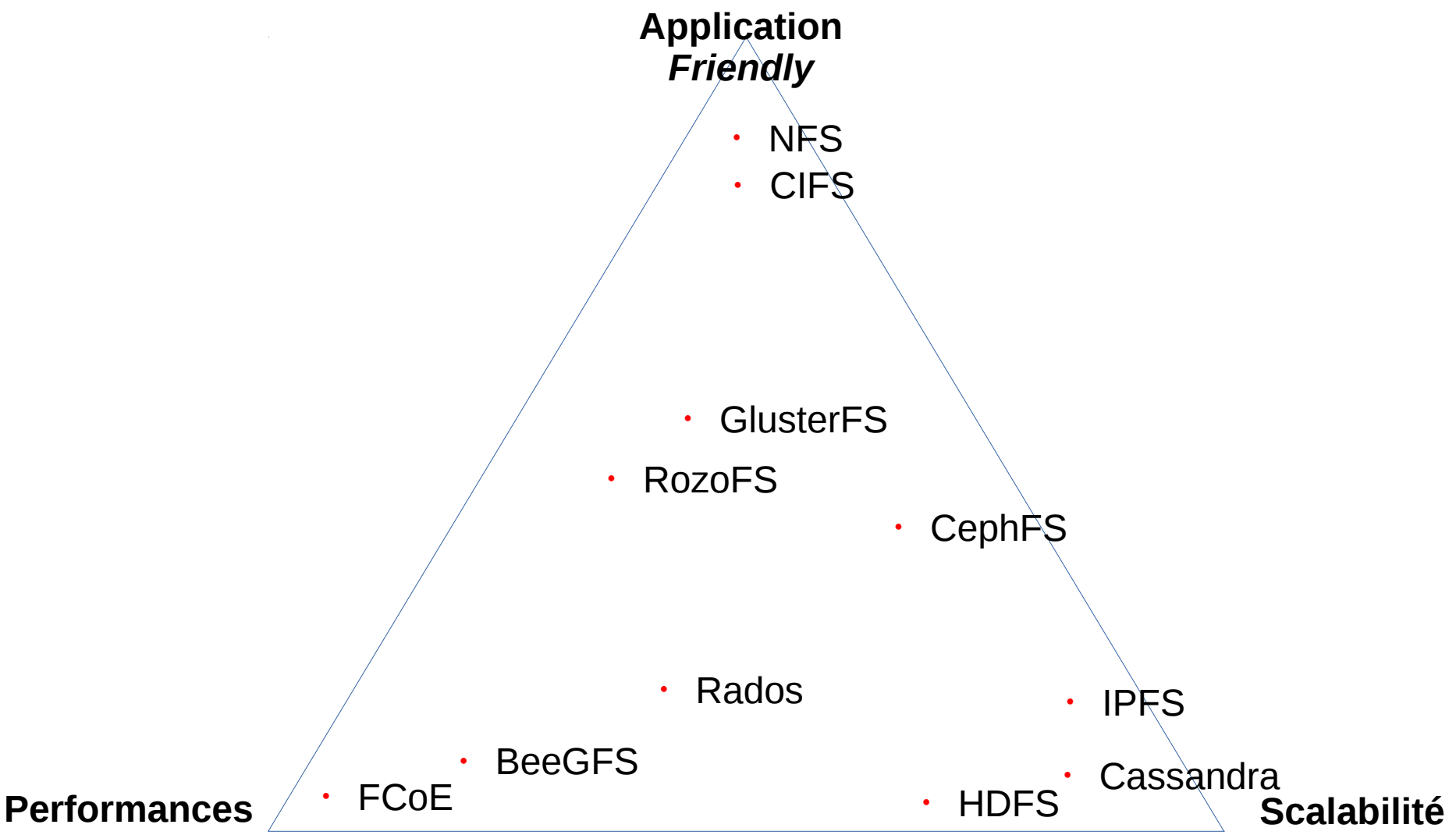
	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8	Total_{th}	Total_{obs}
RozoFS	5.2	5.2	5.1	5.1	5.2	5.2	5.2	5.2	40.5	**41.4**
CephFS	14	9.6	8.5	12	12	9.8	12	12	81	**89.9**

Table II: Evaluation of the storage consumption (in GB) for RozoFS and CephFS. Each of the 8 nodes in the cluster is depicted as s_i. These values are recorded after 9 clients wrote 100 MB files over 30 iterations. The theoretical (*th*) and observed (*obs*) total amount of stored data are given.

Pertin et. al., 2016, ACM Transactions On Storage (TOS), en révision.

