UNIVERSITE BRETAGNE LOIRE

UNIVERSITÉ DE NANTES

Thèse de Doctorat

Dimitri PERTIN

Mémoire présenté en vue de l'obtention du
grade de Docteur de l'Université de Nantes
sous le sceau de l'Université Bretagne Loire

École doctorale : Sciences et technologies de l'information, et mathématiques

Discipline : Informatique et applications, section CNU 27
Unité de recherche : Institut de Recherche en Communications et Cybernétique de Nantes (IRCCyN)

Soutenue le 22 avril 2016
ED 503

Code à Effacement Mojette pour le Stockage Distribué

JURY

Président :	**M. Imants SVALBE**, *Senior Lecturer*, Université Monash, Melbourne, Australie
Rapporteurs :	**M. Pierre DUHAMEL**, Directeur de recherche CNRS, L2S Centrale Supélec, Paris **M. Martin QUINSON**, Professeur des universités, ENS, Rennes
Examinateur :	**M. Jérôme LACAN**, Professeur, ISAE-SUPAERO, Toulouse
Invité :	**M. Évenou PIERRE**, Ingénieur, Rozo Systems, Nantes
Directeur de thèse :	**M. Nicolas NORMAND**, Maitre de conférences titulaire de l'HDR, Université de Nantes
Co-encadrant de thèse :	**M. Benoît PARREIN**, Maitre de conférences titulaire de l'HDR, Université de Nantes

Remerciements

Cette aventure a débuté à Polytech, lorsque ma curiosité m'a poussé à choisir un mystérieux sujet de recherche intitulé : "*Code à effacement ?*". Aujourd'hui, j'ai compris que ce " ?" représente toute l'étendue du sujet. Difficile de réaliser le chemin parcouru jusqu'ici. Et pourtant si j'y suis parvenu, c'est grâce aux personnes qui m'ont soutenu.

Je tiens particulièrement à remercier Imants Svalbe pour avoir suivi mes travaux avec son enthousiasme légendaire, depuis ma soutenance de projet de recherche, jusque dans son rôle de président de jury de thèse. Je fus honoré, et suis très reconnaissant à Pierre Duhamel et Martin Quinson d'avoir accepté de rapporter cette thèse. Je remercie également chaleureusement Jérôme Lacan de m'avoir accueilli à l'ISAE, et d'avoir suivi et enrichi mes connaissances de son expertise des codes, pendant ces trois ans.

Cette thèse n'aurait pu exister sans l'aide de Pierre Évenou, dont les idées et la vision m'ont beaucoup inspiré, ni sans Nicolas Normand et Benoît Parrein, compères enthousiastes et passionnés, qui ont su stimuler ma curiosité, me guider durant ces nombreuses années, et qui ont assuré dans les hauts, comme dans les bas. Merci pour ces moments que l'on a partagés, devant un tableau, comme au bistrot.

Un grand merci à l'équipe de Rozo Systems, notamment à Didier Féron, Jean-Pierre Monchanin et Sylvain David, génies de la programmation qui m'ont tant appris, ainsi que Louis le Gouriellec et Christophe de la Guérande et nos captivantes discussions. Merci également à Aurore pour notre complicité, Lukáš pour nos virées nocturnes, Alex, dit le man, pour nos galères, Josselin le stagiaire et son yoyo, Floflo et ses fruits secs, Lulu pour sa bonne humeur, Romu pour sa mauvaise humeur, JPeG et son remorqueur, Romain le surréaliste, et à l'ensemble des membres d'IVC pour la convivialité.

Je n'aurais sûrement pas débuté de travail de recherche sans Mathieu Cunche et les membres du NICTA qui ont su éveiller ma curiosité pour ce domaine. Je pense également aux membres de l'ISAE pour leur accueil, et plus particulièrement à Jonathan Detchart pour sa connaissance des codes et de la programmation. J'ai aussi beaucoup appris en échangeant avec Suayb Arslan, Andrew Kingston et Shakhar Chandra. Je remercie également José Martinez pour ses connaissances en LaTeXet son aide dans la mise en place de l'expérimentation utilisant OpenMP.

Mille pensées à mes amis, camarades de Polytech, membres du JMB et joueurs du baby pour ces bons moments, à mes frères et mes parents pour m'avoir toujours aidé.

Enfin, une pensée particulière à ma charmante Lucie, pour ses relectures, son soutien, et pour avoir fait partie des meilleurs moments durant ces trois années.

Table des matières

Introduction générale

Le nombre d'appareils interconnectés par Internet augmente de manière exponentielle. En 2003, ce nombre était largement inférieur au nombre d'habitant dans le monde (0.08 appareils par habitant en moyenne). Dans son rapport sur l'évolution d'Internet, Evans estime que ce nombre sera porté à plus de six appareils par habitant (6, 58) d'ici 2020 [Eva11]. Si cette estimation s'avère juste, le nombre total d'appareils connectés atteindra les 50 milliards. L'analyse menée par le cabinet de conseil IDC, auprès d'EMC (leader mondial des solutions de stockage), présente deux facteurs permettant d'expliquer cette augmentation [GR12] : (i) l'extension d'Internet aux objets du quotidien, désignée par le terme « Internet des objets » (IoT pour *Internet of Things*) ; (ii) l'émergence de nouveaux marchés numériques (Chine, Brésil, Inde, Russie, Mexique). Afin de supporter cette croissance, Internet s'adapte : évolution des protocoles (e.g. le passage à IPv6) et des infrastructures (e.g. construction de centres de données). Les utilisateurs aussi s'adaptent et découvrent de nouvelles applications (e.g. informatique en nuage, fouille de données). Dans ce contexte, le rôle des systèmes de stockage de données est crucial puisque cette évolution importante du nombre d'appareils s'accompagne d'une augmentation exponentielle des données générées et stockées. En particulier, le rapport d'IDC estime que la quantité de données stockées dans le monde correspondra à 44 zettaoctets[1] (i.e. 10^{21} octets) d'ici 2020 [GR12]. Parmi cette quantité massive de données, on notera que 27% seront générées par des objets connectés issus de l'IoT. La conception d'un système de stockage nécessite de considérer différents critères (e.g. capacité de stockage, tolérance aux pannes, mise à l'échelle, débits des lectures et écritures). Les systèmes centralisés (composés d'un seul serveur) présentent des limites, notamment en ce qui concerne la mise à l'échelle (les ressources du serveur sont limitées) et la tolérance aux pannes (perte de la totalité des données stockées sur le serveur). Pour satisfaire ces critères, il est en conséquence nécessaire d'utiliser des systèmes de stockage distribués. Un système distribué est défini par Tanenbaum et Steen [TS06] comme « un ensemble de serveurs indépendants, dont les utilisateurs ont une vision cohérente d'un système unique ». Les systèmes de stockage distribués offrent donc une représentation cohérente d'un volume de stockage dont les données sont réparties sur plusieurs serveurs. Dans la suite, nous utiliserons le terme « *Networked Distributed Storage System* » (NDSS) défini par Oggier

[1] Hilbert et López [HL11] rappellent que ce volume de données correspond à la quantité d'information génétique contenue dans un corps humain, mais que « par rapport à la capacité de la nature à traiter l'information, la capacité du monde numérique évolue de façon exponentielle ».

et Datta [OD12] pour insister sur l'interconnexion des supports de stockage à travers un réseau (e.g. bus, ethernet). En fonction de l'application qui travaille sur les données du NDSS, certains critères ont besoin d'être favorisés (pour des raisons économiques). Ainsi, nous allons voir quatre exemples importants d'application qui nécessitent de grands volumes de stockage :

- les services multimédias tels que la vidéo à la demande nécessitent de grandes quantité de données. Par exemple, Netflix dispose pas moins de 40 pétaoctets (i.e. 10^{15} octets) de contenus vidéos sur Amazon S3 [Hun14]. Cette application privilégie la mise à l'échelle afin de supporter un pic de connexions (lors de la sortie d'un nouveau contenu vidéo par exemple) ;

- le « *Big Data* » concerne la fouille et le traitement analytique d'une quantité massive et non structurée de données. Les moteurs de recherche par exemple, doivent gérer une quantité de données théoriquement limitée par l'échelle d'Internet. Pour adresser ce problème, Google a développé son propre système de fichiers distribué, *Google File System* (GFS) [GGL03], ainsi que le modèle de programmation *MapReduce* [DG08]. Ces outils permettent d'extraire des relations entre différents types de contenus, tels que des données collectées sur les pages web, les contenus générés par les utilisateurs, ou encore les données proposées par les différents services Google (e.g. maps, shopping) ;

- le calcul à haute performance (HPC, pour *High Performance Computing*) traite le cas d'importantes quantités de données structurées. Par exemple, Zwaenepoel [Zwa15] adresse le problème de l'optimisation du traitement d'analyse d'un graphe très large (i.e. 32 milliards de sommets, et 10^{12} arêtes) par un système distribué constitué de seulement 32 serveurs. Ce type d'application nécessite principalement de hauts débits en lecture et écriture ;

- l'archivage de données en revanche ne possède pas de contraintes fortes sur les débits. Toutefois, cette application a besoin de NDSS avec d'importantes capacités de stockage, et, qui soient tolérants aux pannes. Par exemple, Amazon Glacier fournit un : « service de stockage sécurisé, durable et à très faible coût pour l'archivage (...) de données rarement consultées et pour lesquelles un délai d'extraction de plusieurs heures reste acceptable »[2].

Problèmes du stockage distribué identifiés

Nous avons vu précédemment des applications qui répondent à des problèmes différents. L'offre des systèmes de stockage est en conséquence fragmentée afin de favoriser certains des critères énoncés. Par exemple, les délais garantis par Amazon Glacier ne sont pas appropriés pour gérer des données sur lesquelles seront exécutés des traitements HPC. À l'inverse, le coût d'une grappe de calcul est beaucoup trop élevé pour y archiver

[2] https ://aws.amazon.com/fr/glacier/

des données. On peut ainsi différencier deux types de données : (i) les données froides qui correspondent à du contenu peu accédé (utilisé typiquement dans les applications d'archivage où les données sont écrites une fois pour être sollicitées à l'occasion) ; (ii) les données chaudes, qui à l'inverse sont fréquemment sollicitées (typiquement le cas des applications HPC qui mettent en jeu plusieurs milliers d'entrées/sorties à la seconde). Les administrateurs des systèmes de stockage sont souvent contraints de définir deux systèmes de stockage différents (un système coûteux pour le traitement intensif, l'autre bon marché pour archiver des données). **Dans ce formalisme, notre premier problème consiste alors à concevoir un système de stockage capable de gérer aussi bien les données froides, que les données chaudes**.

Les applications citées précédemment peuvent interagir avec une quantité massive de données (de l'ordre du pétaoctet par exemple). Il est donc nécessaire de concevoir des systèmes de stockage capables de supporter une charge de plus en plus importante. Les besoins de l'application peuvent également évoluer dans le temps, et nécessiter moins de capacité de stockage. L'approche verticale (*scale-up*) consiste à migrer les données vers des supports de stockage de plus grandes capacités, avant que la capacité du système de stockage ne soit atteinte. Cette approche n'est ni flexible (limite de la taille des ressources) ni économique (requiert d'acheter du matériel récent). **Notre second problème consiste alors à mettre en place un système de stockage capable de gérer dynamiquement les ressources de stockage**.

Les systèmes de stockage sont sujets à des défaillances inévitables. Ces défaillances entraînent l'inaccessibilité temporaire, voire la perte définitive, de blocs de données [For+10]. En particulier, la probabilité d'apparition des pannes augmente avec la taille du système de stockage (e.g. défaillance d'un disque, défaillance réseau). Il est donc nécessaire d'intégrer de la redondance dans le système de stockage. La solution classique pour supporter ces pannes consiste à exploiter la nature des NDSS en distribuant plusieurs copies des blocs de données sur des supports de stockage différents. Cette méthode permet d'accéder à la copie d'un bloc lorsque les autres ne sont pas disponibles. Bien que simple à mettre en œuvre, chaque copie générée ajoute un surcout de redondance de 100%. Cette méthode implique alors un coût de stockage important. **Notre troisième problème consiste à garantir un seuil de redondance permettant au NDSS de supporter les pannes, tout en minimisant cette quantité de redondance**.

Une fois qu'un seuil de redondance est mis en place dans le NDSS, certaines pannes entraînent une perte de données qui se traduit par une réduction de la quantité de redondance. **Notre quatrième problème sera de déterminer un moyen efficace afin de rétablir et maintenir un seuil de redondance**. Pour résumer, les quatre problèmes identifiés dans cette section visent à concevoir un NDSS capable :

1. d'être capable de délivrer de très hauts débits de lecture et d'écriture ;
2. d'adapter dynamiquement ses ressources de stockage ;
3. d'être tolérant aux pannes, tout en minimisant la quantité de redondance ;
4. de maintenir cette redondance dans le temps.

Notre approche

Afin de minimiser la redondance, notre approche sera d'utiliser des codes à effacement. Cette méthode permet de réduire considérablement la quantité de redondance générée par rapport aux techniques de réplication (typiquement d'un facteur 2) [WK02 ; OD12 ; CPK14]. En particulier les codes optimaux (dits MDS pour *Maximum Distance Separable*) minimisent la quantité de redondance nécessaire pour protéger les données. Dans cette thèse, nous comparerons régulièrement nos codes aux codes MDS. Parmi eux, les codes de Reed-Solomon sont largement utilisés [RS60]. Ils sont ainsi intégrés dans plusieurs DFS tels que CephFS [Wei+06] ou DiskReduce[3] [Fan+09]. Des fournisseurs de services en nuage tels que Microsoft Azure [Hua+12] ou Openstack avec Swift [LG14] l'utilisent également. L'utilisation des codes de Reed-Solomon dans les systèmes de stockage s'est démocratisée avec le développement de bibliothèques qui en fournissent des implémentations. En particulier, Ceph et Swift ont intégré la bibliothèque ISA-L (*Intelligent Storage Acceleration Library*) d'Intel® Corporation [Int15].