[1] https://github.com/gluster/glusterfs/tree/master/xlators/cluster/ec/src

Classification (tentative)

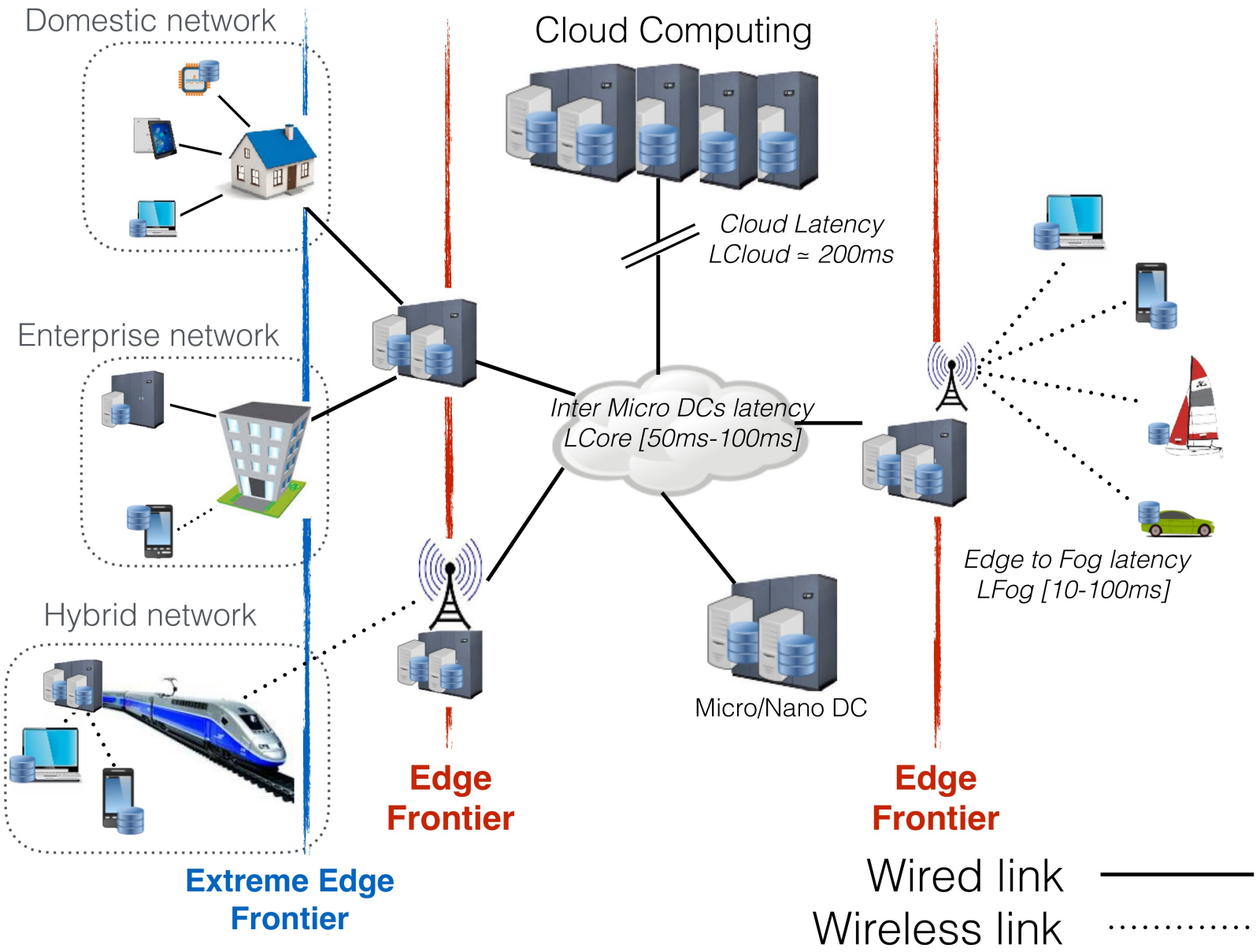


(la ressemblance avec le théorème CAP est volontaire)


JOHN CARPENTER'S
THE
FOG

(image The Fog, John Carpenter)