En revanche, les codes à effacement impliquent une complexité significative due aux opérations d'encodage et de décodage. Dans le contexte de stockage, ces opérations sont déclenchées respectivement à l'écriture et à la lecture de blocs de données. En conséquence, leur utilisation est généralement limitée aux systèmes qui stockent des données froides. Il est toutefois possible de réduire l'impact de ces opérations en utilisant le code sous une forme systématique. Sous cette forme, les données encodées contiennent deux parties : (i) une première partie contient exactement le message à transmettre ; (ii) la seconde partie correspond à des données de redondance. Cette forme permet notamment de ne pas avoir à décoder l'information lorsque la partie contenant le message est disponible. En conséquence, sous cette forme, les débits en lecture et écriture sont plus importants. La conception d'un code sous cette forme sera le sujet d'une de nos contributions dans ce travail de thèse.

Pour la conception d'un NDSS, nous proposons d'utiliser une approche horizontale (*scale-out*). Cette approche consiste à composer un ensemble flexible de serveurs de stockage (on parle de grappe), dont on peut adapter le nombre. Il est ainsi possible d'augmenter ou de réduire la capacité du système. Cette approche est en conséquence plus flexible et plus économique que l'approche *scale-up* [OD12]. En particulier, nous proposons d'utiliser RozoFS : un logiciel de système de stockage (ou SDS ou *Software-Defined Storage*). Plus précisément, RozoFS est un système de fichiers distribué (ou DFS pour *Distributed File System*), ce qui correspond à un NDSS permettant d'interagir avec des fichiers. Notons que d'autres représentations de la donnée existent telles que la forme en blocs (interface proposée par les disques) ou en objets (interface au cœur du DFS Ceph [Wei+06]). En particulier, RozoFS permet d'agréger l'espace disponible depuis un ensemble de supports de stockage. Cette agrégation est exposée à l'utilisateur sous la forme d'un volume de stockage organisé par une arborescence (fichiers et répertoires). En particulier, RozoFS est orienté pour les réseaux locaux (*LAN*) rapides (très haut

[3] DiskReduce est une modification de *Hadoop Distributed File System* (HDFS) qui intègre les codes de Reed-Solomon. HDFS est un DFS open-source basé sur *Google File System* (GFS) [GGL03].

débit, faible latence). Notre approche se distingue alors des architectures pair-à-pair (ou P2P pour *Peer-to-Peer*) et du stockage sur réseaux étendus (*WAN*) de latences plus importantes. RozoFS correspond ainsi à une couche de virtualisation capable d'exploiter du matériel varié et bon marché. Cette solution est alors plus flexible et moins onéreuse. En particulier, RozoFS est un logiciel libre sous licence GNU GPLv2[4] dans lequel il nous sera possible d'intégrer nos contributions.

Contributions

Nous proposons une approche basée sur la géométrie discrète, dans l'objectif de proposer une nouvelle représentation du problème de correction des effacements. En particulier, cette nouvelle approche permet l'élaboration de nouveaux algorithmes pour le codage à effacement. Dans cette optique, nous proposons d'étudier la transformation de Radon qui est une application mathématique pouvant être utilisée afin de représenter des données de manière redondante. Des travaux sur une version discrète de cette transformation, la « transformation de Radon finie » (ou FRT pour *Finite Radon Transform*) ont déjà permis de concevoir un code à effacement MDS [Nor+10]. Dans ce travail de thèse, nous proposons l'utilisation d'une autre version discrète de cette application : la transformation Mojette [GBB95]. Il a notamment été établi que la transformée Mojette dispose d'un algorithme de reconstruction itératif, permettant de reconstruire les données avec une complexité linéaire [NKÉ06]. La conception d'un code systématique, basé sur cette transformée, est motivée par l'objectif de bénéficier d'un code fournissant de bonnes performances en encodage et en décodage. Cette motivation est étendue à l'intégration du code dans RozoFS, afin de fournir un DFS capable de fournir de bonnes performances en lecture et écriture, tout en tolérant les pannes avec une quantité minimum de redondance (code MDS). Ce travail de thèse a ainsi conduit aux trois contributions suivantes :

1. la conception et l'analyse d'une version systématique du code à effacement Mojette, permettant d'accélérer les performances du code, et d'en réduire la quantité de redondance nécessaire ;
2. l'intégration de ce code Mojette systématique au niveau de la distribution des données dans RozoFS, dans l'objectif que celui-ci puisse gérer les données chaudes, tout en limitant le coût de la redondance ;
3. la conception d'une méthode distribuée pour ré-encoder cette redondance, sans avoir à reconstruction la donnée initiale, afin de pouvoir rétablir un seuil de redondance au sein du NDSS.

Plan du manuscrit

Les travaux de cette thèse sont organisés en deux parties qui comportent chacune trois chapitres. La première partie couvre la conception de nouveaux codes à effacement en

[4] Le projet GitHub de RozoFS est accessible à l'adresse suivante : https://github.com/rozofs/rozofs

utilisant conjointement la théorie de l'information et la transformée de Radon discrète. Les trois chapitres qui le composent présentent les éléments suivants :

1. dans le chapitre 1, nous introduisons des notions de la théorie de l'information nécessaires afin d'établir un état de l'art des codes à effacement. Ces notions vont nous permettre de dresser une liste de critères nécessaires afin de comparer les codes abordés dans ce manuscrit. Nous verrons ainsi quelques exemples de codes à effacement (MDS et non-MDS) ;

2. le chapitre 2 introduit la transformation de Radon. Ce chapitre utilise conjointement la géométrie discrète et la théorie des codes. La géométrie discrète permettra de définir deux versions discrètes de la transformation de Radon : la FRT et la transformation Mojette. La théorie des codes sera nécessaire pour concevoir et comprendre les propriétés des codes à effacement basés sur ces transformations. Nous verrons ainsi que la FRT donne un code MDS, tandis que la transformation Mojette dispose d'un algorithme de décodage itératif efficace ;

3. la première contribution majeure est énoncée dans le chapitre 3. Cette contribution correspond une nouvelle conception du code à effacement Mojette sous sa forme systématique. Cette conception a des avantages sur le rendement du code et permet de tendre d'avantage vers un code MDS. Dans un deuxième temps, un algorithme de décodage adapté à cette forme est donné. Nous évaluerons ainsi la quantité de redondance générée par cette nouvelle forme par rapport à la version classique et au cas MDS. Cette évaluation permet de mettre en évidence le rendement quasi-MDS du code conçu.

La première partie ayant permis la conception d'un code à effacement Mojette performant, la seconde s'intéresse à son intégration dans le contexte des systèmes de stockage distribués. Dans cette partie, les deux premiers chapitres mettent respectivement en avant l'utilisation du code à effacement Mojette dans une architecture de stockage distribué, puis spécifiquement dans RozoFS. Le troisième chapitre tente de répondre au problème du maintien d'un seuil de redondance, dans un système dont les données se dégradent au cours du temps. Plus particulièrement, les différents chapitres comportent les éléments suivants :

1. le chapitre 4 présente une analyse théorique et expérimentale des performances du code Mojette dans le contexte du stockage distribué. Les métriques utilisées (nombres d'opérations à l'encodage et au décodage, nombre de blocs impactés par la mise à jour de données) mettent en avant la simplicité algorithmique du code Mojette par rapport à d'autres codes (e.g. codes de Reed-Solomon). En particulier, une mesure des latences en encodage et décodage des implémentations du code Mojette est donnée. Dans les conditions de nos tests, notre nouvelle conception systématique permet de réduire par trois les temps d'encodage par rapport à la forme classique. De plus, aucun décodage n'est nécessaire en lecture, dans le cas où aucune panne ne survient. En particulier, ces mesures montrent également que

notre implémentation est plus performante que l'implémentation des codes de Reed-Solomon fournit par la bibliothèque de référence : ISA-L ;

2. la mise en œuvre et l'intégration du code à effacement Mojette dans le système de fichiers distribué RozoFS est décrite dans le chapitre 5. Une évaluation menée sur la plate-forme Grid'5000 permet de montrer que dans le cadre de nos tests, RozoFS est capable de fournir de meilleures performances que des systèmes basés sur de la réplication (e.g. CephFS), tout en réduisant d'un facteur 2 le volume total stocké ;

3. notre contribution sur le rétablissement du seuil de redondance du NDSS est fournie dans le chapitre 6. Cette contribution concerne la conception d'une méthode distribuée afin de calculer de nouveaux symboles de mots de code. Cette méthode peut être utilisée afin de maintenir le système de stockage à un niveau de redondance désiré. Une évaluation est réalisée afin de mettre en avant le bénéfice de la distribution des calculs.

Dans une dernière partie, nous aborderons la conclusion des travaux présentés dans cette thèse, ainsi que la perspective des futurs travaux de recherche.

Contexte de la thèse

Dans le cadre d'une convention CIFRE, ces travaux de recherche ont été menés conjointement au sein de l'équipe Image et Vidéo Communications (IVC) de l'Institut de Recherche en Communications et Cybernétique de Nantes (IRCCyN), et au sein de l'entreprise Rozo Systems. Cette entreprise développe notamment le système de fichiers distribué RozoFS, qui sera largement abordé dans ce manuscrit.

Une partie de ce travail de recherche a également été financée par le projet ANR FEC4Cloud (appel Emergence 2012). Ce projet a pour objectifs d'analyser et de concevoir des codes à effacement pour le stockage distribué. Les partenaires de ce projet sont l'IRCCyN (coordinateur), Supaero-ISAE et la SATT Ouest Valorisation.

Codes à effacement en géométrie discrète

Introduction de la partie

Les systèmes d'information nécessitent l'ajout de données de redondance au message à transmettre afin que le destinataire puisse reconstituer l'information lorsque celle-ci est détériorée. Dans cette partie, nous nous intéressons d'une part à minimiser la quantité de redondance générée, et, d'autre part, à optimiser les performances du code. L'objectif ici est de concevoir de nouveaux codes à effacement qui se basent sur des transformées discrètes afin de fournir deux choses : un rendement optimal (code MDS), et de nouveaux algorithmes d'encodage et de décodage de faible complexité.

Pour cela, les bases mathématiques de la théorie des codes sont introduites dans le chapitre 1 à travers l'étude des travaux fondamentaux de Shannon [Sha48]. En particulier, nous élaborons une liste de critères comme base pour la comparaison des différents codes présentés dans ce manuscrit. Le chapitre 2 décrit notre approche pour concevoir des codes à effacement à partir de versions discrètes de la transformation de Radon. Le chapitre 3 décrit notre première contribution correspondant à l'élaboration d'une construction du code à effacement Mojette sous une forme systématique. Cette construction vise à réduire la complexité du code, ainsi que la quantité de redondance générée. À l'issue de cette étude, nous présentons une évaluation de cette réduction de redondance par rapport à la version non-systématique et aux codes MDS.

1

Principe et évaluation des codes à effacement

Sommaire

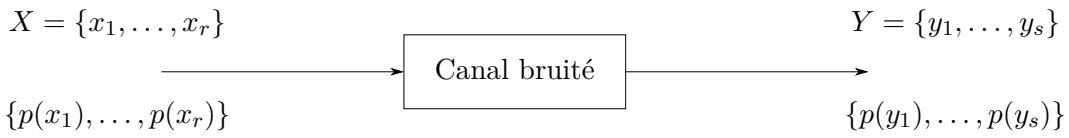
Introduction du chapitre

Les systèmes de communication numérique permettent de faire transiter l'information d'un émetteur vers un destinataire à travers un canal de communication. On considère qu'une transmission est fiable lorsque le destinataire accède à l'information en un temps acceptable. Pour cela, il est souvent nécessaire d'améliorer les débits. Le schéma classique d'une telle transmission repose sur deux étapes. La première consiste à compresser l'information afin d'améliorer les débits. Pour cela, la redondance dans l'information est supprimée (codage source). En revanche, sans ces données de redondance, le message est particulièrement sensible au bruit présent sur le canal. Afin de tolérer une dégradation de l'information, la seconde étape consiste à rajouter de la redondance (codage canal). Apparaissant comme une alternative au codage conjoint source/canal [DR97], ce schéma, engendré par le « théorème de séparation », introduit par Shannon dans les années 1950, considère chaque étape d'encodage indépendamment. Dans ce travail de thèse, nous nous intéresserons uniquement au codage canal. En particulier, nous nous focaliserons sur les dégradations qui engendrent la perte d'une partie de l'information (appelé « effacement »). Nous verrons précisément en quoi la correction d'effacements se distingue de la correction des valeurs des données reçues.

Se pose alors la question du rendement du code. Par exemple, il est possible de générer de la redondance en répétant plusieurs fois le bloc d'information à transmettre. Une quantité suffisante de répétitions permet de n'obtenir aucune erreur au décodage (ce qui revient à supprimer le bruit du canal). Toutefois, l'augmentation du nombre de répétitions entraine la diminution de la part d'information utile dans la transmission. Désigné comme le « rendement » du code, ce taux correspond au rapport entre le nombre de blocs utiles sur le nombre de blocs transférés. En définissant la notion d'entropie, Shannon est parvenu a établir un théorème qui pose les limites du codage canal [Sha48]. Ce théorème démontre que le rendement d'un code possède une valeur limite C, appelée « capacité » du canal, jusque laquelle la probabilité d'erreur est nulle. Ainsi, une transmission réalisée avec un rendement supérieur à C ne peut pas être fiable. Bien que ce théorème montre qu'il existe des codes permettant de corriger les erreurs (à condition que leur rendement soit inférieur à C), on ne savait pas comment les construire.

Dès lors, la conception des codes capables d'atteindre la capacité du canal a été un enjeu important de la théorie des codes. Par exemple, les codes *Low-Density Parity-Check* (LDPC) permettent en théorie d'atteindre cette limite, de manière asymptotique [Gal62]. En pratique (i.e. sur des codes de longueur finie) le constat est différent. Soit l'algorithme de décodage associé est efficace, mais ne permet pas d'atteindre cette limite (décodage itératif), soit il s'en approche, mais au prix d'une complexité algorithmique significative (décodage par maximum de vraisemblance). Nous verrons que les codes ne sont pas égaux, dans la mesure où leurs propriétés dépendent de leurs conceptions.

En 1950, Hamming conçoit le premier code correcteur. Son objectif était que son calculateur à carte perforée puisse finir un long traitement durant le week-end, malgré la présence de *bugs*. Ses travaux fondamentaux comportent notamment la définition de la notion de distance d'un code [Ham50]. Cette notion permet d'évaluer la capacité de



$$P(X|Y) = \{p(x_1, y_1), \ldots, p(x_r, y_1), \ldots, p(x_r, y_s)\}$$

FIGURE 1.1 – Représentation du modèle d'un canal de communication. X et Y représentent des variables aléatoires. Chaque élément de ces variables a une probabilité d'apparition notée respectivement $p(x_i)$ et $p(y_i)$. Une loi de transition $P(X|Y)$ définit les probabilités $p(x_i|y_i)$ de recevoir le symbole y_i sachant que x_i a été envoyé.

correction d'un code. Toutefois, le code qu'il parvient à concevoir permet seulement de corriger une erreur. De la notion de distance, découle la définition du concept de codes MDS (pour « *Maximum Distance Separable* »). Ce terme est utilisé pour qualifier les codes qui fournissent une quantité de redondance minimale relativement à une capacité de correction fixée (la définition mathématique de cette notion sera donnée au cours du chapitre). Depuis l'introduction de la notion de distance, la théorie des codes algébriques s'est largement développée. En particulier, l'ensemble des corps finis $\mathbb{F}$ fournit une structure algébrique adaptée aux codes en blocs. Ce type de codes travaille sur des blocs de données de taille fixe. REED et SOLOMON [RS60] introduisent une famille importante de codes en 1960. Ces code MDS ont la particularité de pouvoir corriger un nombre arbitraire d'erreurs. Ils remplacent donc avantageusement le code de HAMMING, et sont notamment utilisés par la NASA afin de garantir la fiabilité des communications dans le cas des missions spatiales. Bien qu'ils soient encore présents dans de nombreuses applications aujourd'hui, ils sont souvent utilisés conjointement avec les codes de convolution. Introduits par ELIAS en 1955, ces codes forment une alternative aux codes par blocs. Alors que ces derniers découpent le message en blocs de symboles de taille fixe, les codes convolutifs appliquent une fenêtre glissante sur le flux de données à transmettre, et produisent une séquence continue de symboles encodés.

Ce chapitre a pour objectif de définir un état de l'art de la théorie des codes linéaires par blocs. Nous introduirons dans la section 1.1 les concepts mathématiques de la théorie de l'information, telle que présentée dans les travaux de SHANNON [Sha48], avant de nous intéresser à la théorie algébrique des codes, définie dans les travaux de HAMMING [Ham50]. Les propriétés des codes à effacement seront ensuite définies dans la section 1.2, ce qui nous permettra de proposer une liste de critères permettant de les comparer. La section 1.3 présentera une étude des principaux codes à effacement linéaires en blocs (e.g. REED-SOLOMON, LDPC) à travers l'analyse de ces critères.

1.1 Notions de théorie des codes

Dans cette section, nous verrons dans un premier temps la théorie mathématique de l'information, introduite par SHANNON [Sha48]. Cette étude sera le sujet de la section 1.1.1.

Les notions d'entropie, d'information mutuelle et de capacité du canal y seront traitées. Nous verrons par la suite l'exemple du canal binaire symétrique et du canal à effacement dans la section 1.1.2. La section 1.1.3 présentera la théorie algébrique des codes correcteurs, définie par Hamming [Ham50]. Nous y définirons notamment les opérations d'encodage, de décodage, ainsi que les caractéristiques des codes à effacement.

1.1.1 Théorie mathématique de l'information

Un canal de communication est un support de transmission d'information permettant d'acheminer une information depuis un émetteur vers un destinataire. La figure 1.1 de la page 23, représente la modélisation d'un canal. Un canal $C\,[X, Y, P(X|Y)]$ est défini par :

1. Un alphabet d'entrée $X = \{x_1, \ldots, x_r\}$, de cardinal r, représentant la source ;
2. Un alphabet de sortie $Y = \{y_1, \ldots, y_s\}$, de cardinal s, représentant la destination ;
3. D'une loi de transition $P(X|Y)$ qui peut être représentée par la matrice stochastique suivante :

$$P(Y|X) = \begin{pmatrix} P(x_1, y_1) & P(x_1, y_2) & \cdots & P(x_1, y_s) \\ P(x_2, y_1) & P(x_2, y_2) & \cdots & P(x_2, y_s) \\ \vdots & \vdots & \ddots & \vdots \\ P(x_r, y_1) & P(x_r, y_2) & \cdots & P(x_r, y_s) \end{pmatrix}. \tag{1.1}$$

Une source d'information $\mathcal{S}$ est définie par un couple $\mathcal{S} = (A, P)$ où A est un alphabet $A = \{s_1, \ldots, s_{|A|}\}$ et P est une distribution des probabilités sur A, c'est à dire que p_i correspond à la probabilité d'apparition de s_i lors d'une transmission. Dans un document rédigé en français par exemple, la probabilité d'apparition de la lettre « e » est plus importante que pour les autres lettres de l'alphabet. On dit que la source est « sans mémoire » lorsque l'apparition de s_i est indépendant de s_j pour $i \neq j$, et que leur probabilité n'évolue pas pendant la transmission. Dans le cas de notre modélisation, chaque élément de $x_i \in X$ a une probabilité d'apparition $p(x_i)$. Soit $p_{x_i|y_i} = P(X = x_i, Y = y_i)$, la probabilité de recevoir le symbole y_i sachant la valeur du symbole émis x_i. Par exemple, sur un canal sans bruit, $p_{x|x} = 1$ et $p_{x|y} = 0$ pour $x \neq y$.

Dans la section suivante, nous étudierons deux canaux particuliers : le canal binaire symétrique, et le canal à effacement. Dans le reste de cette section, nous allons définir les différentes caractéristiques d'un canal, qui nous seront utiles pour la suite. En particulier, nous définirons l'entropie, l'information mutuelle et la capacité d'un canal. L'ensemble de ces définitions est introduit dans les travaux de Shannon [Sha48].

Entropie

Pour pouvoir définir l'entropie, il faut introduire au préalable le concept de « degré d'originalité ». Ce degré d'originalité $I(s_i)$ (parfois appelé « auto-information ») est défini comme l'information transmise par le symbole s_i (en bit) et vaut [Sha48, p. 1] :

$$I(s_i) = -\log_2(p_i). \tag{1.2}$$

En conséquence, un symbole qui apparaît souvent ne véhicule que peu d'information par rapport à un symbole qui apparaît très rarement. L'entropie $H(\mathcal{S})$ d'une source correspond à la valeur moyenne du degré d'originalité de l'ensemble des symboles de l'alphabet. Elle est ainsi définie comme [Sha48, p. 11] :

$$\begin{aligned} H(\mathcal{S}) &= \mathbb{E}\left[I(s_i)\right] \\ &= -\sum_{x=1}^{r} p_x \log_2\left(p_x\right) = \sum_{x=1}^{r} p_x \log_2\left(\frac{1}{p_x}\right), \end{aligned} \tag{1.3}$$

où $\mathbb{E}$ correspond à l'espérance mathématique. L'entropie $H(\mathcal{S})$ s'exprime ici en bits par symbole. Il s'agit d'un concept fort, puisqu'il donne une mesure de la quantité minimale d'information nécessaire afin de représenter la donnée sans perte, ce qui correspond à une mesure de l'incertitude des informations provenant de la source. Deux cas particuliers peuvent être identifiés. Premièrement, si $p(s_i) = 1$ et $p(s_j) = 0$ pour $i \neq j$, alors $H(\mathcal{S}) = 0$, c'est à dire que le résultat est connu d'avance (aucune incertitude sur la valeur reçue). À l'inverse, si tous les symboles sont équiprobables $p(s_i) = \frac{1}{|A|}$, alors $H(\mathcal{S}) = \sum_{i=1}^{|A|} \frac{1}{|A|} \log_2 |A| = \log_2 |A|$. Il est alors impossible de prédire le résultat puisque l'incertitude est maximale.

Entropie de lois conjointes Notre modèle de canal représente l'émetteur et le destinataire par deux variables aléatoires. Soit (X, Y) la loi conjointe des variables aléatoires X et Y. On cherche à déterminer l'incertitude associée à ces deux variables. Pour cela, on détermine l'entropie de la loi conjointe qui est définie ainsi :

$$H(X,Y) = \mathbb{E}\left[\log_2\left(\frac{1}{P(X,Y)}\right)\right] = -\sum_{x}\sum_{y} p_{x|y} \log_2\left(p_{x|y}\right), \tag{1.4}$$

où $p_{x|y}$ correspond à la probabilité $p(X = x, Y = y)$.

Entropie de lois conditionnelles Dans l'objectif d'obtenir une modélisation d'un canal, on cherche à modéliser les perturbations du canal par des probabilités conditionnelles. Pour cela, on utilise l'entropie conditionnelle $H(X|Y = y)$ qui correspond à l'incertitude de X lorsqu'une valeur de Y est connue :

$$H(X|Y = y) = -\sum_{x} p_{x|y} \log_2 p_{x|y}\ , \tag{1.5}$$

Cette notion peut être ensuite étendue dans le cas où l'on connait l'ensemble des degrés d'originalité des symboles de Y [Sha48, p. 12] :

$$H(X|Y) = -\sum_{x,y} p_{x|y} \log_2\left(\frac{p_y}{p_{x|y}}\right). \tag{1.6}$$

$H(X|Y)$ correspond à la quantité d'information perdue sur le canal.

Information mutuelle

Lors d'une transmission d'information sur un canal, le récepteur doit être capable de déterminer le symbole x_i transmis depuis la source, à partir des informations reçues y_i par le récepteur. L'information mutuelle $I(X,Y)$ mesure la quantité d'information reçue. Elle correspond à la quantité d'information restante lorsque l'on soustrait l'information perdue sur le canal $H(X|Y)$, à l'information émise par l'émetteur $H(X)$ [Sha48, p. 12] :

$$\begin{aligned} I(X,Y) &= H(X) - H(X|Y) \\ &= \sum_x \sum_y p_{x|y} \log_2 \left(\frac{p_{x|y}}{p_x p_y} \right) . \end{aligned} \tag{1.7}$$

En conséquence, deux cas particuliers découlent de cette notion. Si $H(X) = H(X|Y)$, les variables sont indépendantes, ce qui signifie qu'aucune information n'est obtenue. Dans le deuxième cas, $H(X|Y) = 0$, ce qui signifie que l'information mutuelle est maximum. Dans cette situation, Y est entièrement déterminée par X et le canal ne provoque pas d'erreur.

Capacité d'un canal

La capacité d'un canal $C(X,Y)$ est définie ainsi [Sha48, p. 22] :

$$C(X,Y) = \max I(X,Y) . \tag{1.8}$$

Elle correspond à la quantité maximale d'information qui peut être transmise par le canal.

1.1.2 Exemples de canaux

Notre étude va s'intéresser aux deux canaux suivants : le canal binaire symétrique, et le canal à effacement.

Canal binaire symétrique

Le canal binaire symétrique (CBS) est un canal dont la source est définie par un alphabet $A = \{0,1\}$. En conséquence, elle émet des bits à travers un canal caractérisé par une probabilité p d'inverser la valeur du bit. La figure 1.2 de la prochaine page, illustre ce canal. La loi de transition de ce canal est définie ainsi :

$$P(X|Y) = \begin{cases} 1-p & \text{si X=Y} \\ p & \text{sinon} \end{cases} . \tag{1.9}$$

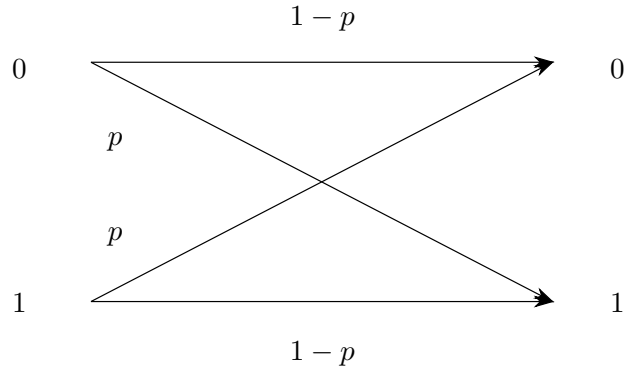


FIGURE 1.2 – Représentation d'un canal binaire symétrique. La source transmet sur le canal des bits dont la valeur peut être altérée avec une probabilité p.