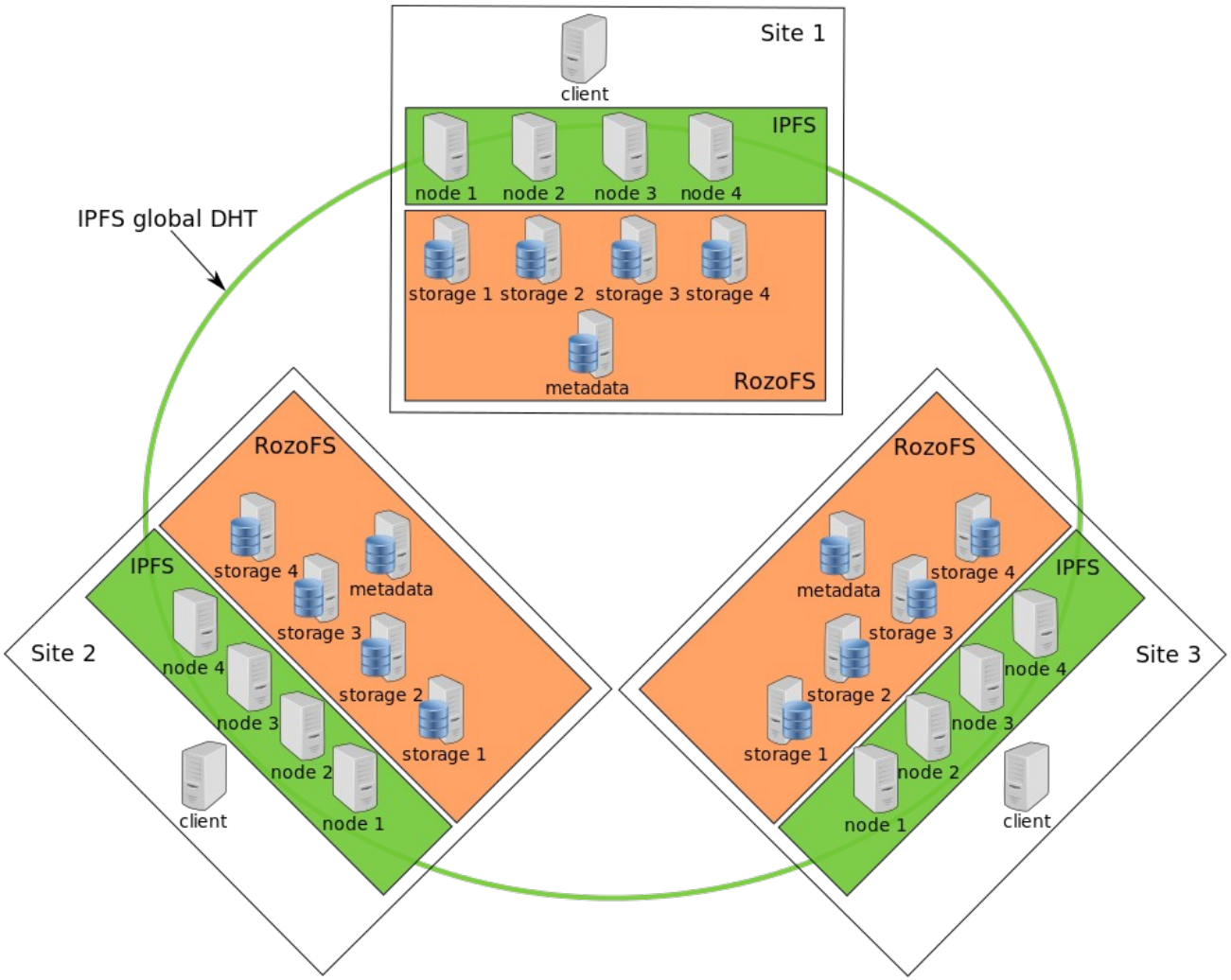
Fog Computing / Fog Networking



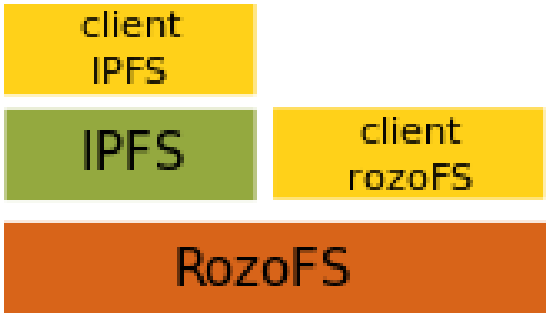
Intérêt d'une architecture Fog

- Décentralisation du modèle Cloud
- Re-localisation des données
- Sécurisation / *privacy*
- Faible latence (QoS/QoE)
- Systèmes distribués au sein d'un site *Fog*
- Optimisation locale de l'espace de stockage (déduplication, codes correcteurs, …)
- Consommation énergétique moindre

Couplage Fog et DFS



Vue topologique



Vue architecturale

Thèse en cours Bastien Confais, CNRS, 2016
Co-encadrement Adrien Lebre, INRIA