L'entropie est à son maximal lorsque la distribution de Y est uniforme. Dans le cas du canal symétrique, cela correspond à une distribution de X uniforme. En conséquence, l'information mutuelle est maximale lorsque $p = 0.5$. La capacité du CBS correspond à [Dum+07, p. 316] :

$$\begin{aligned} C &= 1 - H_2(p) \\ &= 1 - +p\log_2(p) + (1-p)\log_2(1-p) \end{aligned}, \tag{1.10}$$

où $H_2(p)$ correspond à l'entropie d'une variable aléatoire d'une loi de BERNOULLI (ou loi binaire), de paramètre p. La capacité du canal vaut donc 1 quand la probabilité d'erreur p a valeur dans $\{0, 1\}$. En particulier, lorsque $p = 0$, il n'y a jamais d'erreur de transmission, et Y correspond à X. En revanche, lorsque $p = 1$, la valeur du symbole reçu correspond toujours à l'inverse du symbole émis. Un code dont le rendement vaut 1 (i.e. sans redondance) suffit alors dans ce cas. En revanche, la capacité est nulle quand $p = 0,5$. Ce dernier cas est le plus défavorable puisque X et Y sont indépendants (la sortie n'apporte aucune connaissance sur l'entrée). Du point de vue du rendement, la quantité de redondance nécessaire doit alors tendre vers l'infini.

Canal à effacement

Le canal binaire à effacement (CBE) a été introduit par ELIAS [Eli55]. Ce canal se distingue du CBS par le fait que l'information n'est pas modifiée, mais effacée lors de la transmission. Un effacement correspond simplement à la perte de l'information. Sur la page suivante, la figure 1.3 illustre ce canal, et l'on y représente la perte d'information par le symbole « ? ». La probabilité qu'un symbole soit effacé vaut p, ce qui signifie qu'un symbole est correctement transmis avec une probabilité $1 - p$. En conséquence, on peut représenter la loi de transition $P(X|Y)$ ainsi :

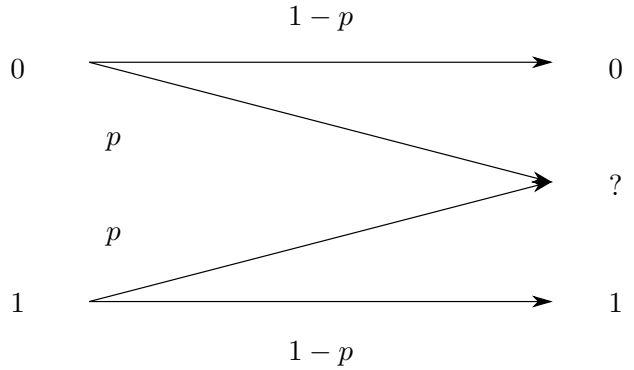


FIGURE 1.3 – Représentation d'un canal binaire à effacement. La source transmet sur le canal des bits qui peuvent être effacés avec une probabilité p. L'effacement représente la transition d'un bit vers « ? ».

$$P(X|Y) = \begin{pmatrix} 1-p & p & 0 \\ 0 & p & 1-p \end{pmatrix} . \tag{1.11}$$

Si l'on reçoit un 0 ou un 1, alors il n'y a aucun doute sur la valeur émise. En conséquence, $H(X|Y=0) = H(X|Y=1) = 0$. En revanche, si on reçoit un « ? », rien n'a été appris sur l'entrée, et donc $H(X|Y=?) = H(X)$. La capacité du canal à effacement vaut $C = 1-p$. On peut ainsi comparer la capacité de ce canal par rapport au précédent. En particulier, puisque $\forall p \in [0, \frac{1}{2}], p < H(p)$, la capacité du canal à effacement est supérieure à celle du CBS. Il est donc plus facile de gérer les effacements que les modifications de valeurs. Ce résultat était prévisible puisque dans le cas du CBS, la correction d'erreurs nécessite au préalable de détecter l'emplacement de ces erreurs avant de pouvoir rectifier leurs valeurs. Dans le cas du canal à effacement en revanche, la position de l'erreur est connue, et la correction consiste uniquement à restituer la valeur de l'information perdue.

Bien que le CBE permet de comprendre le principe des effacements, les applications transmettent des informations sous forme de blocs (ou de paquets). Ces blocs correspondent à des mots de plusieurs bits. Le modèle utilisé est appelé « canal à effacement de paquets » est correspond au CBE dans le cas où les symboles transmis sont des mots de bits (et non plus des bits). En conséquence, un effacement entraîne la perte d'un mot entier.

1.1.3 Théorie algébrique des codes correcteurs

Cette section présente une étude des codes linéaires par blocs à travers une approche algébrique. Une définition des opérations d'encodage et de décodage sera présentée, ainsi que la notion fondamentale de la distance de HAMMING [Ham50]. Cette notion fondamentale de la théorie des codes permet de déterminer la capacité de correction d'un code.

Codes par blocs

Jusque là, nous avons étudié le transfert d'information bit à bit. En pratique, on souhaite transférer un flux d'information de taille conséquente. Pour travailler efficacement sur ce flux d'information, il est préférable de le segmenter en éléments de taille fixe. Ces éléments sont appelés « mots » (ou encore blocs ou paquets). Les mots de code sont le résultat d'une opération d'encodage utilisant des mots d'information en entrée. Dans la suite, nous définirons ce qu'est l'encodage, le décodage, le rendement d'un code, ainsi que la distance de HAMMING.

Encodage Pour un code correcteur (n, k), l'encodage correspond à une application injective $\phi : A^k \to A^n$, où A est un alphabet (dans le cas du canal binaire, $A = \{0, 1\}$) et où $k \leq n$. En particulier, on appelle k la dimension du code, et n sa longueur. L'ensemble $\mathcal{C}_\phi = \{\phi(s) : s \in A^k\}$ correspond au code (n, k) dont les éléments sont des mots de code. Les corps finis correspondent à des structures discrètes adaptées aux applications de codage. Lors d'une transmission, les valeurs de n et k sont définies en fonction de la capacité du canal. Les mots de code ainsi calculés sont transmis sur le canal en direction du destinataire.

Décodage Le décodage consiste à vérifier si les mots reçus appartiennent à $\mathcal{C}_\phi$. Si ce n'est pas le cas, deux stratégies sont possibles pour corriger l'erreur :

1. Le récepteur peut demander à la source de réémettre le message. Cette stratégie appelée *Automatic Repeat reQuest* (ARQ), n'est pas toujours possible (certains médias ne disposent pas de canal de retour). De plus elle entraîne un délai significatif ;

2. L'émetteur ajoute a priori de l'information de redondance afin que le destinataire puisse reconstituer l'information en cas de perte. C'est cette stratégie, appelée *Forward Erasure Code* (FEC), qui sera principalement étudiée dans cette thèse.

Dans le deuxième cas, s'il existe un entier i et un unique mot de code ayant au moins $n - i$ bits égaux au mot reçu, alors on corrige le mot reçu par ce mot. Sinon, on ne peut pas corriger l'erreur.

Rendement Le « rendement » R d'un code (n, k) correspond à la quantité de symboles sources contenus dans un mot de code. Il est défini ainsi :

$$R = \frac{k}{n}. \tag{1.12}$$

Plus le rendement est grand, plus les mots de code contiennent d'informations utiles. En revanche, la capacité de correction diminue. Une fonction de codage ϕ optimale doit générer une quantité de redondance minimale pour une capacité de correction désirée.

Distance de Hamming Soit x et y deux mots de même longueur, construits sur un alphabet A. La distance de HAMMING $d_H(x, y)$ entre x et y correspond au nombre de symboles qui diffèrent entre les deux mots. Par exemple $d(118, 218) = 1$. La distance minimale $d_{\min}(\mathcal{C})$ d'un code $\mathcal{C}$ est définie par :

$$d_{\min}(\mathcal{C}) = \min_{x,y \in \mathcal{C}} d_H(x, y). \tag{1.13}$$

La distance minimale exprime la capacité de correction des codes correcteurs. Supposons que les mots de code $\mathcal{C}$ soient choisis de façon à ce que $d_{\min}(\mathcal{C}) = 2d + 1$. Dans ce cas, le code permet de détecter $2d$ erreurs et d'en corriger d. Nous allons à présent nous intéresser à la conception de l'application ϕ. Dans la prochaine section, nous nous intéresserons aux cas des codes linéaires.

Codes linéaires

Dans le cas où l'application ϕ est linéaire, on dit que le code (n, k) est linéaire. L'avantage des codes linéaires est que les opérations de codage et de décodage sont réalisées en temps polynomial sur n. Il faut alors munir A d'une structure vectorielle. En particulier les corps finis $\mathbb{F}$ correspondent à des ensembles adaptés pour l'étude des codes. Dans la suite de cette section, nous verrons l'application d'encodage, les propriétés MDS et systématique du code, ainsi que la matrice de parité et le décodage.

Application d'encodage linéaire Un code (n, k) est linéaire s'il existe une application linéaire $\phi : \mathbb{F}^k \to \mathbb{F}^n \mid \phi(\mathbb{F}^k) = \mathcal{C}$. Une telle application peut alors être décrite par une matrice G de taille $k \times n$, à coefficient dans $\mathbb{F}$, appelée « matrice génératrice ». L'encodage correspond alors à la multiplication matricielle suivante :

$$Y = XG, \tag{1.14}$$

où $X \in \mathbb{F}^k$ et $Y \in \mathbb{F}^n$. La forme et le contenu de cette matrice sont déterminants pour définir un bon code correcteur. Au cours de ce manuscrit, plusieurs matrices d'encodage seront présentées. Nous verrons que la structure des matrices d'encodage joue un rôle essentiel sur la complexité des opérations d'encodage et de décodage (e.g. matrice creuse, matrice de VANDERMONDE).

Nous allons voir comment obtenir une matrice d'encodage permettant de définir un code systématique, afin de réduire la complexité des opérations d'encodage et de décodage.

Forme systématique des codes Considérons la matrice génératrice G d'un code linéaire, de taille $n \times k$. Il est possible de transformer cette matrice G sous une forme particulière qui contient une matrice identité I_k. Par exemple, on peut utiliser la méthode d'élimination de GAUSS-JORDAN pour obtenir une telle matrice :

$$G = [I_k | T] \ . \tag{1.15}$$

Tout code linéaire peut être écrit sous cette forme, appelée forme systématique. Par opposition, lorsque G ne contient pas de matrice I_k, les codes engendrés sont non-systématiques. Contrairement à ceux-ci, les codes de forme systématique ont l'avantage d'intégrer le message source en clair, dans le mot encodé. Effectivement, les k premiers symboles du mot de code correspondent au message, tandis que les $r = n - k$ derniers correspondent à des symboles de parité. Ces symboles de parité sont engendrés par la matrice T.

En pratique, cette forme a deux avantages. Elle permet tout d'abord de réduire l'opération d'encodage (puisque $\frac{k}{n}$ pourcent du mot de code correspond au message). De plus, elle permet au récepteur d'accéder directement à la donnée lorsque les k premiers symboles n'ont pas été altérés durant la transmission. Dans ce cas, aucune opération de décodage n'est nécessaire.

Nous allons à présent évaluer le rapport entre la capacité de correction et le rendement, en définissant la borne de Singleton [Sin64].

Borne de Singleton et codes MDS Il existe une relation entre la distance minimale $d_{\min}(\mathcal{C})$ et les paramètres (n, k) du code. Cette relation, appelée « borne de Singleton », est définie ainsi [Sin64] :

$$d_{\min}(\mathcal{C}) \leq n - k + 1 \,. \tag{1.16}$$

En conséquence, un code linéaire de distance minimale $d_{\min}(\mathcal{C})$ doit nécessairement générer au moins $n - k + 1$ symboles supplémentaires. Lorsqu'un code atteint la borne de Singleton, il fournit précisément cette quantité de redondance. Dans ce cas, la quantité de redondance est minimale pour une valeur de rendement fixé. Les codes qui atteignent la borne de Singleton sont appelés « codes MDS », pour *Maximum Distance Separable*.

MacWilliams et Sloane [MS06, p. 317] relient le terme « *separable* » à la séparation des symboles (message et parité) dans le cas des codes systématiques. Nous préférons considérer que les codes MDS peuvent être systématiques ou non-systématiques. Nous resterons alors sur la définition des codes MDS comme étant des codes tels que $d_{\min}(\mathcal{C}) = n - k + 1$.

On peut à présent définir une nouvelle matrice H d'une application dont les noyaux correspondent aux mots de code.

Matrice de contrôle de parité Il est également possible de définir un code linéaire en utilisant une application linéaire dont le noyau correspond au code. Cette application est représentée par une matrice H, appelée « matrice de contrôle de parité » :

$$\mathcal{C} = \left\{ (y_1, \ldots, y_n) : H \times \begin{pmatrix} x_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix} = 0 \right\} . \tag{1.17}$$

Une telle matrice est facilement déterminée à partir de la matrice $G = [I|T]$ définie précédemment. En particulier, $G \times H^t = 0$, on en conclut que :

$$H = \left[-T^t \middle| I_r\right] \ , \tag{1.18}$$

où T^t est la transposée de la matrice T, et I_r est une matrice identité de taille $r \times r$, où $r = n - k$. Nous verrons dans la prochaine définition que cette matrice permet de déterminer le syndrome d'un mot de code, et de détecter les erreurs.

Syndrome et décodage Lorsqu'un message y est reçu, pour déterminer si celui-ci correspond à un mot de code (et donc s'il n'a pas été altéré, a priori) on calcule la valeur Hy, qui correspond au « syndrome » du mot de code. Si le syndrome n'est pas nul, la présence d'erreurs est validée. On cherche alors à déterminer la valeur du mot de code x à partir de y. Pour cela, on calcule le vecteur d'erreurs $e = y - x$. Le décodage consiste donc à trouver l'unique élément $x \in \mathcal{C} \mid d_H(x, y) \leq t$, où t correspond au nombre d'erreurs que peut corriger le code.

1.2 Codage à effacement

Nous avons vu précédemment le cas du canal à effacement. Dans cette section, nous allons détailler le principe des codes correcteurs qui s'appliquent sur ce canal. En pratique, nous verrons dans la section 1.2.1 que les codes s'appliquent à des paquets de données plutôt que sur des bits. Le principe des codes à effacement sera ensuite étudié dans la section 1.2.2. Enfin, nous verrons dans la section 1.2.3 ce qui distingue les différents codes à effacement.

1.2.1 Codage par paquets sur la couche applicative

En pratique, les symboles peuvent représenter différents contenus d'information. Sur la couche physique, les symboles correspondent à des bits (comme nous l'avons vu jusqu'à présent). Dans le cas du standard de transmission vidéo DVB-H [Far+06], un symbole correspond à un octet. Pour un protocole réseau, il peut s'agir de paquets réseau [Tou+11 ; Lac+09]. Dans un contexte de stockage, un symbole peut également représenter des parties de fichiers, voire des fichiers entiers [Hua+12].

Jusque là, nous avons considéré le cas du canal binaire à effacement. Dans ce contexte, les codes travaillent sur un mot constitué de n symboles. En informatique, les transmissions d'information réalisées au niveau applicatif travaillent sur de grandes tailles d'informations découpées en paquets (ou blocs). On désigne les codes à effacement appliqués à la couche applicative par le terme « codes AL-FEC » (pour *Application-Level Forward Erasure Codes*) [Cun10]. Dans la suite, nous considérerons alors le canal à effacement par paquets comme étant le canal sur lequel les codes génèrent n blocs de redondance à partir de l'information contenue dans k paquets.

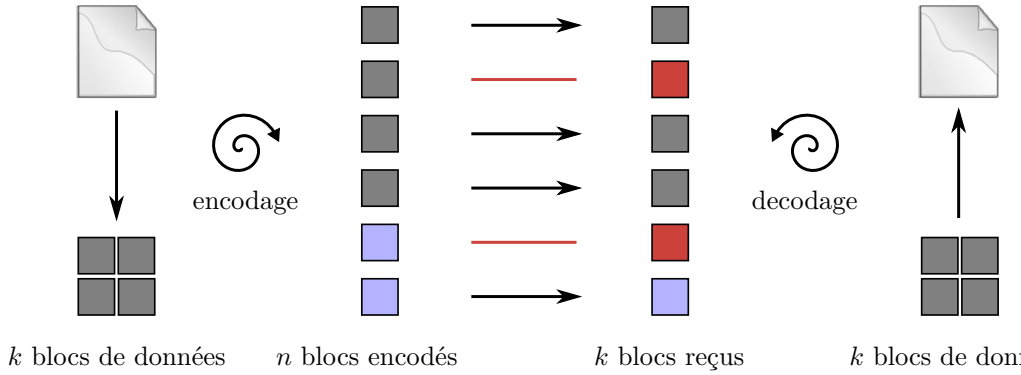


FIGURE 1.4 – Principe d'un code à effacement (n, k). Un flux d'information de $\mathcal{M}$ octets est découpé en k blocs de données (en gris). L'encodage systématique consiste à générer $r = (n - k)$ paquets de parité (en bleu clair). L'ensemble de ces n paquets encodés est envoyé sur le canal. Le décodage consiste à reconstruire les k blocs de données. Si le code est MDS, ce décodage tolère la perte de n'importe quel ensemble de r paquets (en rouge).

1.2.2 Principe des codes à effacement

Un code à effacement (n, k) permet de générer n paquets encodés à partir d'un ensemble de k paquets de données, tel que $k \leq n$. La figure 1.4 illustre le principe d'un code à effacement $(6, 4)$ MDS. Dans l'exemple de la figure, le code est présenté sous sa forme systématique. Les $n = 6$ paquets encodés contiennent alors les $k = 4$ paquets d'information (blocs systématiques), auxquels s'ajoutent r paquets de parité (ou de redondance). Lors de la réception des paquets, il est nécessaire de vérifier si certains blocs systématiques ont été perdus. Le cas échéant, l'opération de décodage consiste à reconstruire cette information à partir des informations de parité. Si le code est MDS, l'information contenue dans les r blocs de parité permet de reconstruire n'importe quel sous-ensemble de r blocs de données perdus.

1.2.3 Distinction entre les différents codes

De nombreux codes à effacement existent [RS60 ; Gal62 ; Lub02 ; Sho06]. Ce qui les différencie repose dans l'organisation des informations de redondance, ainsi que dans les relations de linéarité qui permettent de les calculer. En particulier, cette distinction de conception entraîne des différences sur la complexité d'encodage et de décodage. Nous proposons ainsi les six critères de distinction suivants :

1. la **complexité théorique** dépend du nombre d'opérations élémentaires nécessaires (il s'agit généralement d'additions et de multiplications dans un corps fini). Cette information donne un aperçu de l'évolution de la complexité de l'algorithme en fonction de la taille en entrée. Cependant, elle ne prend pas en compte le coût des opérations de base. En conséquence elle ne peut permettre à elle seule de distinguer différents codes ;

2. la **complexité des opérations de base** constitue alors un deuxième critère. En particulier, une multiplication nécessite plus d'opérations qu'une addition. Le coût d'une opération dépend cependant de la manière dont elle est mise en œuvre ;

3. l' **indépendance des paramètres** de codes forme le troisième critère. Celui-ci correspond à la possibilité de choisir arbitrairement les valeurs (n, k) du code. Nous verrons dans la prochaine section l'exemple du code de parité, dont la longueur et la dimension sont liées par la relation $n = k + 1$;

4. le **rendement** correspond au quatrième critère. Nous cherchons à concevoir des codes MDS qui minimisent la quantité de redondance nécessaire ;

5. la **complexité à déterminer une forme systématique** varie pour l'ensemble des codes. Les codes de parité sont par définition systématiques, et ne nécessitent pas de méthode de Gauss-Jordan, comme nous avons vu précédemment ;

6. l'**évaluation de l'implémentation** du code correspond à notre dernier critère. Il consiste à mesurer les débits d'encodage et de décodage du code. Toutefois, ce critère dépend significativement de l'implémentation utilisée.

Nous allons détailler dans la suite quelques exemples de codes à effacement.

1.3 Exemples de codes à effacement

Les codes à effacement ont été un sujet de recherche très prolifique en publications scientifiques. Aussi, le nombre de codes qui ont été conçus est très important. Dans cette section, nous allons étudier quatre codes à effacement qui représentent les différentes familles de codes à effacement : les codes de répétition, les codes de parité, les codes de Reed-Solomon et les codes LDPC. Nous verrons en particulier leurs caractéristiques et leurs distinctions. Nous donnerons en conclusion un récapitulatif de cette étude.

1.3.1 Les codes de répétition

Les codes de répétition correspondent à des codes $(n, 1)$, dont la mise en œuvre est simple. Il s'agit de répéter plusieurs fois les symboles à transmettre. Dans le cas du canal binaire à effacement, chaque bit est répété n fois. Prenons l'exemple d'un code de longueur 3, le code génère alors les deux mots de codes suivant : $\mathcal{C} = \left\{ \begin{pmatrix} 0 & 0 & 0 \end{pmatrix}, \begin{pmatrix} 1 & 1 & 1 \end{pmatrix} \right\}$. La matrice d'encodage de ce code correspond à la matrice suivante :

$$G = \underbrace{\begin{pmatrix} 1 & 1 & \cdots & 1 \end{pmatrix}}_{n} . \tag{1.19}$$

Dans le cas du canal binaire symétrique, le décodage consiste à déterminer quelle valeur est la plus répétée dans le mot reçu. Pour le canal à effacement, dès lors qu'un symbole $(k = 1)$ parmi les n est reçu, il correspond au symbole transmis par l'émetteur.

Puisque les mots de code n'ont aucun bit en commun, cela signifie que la distance minimale vaut $d_{\min}(\mathcal{C}) = n$. En conséquence, ce code est MDS. Toutefois, le problème de cette technique réside dans le coût d'une transmission. La transmission d'un symbole d'information nécessite l'envoie de n symboles. Le rendement de ce code vaut $R = \frac{1}{n}$, ce qui pénalise significativement le critère 4 énoncé précédemment. Dans la suite, nous verrons des codes proposant de meilleurs rendements.

1.3.2 Les codes de parité

Le code de parité est un autre code simple à mettre en œuvre. Il permet un meilleur rendement que les codes de répétition, mais sa capacité à corriger est limitée à une erreur : il s'agit d'un code $(n, k = n - 1)$. La matrice d'encodage d'un code de parité de longueur n correspond à :

$$G = \left(\begin{array}{cccc|c} 1 & 1 & \cdots & 0 & 1 \\ 0 & 1 & \cdots & 0 & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & 1 \end{array}\right) = \left(\underbrace{\begin{array}{cccc} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{array}}_{I_{n-1}} \middle| \begin{array}{c} 1 \\ 1 \\ \vdots \\ 1 \end{array}\right). \tag{1.20}$$

Le code de parité étant systématique, sa matrice génératrice G est composée d'une matrice identité I_{n-1}. La dernière colonne correspond à la colonne de parité. Cette colonne permet de construire un symbole dont la valeur est la somme de la valeur des autres symboles. En conséquence, s'il manque un élément du mot à la réception, sa valeur peut être calculée comme la somme des valeurs des autres symboles. Quelle que soit la longueur de ce code, sa distance minimale vaut 2. Ainsi, bien qu'il soit MDS, ce code n'est capable de corriger qu'un seul effacement. Ce code n'est donc pas bon au regard du critère 3. En revanche, son rendement vaut $R = \frac{n-1}{n}$ ce qui signifie qu'une quantité plus importante de données utiles est contenue dans un mot de code par rapport au code de répétition.

1.3.3 Les codes de Reed-Solomon

Les codes de Reed et Solomon [RS60] sont les codes à effacement les plus populaires. Cette popularité provient du fait qu'ils correspondent à des codes MDS, dont les paramètres (n, k) peuvent être choisis arbitrairement. Pour cela, la matrice génératrice de taille $n \times k$ doit avoir la propriété suivante : n'importe quelle sous-matrice de taille $k \times k$ de G est inversible. Les matrices de Vandermonde $V_{i,j} = \alpha_i^{j-1}$, où les α_i correspondent à des éléments du corps fini, possèdent une telle propriété. L'encodage correspond alors à multiplier une telle matrice avec le vecteur colonne représentant le message à transmettre [Lac+09].

Soit x' le message à transmettre, x le mot de code transmis, et y le message reçu. Dès que le destinataire reçoit k symboles parmi les n calculés, il est capable de décoder le

message. Pour cela, on construit une matrice G' constituée des k lignes de G correspondant aux symboles reçus. Puisque G est une matrice de VANDERMONDE, G' est nécessairement inversible. En conséquence, le message x' est reconstitué par l'opération suivante : $x' = G'^{-1}y$.

Bien que les codes de REED-SOLOMON soient MDS, leur défaut provient de leur complexité calculatoire lors des opérations d'encodage et de décodage. Le décodage nécessitant une multiplication matricielle, implique $\mathcal{O}(k^3)$ opérations arithmétiques dans un corps de GALOIS. Notons que la mise en œuvre des multiplications dans un corps de GALOIS a un coût significativement élevé par rapport aux additions (ce qui pénalise le critère 1 et 2). Plusieurs méthodes ont été proposées pour réduire cette complexité. BLÖMER et al. [Blö+95] utilisent des matrices d'encodage basées sur des matrices de CAUCHY. En particulier, ils représentent la matrice d'encodage de façon à réaliser les opérations sans multiplication, ce qui permet de réduire la complexité à $\mathcal{O}(k^2)$. SORO et LACAN [SL10] ont par la suite réduit cette complexité à $\mathcal{O}(k \log k)$ en utilisant la transformée de FOURIER.

1.3.4 Les codes LDPC

Nous avons vu que malgré l'optimalité des codes de REED-SOLOMON en matière de rendement (codes MDS), ils induisent une complexité significative lors des opérations d'encodage et de décodage. Les codes LDPC sont à l'inverse des codes aux complexités linéaires, mais non MDS. Pour répondre au problème du canal binaire symétrique, GALLAGER [Gal62] a proposé des codes basés sur une matrice de parité à faible densité (LDPC). Ces codes ont ensuite été proposés dans le cas du canal à effacement par LUBY et al. [Lub+97]. Cette famille de code utilise l'algorithme de propagation de croyance (*Belief Propagation*). Il s'agit d'un algorithme itératif qui permet au décodage d'atteindre une complexité linéaire. Les codes LDPC peuvent être représentés soit par une matrice de contrôle de parité, soit par un graphe de TANNER.

La matrice de parité Pour faciliter l'opération de décodage, la matrice de parité H correspond à une matrice binaire creuse. Cela signifie que la matrice H est essentiellement constituée de 0 par rapport à la quantité de 1. Prenons l'exemple suivant représentant une matrice de parité de taille 8×4 définissant un code LDPC :

$$H = \begin{pmatrix} 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \end{pmatrix}. \tag{1.21}$$

Pour être creuse, une matrice doit garantir que le nombre de 1 dans chaque colonne $w_c \ll n$ et le nombre de 1 par ligne $w_r \ll k$. En pratique, la taille des matrices des codes LDPC est suffisamment importante pour pouvoir garantir cela (ce n'est pas le cas de l'exemple décrit par l'équation (1.21)). La matrice de l'équation (1.21) permet de représenter 4 équations mettant en jeu les bits d'un octet. L'avantage de cette représentation est de

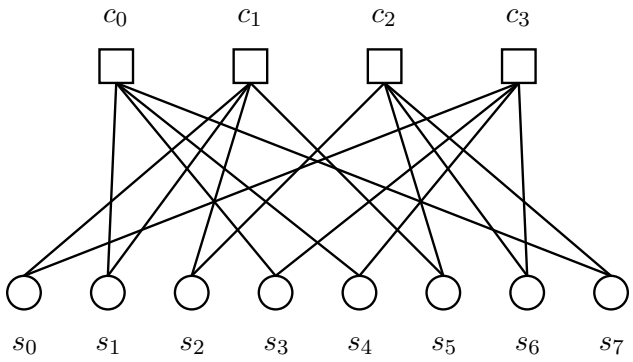


FIGURE 1.5 – Représentation d'un graphe de TANNER correspondant au code LDPC de l'équation (1.21). Chaque sommet rond correspond à un symboles s_i. Les sommets c_i définissent des contraintes de parité entre les symboles, représentées par les liens.

pouvoir analyser la structure de la matrice. Il est possible d'accélérer le décodage du code lorsque qu'une partie de H possède une structure particulière (triangulaire, par blocs, etc.).

Le graphe de Tanner La figure 1.5 représente le graphe de TANNER correspondant à l'équation (1.21). Un tel graphe contient deux ensembles de sommets qui correspondent (i) aux bits à transférer s_i ; (ii) aux sommets de contrainte c_i. Chaque arête fait le lien entre les deux ensembles de sommets. Ces deux représentations sont équivalentes.

Décodage

Il existe deux techniques usuelles de décodage : le décodage itératif et le décodage au maximum de vraisemblance.

Décodage itératif Le décodage itératif est particulièrement performant, en revanche il n'est pas optimal en capacité de correction. Il consiste à considérer les équations décrites par l'équation (1.21) dont l'une des variables est inconnue. La valeur de cette variable correspond donc à la constante de l'équation. Par la suite, la valeur de cette variable est mise à jour dans l'ensemble des équations du système. Dès lors qu'aucune équation ne contient plus qu'une seule inconnue, ou bien dès lors que l'ensemble des symboles est reconstruit, l'algorithme s'arrête.

Décodage au maximum de vraisemblance Contrairement à la méthode itérative, cette méthode de décodage est optimale en capacité de correction. En revanche, elle a une complexité plus importante. Dans le cas du canal binaire symétrique, cette technique consiste à déterminer le mot de code le plus proche du mot reçu (le plus vraisemblable). Dans le cas du canal à effacement, cette méthode revient à résoudre un système linéaire avec une technique semblable à une élimination de GAUSS-JORDAN. CUNCHE et ROCA

	Répétition	Parité	Reed-Solomon	LDPC
C1. complexité théorique	–	✓	–	✓
C2. complexité opérationnelle	✓	✓	–	✓
C3. indépendance des paramètres	✓	–	✓	✓
C4. rendement	–	✓	✓	–
C5. complexité systématique	✓	✓	–	✓

Table 1.1 – Résumé de l'analyse théoriques des différents codes en fonction des critères énumérés précédemment (voir section 1.2.3). Le tableau distingue les critères satisfaits par une coche. Le critère 6 n'est pas évalué puisqu'il concerne les implémentations de ces codes.

[CR08] ont conçu une méthode hybride permettant de commencer le décodage par une méthode itérative, puis de passer par une technique au maximum de vraisemblance si l'algorithme s'arrête. En particulier, plus l'algorithme itératif reconstruit de symboles, plus la complexité de l'élimination de Gauss-Jordan se réduit.

Conclusion de la section

Le tableau 1.1 résume l'analyse théorique en fonction des cinq premiers critères établis dans section 1.2.3. Chaque code proposé possède ses défauts et ses avantages, donc aucun n'est parfait. Par exemple, la complexité linéaire $\mathcal{O}(n)$ du décodage itératif des codes LDPC, qui n'utilise que des opérations d'addition, offre de bonnes performances. En revanche, ces codes ne sont pas MDS. À l'inverse, les codes de Reed-Solomon ont un rendement optimal (codes MDS), mais leur complexité au décodage est moins bon que les codes LDPC. De plus, ils nécessitent des opérations de multiplication dans les corps de Galois. Ces opérations sont plus compliquées à mettre en œuvre que de simples additions. Il existe cependant de nombreuses publications qui ont permis de réduire la complexité des codes de Reed-Solomon. Ainsi, bien que leur complexité soit souvent considérée quadratique, les travaux de Soro et Lacan [SL10] ont permis de réduire cette complexité à $\mathcal{O}(n \log n)$.

Conclusion du chapitre

Ce chapitre nous a permis de présenter un état de l'art des codes correcteurs pour le canal à effacement. La section 1.1 a introduit les notions nécessaires pour comprendre la théorie des codes, dont notamment celle de la capacité d'un canal. Par la suite, nous avons étudié le cas du canal à effacement dans lequel certains symboles ont une probabilité d'être effacé. En comparant leur capacité respective, nous avons vu que la correction de l'effacement est plus simple à résoudre que la correction d'erreur.

Afin de résoudre le problème du canal à effacement par paquets, il est nécessaire de proposer des codes à effacement capables de générer des paquets de redondance au

niveau du récepteur. En particulier, les codes AL-FEC sont utilisés au niveau de la couche logicielle afin que le destinataire puisse reconstituer l'information même lorsqu'une partie des paquets a été effacée. La section 1.2 présente le principe de ces codes à effacement linéaires ainsi que six critères permettant de les caractériser.

Enfin, quelques exemples représentatifs des codes à effacement linéaires en blocs ont été vus dans la section 1.3. Plus particulièrement, bien qu'ils soient simples à mettre en œuvre, les codes de répétition sont coûteux du fait de leur mauvais rendement. Les codes de parité, en revanche, offrent un meilleur rendement, mais ne peuvent toutefois corriger qu'un seul effacement. Ayant une capacité de correction importante et étant MDS, les codes de Reed-Solomon sont par conséquent les plus populaires. Toutefois, ces codes impliquent une complexité quadratique au décodage qui les pénalise. Quant aux codes LDPC, ils possèdent un algorithme de décodage itératif linéaire efficace, mais n'ont pas un rendement optimal (MDS dans le cas asymptotique).

Pour résumé, aucun des codes proposés dans ce chapitre n'est parfait. Un code parfait correspondrait à un code MDS, de faible complexité en encodage et décodage, et dont la capacité de protection peut être fixée arbitrairement. Nous verrons dans les prochains chapitres que le code à effacement Mojette s'en rapproche.

2

Conception de codes à effacement en géométrie discrète

Sommaire

Introduction du chapitre

Dans le chapitre précédent, nous avons vu que les codes à effacement permettent de générer la redondance nécessaire pour contrebalancer la perte d'une partie de l'information. En particulier, les codes de Reed et Solomon [RS60] sont les plus utilisés. Bien qu'optimal au sens MDS, ces codes utilisent des algorithmes de complexité quadratique. Nous avons vu précédemment qu'un code parfait doit être MDS, de paramètres (n, k) arbitraires, et disposant d'algorithmes d'encodage et de décodage de faible complexité. Dans ce chapitre, nous proposons d'attaquer le problème de reconstruction à partir d'une approche par géométrie discrète, en détaillant les travaux qui ont été réalisés jusqu'à aujourd'hui, dans le but de concevoir de nouveaux codes à effacement.

La transformation de Radon [Rad17] est une application utilisée en reconstruction tomographique. Elle permet de représenter une fonction par ses projections suivant différents angles. Évoqué précédemment, le problème de reconstruction (ou d'inversion) correspond à savoir s'il est possible de reconstruire cette fonction à partir des seules informations de projections. Nous verrons dans la suite qu'il est possible de définir des versions de cette transformation capables de représenter l'information de manière redondante. Notre objectif est ainsi de déterminer des algorithmes capables de reconstruire l'information à partir d'un ensemble suffisant de projections (comme un code à effacement). L'inversion de la transformation de Radon possède des liens avec la transformation de Fourier. Ces liens sont décrits par le théorème de la tranche centrale [Bra56]. Ce théorème peut être utilisé pour concevoir un algorithme de reconstruction basé sur la transformation de Fourier rapide FFT et bénéficier d'une complexité quasi-linéaire $\mathcal{O}(n \log n)$ [CLW69].

La première étape consiste à discrétiser la transformation de Radon, initialement définie dans le domaine continu [Rad17]. Cette étude sera réalisée en section 2.1. Matúš et Flusser sont parvenus à définir une version discrète de la transformation de Radon, qui conserve toutes les propriétés de la version continue [MF93] (en particulier, le théorème de la tranche centrale). Plus particulièrement, nous verrons deux versions discrètes et exactes de la transformation de Radon : (i) la transformation de Radon finie (pour *Finite Radon Transform* ou FRT) ; (ii) la transformation Mojette. Le principe de cette première transformation sera étudiée dans la section 2.2, avant de nous intéresser à un algorithme d'inversion, ainsi que sa mise en œuvre comme code à effacement. Nous mettrons en avant la propriété périodique de la FRT, et nous verrons qu'elle permet d'en fournir un code MDS [Nor+10]. La section 2.3 présente la seconde version discrète de la transformation de Radon. Bien que la propriété apériodique des projections Mojette empêche la conception d'un code à effacement MDS, elle possède un algorithme de reconstruction itératif efficace. Nous verrons en particulier l'algorithme de Normand et al. [NKÉ06] qui permet de reconstruire chaque symbole avec une complexité linéaire. Après avoir défini le critère de Katz permettant de garantir l'unicité de la solution de reconstruction, nous verrons comment construire un code à effacement à partir de cette transformation. Dans ce chapitre, nous verrons les propriétés des *fantômes* qui sont des éléments de l'image invisible dans l'espace transformé [Kat78]. Ces fantômes nous

permettront non seulement de comprendre les processus d'inversion dans ce chapitre, mais seront également utilisés au chapitre 6 afin de concevoir une méthode pour générer de nouvelles projections.

2.1 Discrétisation de la transformation de Radon continue

La transformation de Radon [Rad17] est une application linéaire permettant de représenter une fonction par ses projections suivant différents angles. L'inversion de cette opération consiste à reconstruire la fonction à partir des valeurs de projections (solution du problème de reconstruction en tomographie). La tomographie est une catégorie des problèmes inverses qui consiste à reconstruire un objet à partir d'un ensemble de mesures partielles et discrètes (i.e. les projections). En particulier, cette technique permet la visualisation et le stockage d'une version numérique d'un objet.

Dans le milieu médical, il faudra attendre 1972 avant que Hounsfield ne parvienne à concevoir le premier scanner à rayon X, sans pour autant qu'il n'ait eu au préalable connaissance des travaux de Radon. Il remportera le prix Nobel de médecine en 1979 avec Cormack pour leurs travaux respectifs sur le développement de la tomographie numérique [Hou73 ; Cor63]. Cette technique, largement répandue dans le milieu médical, a également été utilisée en astronomie par Bracewell [Bra56], en géophysique ou encore en mécanique des matériaux. Une étude plus approfondie de la tomographie et de ses applications est présentée dans les travaux de thèse de Der Sarkissian [Der15]. L'originalité de nos travaux de recherche consiste à détourner l'utilisation de cette technique utilisée en imagerie, pour concevoir des codes à effacement appliqués à la transmission et au stockage d'information.

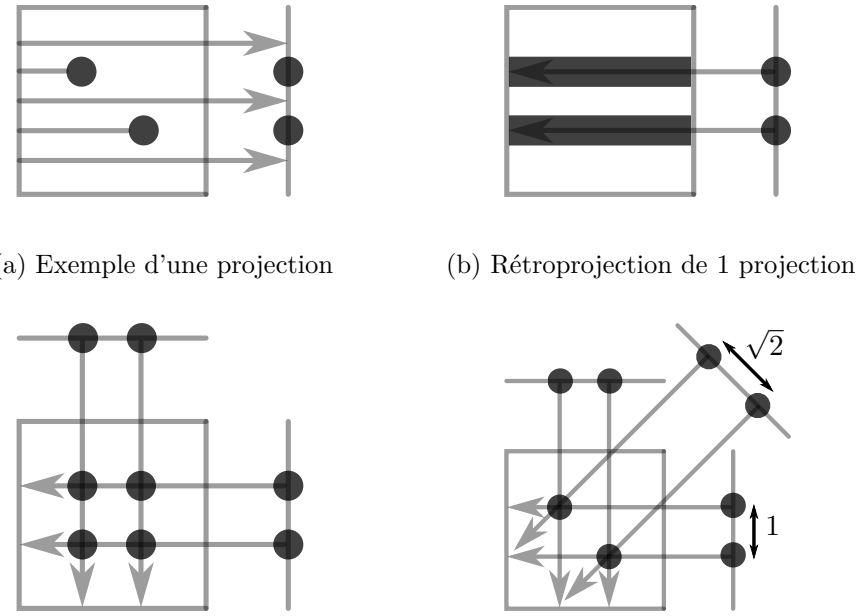
Dans la suite, nous introduirons en section 2.1.1 la transformation de Radon continue telle que définie par Radon [Rad17]. Afin de comprendre comment discrétiser cette transformation, quelques notions de géométrie discrète seront ensuite définies dans la section 2.1.2. Enfin nous verrons une première approche du problème de tomographie discrète avec un exemple en section 2.1.3 présentant une méthode de résolution algébrique.

2.1.1 Transformation de Radon dans le domaine continu

La transformation de Radon est une application qui répond au problème de la tomographie. Nous introduirons dans un premier temps ce problème dans un contexte pratique : l'imagerie médicale. Puis nous verrons la définition de la transformation de Radon continue telle que définie dans les travaux fondamentaux de Radon [Rad17].

Problème de la tomographie

En imagerie médicale, le problème de la tomographie correspond à un problème inverse. Ce problème consiste à reconstituer une image à partir de ses projections. On distingue alors deux processus dans la résolution de ce problème : (i) *l'acquisition* des données ; (ii) la *reconstruction* de l'image. Ces deux processus sont représentés dans la figure 2.1.



(a) Exemple d'une projection

(b) Rétroprojection de 1 projection

(c) Rétroprojection de 2 projections

(d) Rétroprojection de 3 projections

FIGURE 2.1 – Représentation des différentes étapes en tomographie. Figure 2.1a représente une étape d'acquisition des données par la projection des deux formes de l'image sur l'horizontale. Figures 2.1a à 2.1d représentent itérativement la rétroprojection des données.

Celle-ci présente les différentes étapes d'une reconstruction tomographie appliquée à une image composée de deux disques.

L'acquisition met en jeu la rotation d'un capteur qui mesure des projections 1D autour d'une zone du patient. Cette technique est notamment utilisée dans les scanners à rayons X [Hou73]. Ces appareils envoient une série de rayons X à travers le patient à différents angles. Les récepteurs situés de l'autre côté du sujet mesurent l'absorption de ces rayons par les tissus organiques. Il est alors possible de déterminer le volume de tissu traversé par ces rayons. La mesure suivant la direction horizontale, est représentée dans la figure 2.1a. L'émetteur situé à gauche envoie des rayons en parallèle (d'autres cas où les rayons sont émis en éventail ou en cône existent, mais ne sont pas traités ici) et permet au capteur situé à droite de mesurer une empreinte des deux formes étudiées.

Une fois l'acquisition terminée, un traitement informatique permet de reconstruire une coupe du patient. Ce traitement correspond à l'opération inverse. Une technique pour reconstruire l'image consiste à rétroprojeter la valeur des projections dans le support à reconstruire. Si l'on ne dispose pas de suffisamment de projections, alors l'ensemble des solutions possibles peut être significativement grand (voire infini) et il est impossible de déterminer une solution unique. Plusieurs itérations de l'opération de rétroprojection sont représentées dans les figures 2.1a à 2.1c. Plus particulièrement, la figure 2.1b montre que l'ensemble des solutions est infini lorsque l'on n'utilise qu'une projection. Plus on

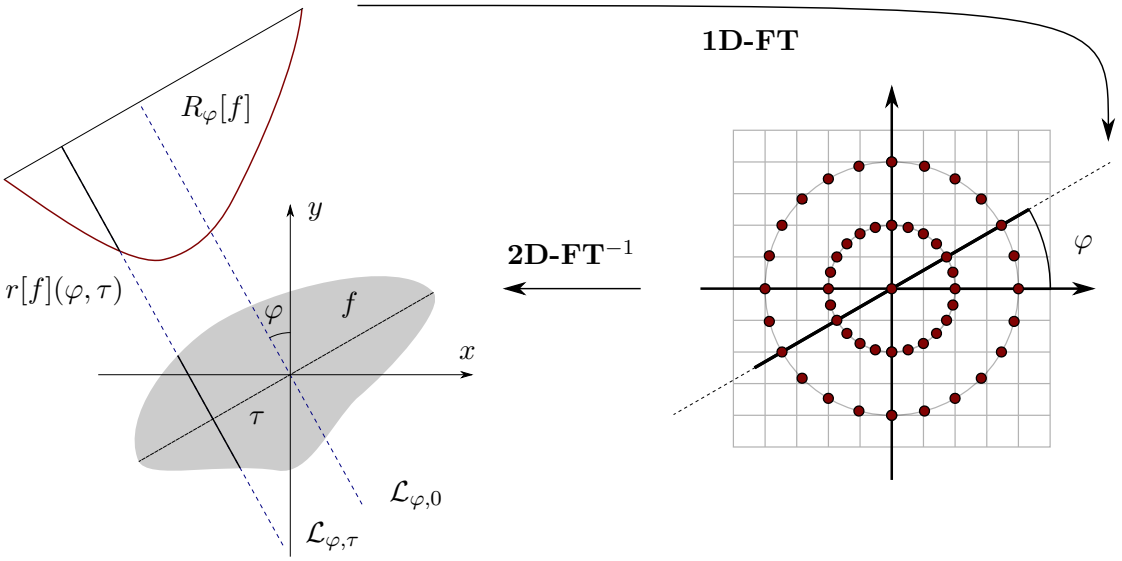


Figure 2.2 – Représentation d'une projection $R_\varphi[f]$ de la fonction f suivant l'angle de projection φ, et illustration de la reconstruction par le théorème de la tranche centrale (TF désigne Transformation de Fourier).

utilise de projections, plus la dimension de l'espace des solutions se réduit. La figure 2.1c montre cela en mettant en jeu une seconde projection. Pour finir, si suffisamment de projections sont utilisées, on parvient à déterminer une unique solution (i.e. l'image initiale). C'est le cas de la figure 2.1d dans laquelle trois projections sont utilisées. Pour définir des codes à effacement, nous verrons des critères sur le nombre de projections nécessaires afin de garantir l'unicité de la solution.

Transformation de Radon

En 1917, Radon a défini la théorie mathématique de sa transformation dans ses travaux fondamentaux [Rad17]. La transformation de Radon $\mathcal{R}\colon f \mapsto R$ est une application permettant de calculer les projections 1D d'une fonction 2D $f\colon \mathbb{R}^2 \to \mathbb{C}$. Soit $\{\mathcal{L}_\varphi\} = \{t = x\cos\varphi + y\sin\varphi \mid t \in \mathbb{R}\}$ l'ensemble des droites de projections définies par un angle φ. Une projection $R\colon f \mapsto r$ est l'ensemble des intégrales curvilignes de f le long des droites de projection $\{\mathcal{L}_\varphi\}$:

$$R_\varphi[f] = \int_{\mathcal{L}_\varphi} f(x,y)\, d\ell \,, \tag{2.1}$$

où $d\ell$ correspond aux variations le long des droites de $\{\mathcal{L}\}$. La figure 2.2 représente la projection $R_\varphi[f]$. En particulier, une mesure de projection $r[f](\varphi,t)$ est définie telle que :

$$r[f](\varphi,t) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(x,y)\delta(t - x\cos\varphi - y\sin\varphi)\, dx\, dy \,, \tag{2.2}$$

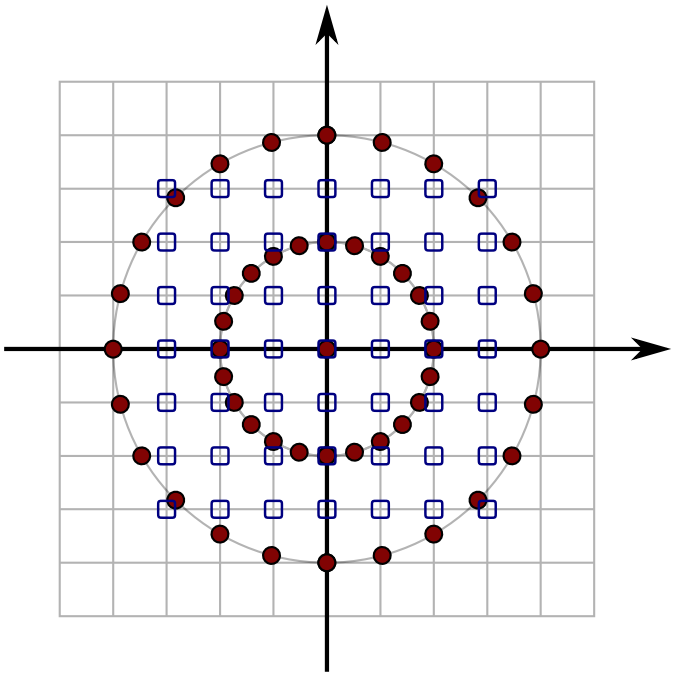


FIGURE 2.3 – L'inversion de la transformée de FOURIER rapide nécessite l'interpolation de la grille polaire (ronds rouges) obtenue par le théorème de la tranche centrale, à la grille cartésienne (carrés bleus).

où φ représente l'angle de projection, et $|t|$, la distance par rapport à l'origine. $\delta(\cdot)$ correspond à la distribution de DIRAC. La mesure de projection $r[f](\varphi, t = \tau)$ est représentée dans la figure 2.2. Notons que la projection $r[f](\varphi, t) = r[f](\varphi + \pi, -t)$. Dans ce formalisme, la transformée de RADON d'une fonction f est l'ensemble des projections tel que :

$$\{r[f](\varphi, t) \mid \varphi \in [0, \pi[, t \in \mathbb{R}\} \,, \tag{2.3}$$

La seconde étape en tomographie consiste à inverser l'opération précédente. Cette opération consiste à déterminer f en tout point de l'espace, c'est à dire :

$$\{f(x, y)|(x, y)\ in\mathbb{R}^2\} \,, \tag{2.4}$$

à partir de l'ensemble des projections. RADON [Rad17] a montré qu'il était possible d'inverser l'opération décrite dans l'équation (2.2). Pour cela, le théorème de la tranche centrale peut être utilisé. Ce théorème crée un lien entre la transformation de FOURIER 1D d'une projection, et la tranche orthogonale à la direction de projection de la transformée de FOURIER 2D de f. La figure 2.2 illustre la transformation d'une projection de RADON en une tranche du domaine de FOURIER. En utilisant l'ensemble des projections, il est possible de construire entièrement le domaine de FOURIER. Une transformation de FOURIER 2D inverse de ce domaine permet de reconstruire l'image f.

Cependant, la principale limitation de cette méthode réside dans la nécessité d'interpoler la grille polaire à la grille cartésienne dans le domaine de Fourier. La figure 2.3 illustre ce problème dans lequel il faut faire correspondre les éléments du repère polaire (ronds rouges) aux éléments du repère cartésien (carrés bleus). Bien que des méthodes efficaces existent pour répondre à ce problème [Ave+06], nous verrons dans la suite de nos travaux, des méthodes qui ne nécessitent pas d'interpolation. Pour résumer, la reconstruction suit les étapes suivantes :

1. calculer les transformées de Fourier 1D de chaque projection afin de remplir la grille polaire ;

2. interpoler la grille polaire sur la grille cartésienne ;

3. calculer la transformation de Fourier 2D inverse pour obtenir une version échantillonnée de f.

Il existe trois raisons pour lesquelles la reconstruction par la transformation de Radon est un *problème mal posé*, au sens défini par Hadamard [Had02] : (i) la solution ne peut être retrouvée puisque les mesures réalisées lors de l'acquisition intègrent du *bruit* dans les données ; (ii) il n'est de plus pas possible de garantir l'*unicité* de la solution puisque l'acquisition mesure un nombre fini de projections ; (iii) les données mesurées correspondent à un échantillon de l'image, caractérisé par la distance des capteurs (dans le cas d'une étude géométrique parallèle). Enfin, une petite erreur d'acquisition entraîne de fortes variations des résultats.

Dans ce chapitre, nous présenterons des versions exactes et discrètes de la transformation de Radon. Ces versions exactes reposent sur un échantillonnage optimal, ce qui permet de ne pas avoir à réaliser d'interpolation lors de la reconstruction. L'échantillonnage est optimal lorsque les projections couvrent uniformément l'ensemble des éléments de l'image. Pour définir ces versions discrètes, nous allons avoir besoin de notions de géométrie discrète, que nous tacherons de définir dans la prochaine section.

2.1.2 Quelques bases de la géométrie discrète

Le procédé permettant de transformer des éléments continus en éléments discrets est appelé *discrétisation* (ou *numérisation*). Il est ainsi possible de transformer une fonction continue $f : \mathbb{R}^2 \to \mathbb{R}$ en une fonction discrète $f : \mathbb{Z}^2 \to \mathbb{Z}$. Nous allons définir ici les notions de géométrie discrète nécessaires pour comprendre la discrétisation de la transformation de Radon. Nous étudierons dans un premier temps les aspects topologiques qui nous permettront de comprendre la représentation discrète de l'image à reconstruire. Par la suite, nous verrons quelques objets relevant de la géométrie discrète, comme les angles et les droites, qui nous permettront de définir les droites de projection. Ces notions sont extraites du livre de Coeurjolly et al. [CMC07a].

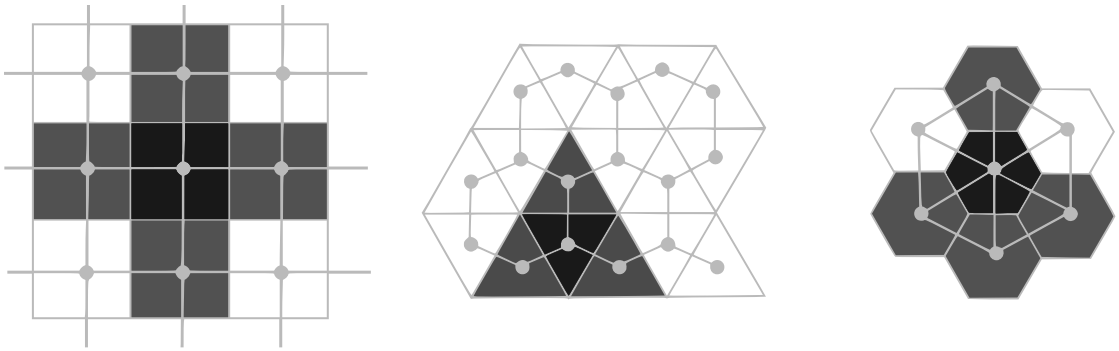


FIGURE 2.4 – Représentation des trois pavages réguliers possibles sur $\mathbb{Z}^2$ (carré, triangle, hexagone). Le maillage de chaque pavage est représenté en gris clair. En particulier, le maillage du pavage carré est carré. L'ensemble des voisins d'une cellule (gris foncé) est également représenté (en gris). Extrait de [CMC07a]

Notions topologiques : pavage et connexité dans le domaine discret

Un espace discret $\mathbb{Z}^n$ est une décomposition du plan de dimension $n \geq 2$ en *cellules*, et l'ensemble de ces cellules forment un *pavage*. Une cellule peut être représentée par son centre de gravité, que l'on appelle *point discret*, et l'ensemble des points discrets d'un pavage forme un *maillage*. Par la suite, on travaillera sur des pavages vérifiant les trois propriétés suivantes :

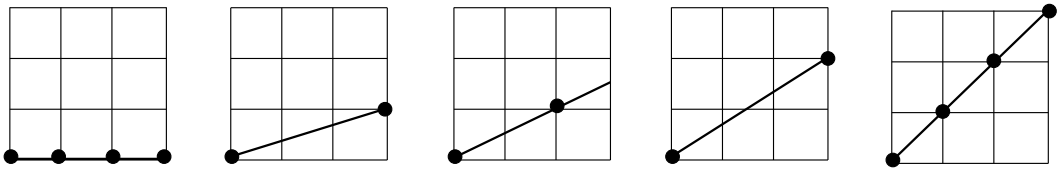
1. pavages réguliers : les cellules correspondent à des formes régulières (i.e. des polygones dont tous les côtés et tous les angles sont égaux) et dont les sommets sont en contact avec un nombre fini de sommets appartenant à d'autres cellules (ce qui remplit l'espace sans recouvrement) ;

2. de dimension 2, c'est-à-dire que l'image correspond à une partie de $\mathbb{Z}^2$ (par la suite, on pourra utiliser le terme *pixel* pour désigner les cellules) ;

3. dont les cellules sont facilement adressables. Pour cela on utilisera un pavage carré afin d'adresser directement les éléments par un couple de coordonnées (x, y) (le maillage d'un pavage carré fournit une base orthonormée contrairement au maillage d'un pavage triangulaire ou hexagonal).

Par la suite, nous utiliserons un pavage carré semblable à celui représenté dans la figure 2.4. Plus précisément, on considérera une image numérique comme l'application $f : E \to F$, où $E \subset \mathbb{R}^2$ correspond au domaine de définition de l'image, et F correspond à l'ensemble des couleurs des pixels. Nous considérerons en général que $E = \mathbb{Z}^2$, c'est-à-dire correspondant à un pavage carré. Le contenu des pixels dépend de la nature de l'image[1]. Dans le cas général, on considère $f : \mathbb{Z}^2 \to \mathbb{R}$.

[1]Dans le cas d'une image binaire, l'ensemble $F = \{0, 1\}$. Dans le cas d'une image dont les couleurs sont encodées par RGB, l'ensemble F correspond à $\{0, 255\}^3$.



(a) $\frac{q}{p} = \frac{0}{1}$ (b) $\frac{q}{p} = \frac{1}{3}$ (c) $\frac{q}{p} = \frac{1}{2}$ (d) $\frac{q}{p} = \frac{2}{3}$ (e) $\frac{q}{p} = \frac{1}{1}$

FIGURE 2.5 – Représentation des différents angles discrets (p, q) qui composent la suite de Farey F_3. Cette suite permet de définir l'ensemble des angles possibles dans $[0, \frac{\pi}{4}]$ pour un pavage (3×3).

Les notions topologiques dans le domaine discret sont définies à partir de la notion de *voisinage* et de *connexité* [Ros70 ; Ros79]. Soient P et Q, deux points définis par leurs coordonnées (x, y) dans un pavage carré. P et Q sont deux points voisins si une et une seule de leurs coordonnées diffère d'une unité. En particulier, le point P possède quatre voisins qui correspondent aux points de coordonnées $(x-1, y), (x+1, y), (x, y-1), (x, y+1)$. Dans ce cas, on parle de 4-*connexité*. La notion de la connexité permet de définir un *chemin* comme une suite de points de telle manière que deux points consécutifs de ce chemin soient voisins.

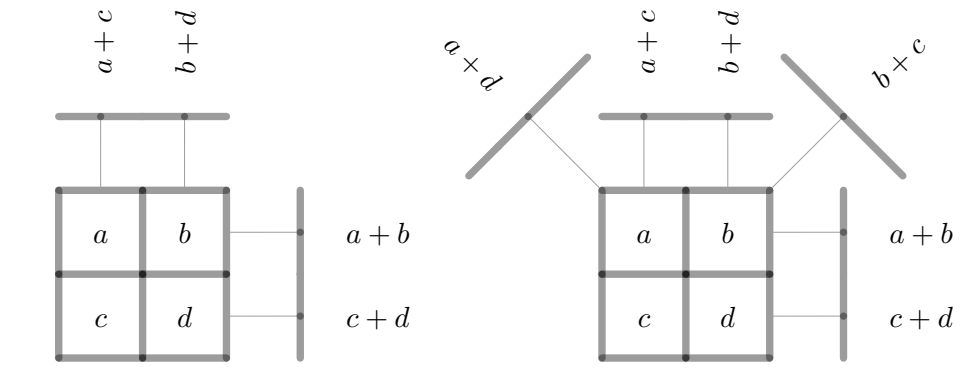
Pour finir, on définit une *composante connexe* comme étant un ensemble maximal S de points discrets tel que pour tout couple de points (P, Q) de S, il existe un chemin reliant P à Q dont tous les points appartiennent à S.

Angle discret et droite discrète

Nous allons à présent introduire les représentations discrètes des angles et des droites. Dans cette section, on cherche à déterminer les points d'intersection entre le maillage défini par les points d'un ensemble discret de taille $(N \times N)$, et une droite d'équation $y = ax + b$. Pour que cette intersection ne soit pas vide, il est nécessaire que la pente de la droite soit de la forme :

$$0 \leq \frac{q}{p} \leq 1 \,, \tag{2.5}$$

avec $p, q \in \mathbb{Z}$ et où p et q sont des entiers premiers entre eux, vérifiant $q \leq p \leq N$. L'ensemble des pentes des droites possibles défini par l'équation (2.5) sur $[0, \frac{\pi}{4}]$ forment une suite de Farey d'ordre N, notée F_N [Fra24]. Les autres droites sont obtenues par symétrie. Une suite de Farey F_N est l'ensemble des fractions irréductibles comprises entre 0 et 1, ordonnées de manière croissante et dont le dénominateur n'est pas plus grand que N. Chaque fraction d'une suite de Farey correspond à un vecteur $[p, q]$ de $\mathbb{Z}^2$. En particulier, MINKOWSKI a montré que si l'on considère deux vecteurs de Farey consécutifs de F_N, alors il ne peut y avoir de point dans le parallélogramme formé par ces deux vecteurs [Min68]. Par exemple, la suite de Farey d'ordre 3 permet d'obtenir l'ensemble des pentes possibles pour un pavage de (3×3), et correspond à :



(a) Exemple non inversible (b) Exemple inversible

FIGURE 2.6 – Exemple de deux représentations d'une image discrète (2×2) par un ensemble allant de quatre à six valeurs de projection, calculées suivant les directions $(p, q) = \{(0, 1), (1, 1), (0, 1), (-1, 1)\}$. Ces valeurs de projections correspondent à la somme des valeurs des pixels (représentées par les lettre $\{a, b, c, d\}$) traversés par la droite de projection.

$$F_3 = \left\{ \frac{0}{1}, \frac{1}{3}, \frac{1}{2}, \frac{2}{3}, \frac{1}{1} \right\} . \tag{2.6}$$

La figure 2.5 représente de manière géométrique les droites issues des pentes générées par F_3 dans un pavage carré de taille (3×3). Visuellement, une direction (p, q) correspond à la droite reliant le point d'origine avec le point obtenu par un décalage horizontal de p, et par un décalage vertical de q.

Dans la suite de nos travaux, nous utiliserons le terme *direction discrète* pour désigner le couple d'entier $(p, q) \in \mathbb{Z}^2$, premiers entre eux, correspondant à la direction de la droite de pente $\frac{q}{p}$. Quant aux angles discrets, ils seront nécessaires pour définir les versions discrètes de la transformation de RADON (sections 2.2 et 2.3).

2.1.3 Méthode algébrique de reconstruction d'une image discrète

Nous allons dans cette section utiliser une approche algébrique afin d'introduire le processus de reconstruction d'une image discrète, à partir d'un ensemble de projections obtenues à différents angles discrets. La figure 2.6 représente une image discrète composée de quatre pixels identifiés par leurs valeurs $\{a, b, c, d\}$. Dans cet exemple, les valeurs des projections correspondent à la somme des valeurs des pixels traversés par les droites de projections. Deux questions se posent alors : (i) est-il possible de déterminer de manière unique la solution du problème ? (ii) quelle méthode utiliser pour reconstruire cette solution efficacement ?

Ce problème peut être vu comme un problème d'algèbre linéaire. Dans cette représentation, les pixels de l'image forment les inconnues à reconstruire tandis que les projections correspondent aux équations linéaires. Dans l'exemple proposé de la figure 2.6a, on sou-

haite représenter l'image par ses projections verticales et horizontales. Posons le problème sous la forme d'un système d'équations linéaires à 4 équations et 4 inconnues, auxquelles on affecte des valeurs aux projections :

$$\left\{ \begin{array}{ll} a+b & =5 \\ c+d & =5 \\ a+c & =4 \\ b+d & =6 \end{array} \right\} . \tag{2.7}$$

On peut écrire ce système d'équations linéaires sous la forme matricielle : $\mathbf{A}x = y$, où $\mathbf{A}$ est la matrice de projection de taille 4×4, x est un vecteur colonne à quatre inconnues et y contient les valeurs des projections. Cela correspond à la multiplication matricielle suivante :

$$\begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix} \begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix} = \begin{pmatrix} 5 \\ 5 \\ 4 \\ 6 \end{pmatrix} . \tag{2.8}$$

Dans cet exemple, la matrice $\mathbf{A}$ n'est pas inversible (son déterminant est nul). Seulement trois équations du système sur quatre sont indépendantes. En effet, la somme de la première équation avec la deuxième est égale à la somme de la troisième avec la quatrième. En conséquence, la reconstruction depuis ces projections fournit une infinité de solutions, tant que la valeur d'un pixel n'est pas connue. Autrement dit, l'ensemble des projections mesurées n'est pas suffisant pour déterminer de manière unique une solution. En revanche, un ensemble de projections suffisant est représenté dans la figure 2.6b. Dans cet exemple, on rajoute deux nouvelles mesures suivant les directions des diagonales. En conséquence, on obtient un système surdéterminé que l'on peut représenter sous forme matricielle :

$$\begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix} \begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix} = \begin{pmatrix} 5 \\ 5 \\ 4 \\ 6 \\ 3 \\ 7 \end{pmatrix} . \tag{2.9}$$

Le système comporte à présent 4 inconnues pour 6 équations. On est capable de déterminer de manière unique une solution de reconstruction à travers la méthode suivante :

$$\mathbf{A}x = y, \tag{2.10}$$

$$\mathbf{A}^\intercal \mathbf{A}x = \mathbf{A}^\intercal y, \tag{2.11}$$

$$x = [\mathbf{A}^\intercal \mathbf{A}]^{-1} \mathbf{A}^\intercal y, \tag{2.12}$$

ce qui donne $(a = 1, b = 4, c = 3, d = 2)$. Bien que cette méthode fonctionne, elle n'est pas efficace. En effet, pour déterminer l'unicité de la solution, il faut calculer le déterminant de la matrice $[\mathbf{A}^\intercal\mathbf{A}]$ puis inverser cette matrice, en utilisant par exemple la méthode de Gauss-Jordan. Une autre méthode consiste à utiliser la décomposition LU, qui consiste à décomposer $\mathbf{A}$ en deux matrices triangulaires L et U afin de faciliter les calculs par rapport à la méthode de Gauss-Jordan. Puisque $\mathbf{A} = LU$, $\det \mathbf{A} = \det L \det U$, ce qui est simple à déterminer dans le cas des matrices diagonales. La complexité de calcul du déterminant est cubique. Une fois la matrix $\mathbf{A}$ décomposée, il est possible de résoudre équation (2.11) en $\mathcal{O}(n^2)$ opérations, pour une matrice carrée de taille n. Ces calculs peuvent devenir coûteux lorsque la taille des images augmentent.

Dans cette section, deux critères d'efficacité ont donc été mis en avant : (i) la détermination efficace d'un critère de reconstruction (nous avons vu que le calcul du déterminant est coûteux) ; (ii) la méthode d'inversion doit avoir une complexité faible (i.e. meilleure que quadratique). Dans la suite de ce chapitre, nous détaillerons deux méthodes conçues à partir d'une approche par géométrie discrète. Il s'agit de versions discrètes et exactes de la transformation de Radon : la FRT et la transformation Mojette (respectivement étudiées dans les sections 2.2 et 2.3). Plus particulièrement, l'approche par géométrie discrète nous permettra de définir pour ces deux méthodes, des critères de reconstruction efficaces, ainsi que des algorithmes de reconstruction de faible complexité.

2.2 Code MDS par transformation de Radon finie

Nous avons vu dans la section précédente que la transformation de Radon continue constitue une application mathématique qui possède une opération inverse. Cependant, puisque nous allons traiter des données numériques, il est nécessaire de définir une version discrète de cette application.

Cette section présente la FRT qui est une version discrète et exacte de la transformation de Radon définie par Matúš et Flusser [MF93]. La particularité de cette transformation est de considérer une géométrie périodique, caractérisée par la taille du support. Matúš et Flusser ont montré que cette propriété permet de construire un nombre fini de projections. Le calcul de ces projections, ainsi que la méthode de reconstruction de l'image, seront les sujets d'étude de la section 2.2.1. La section 2.2.2 permettra de définir les fantômes, qui sont des éléments du noyau de la transformée [Kat78]. Les fantômes ne sont pas spécifiques à la FRT, aussi nous les réutiliserons par la suite pour la transformation Mojette (section 2.3), et pour le calcul de nouvelles projections (chapitre 6). Chandra et al. ont proposé un algorithme qui permet de supprimer ces fantômes afin de reconstruire la donnée [CS08 ; Cha+12]. Nous étudierons cet algorithme afin de concevoir un code à effacement. Dans cette section, la présentation des codes basés sur la FRT repose sur les travaux de Normand et al. [Nor+10]. En particulier, ces travaux proposent une mise en œuvre de la FRT qui fournit un code à effacement MDS.

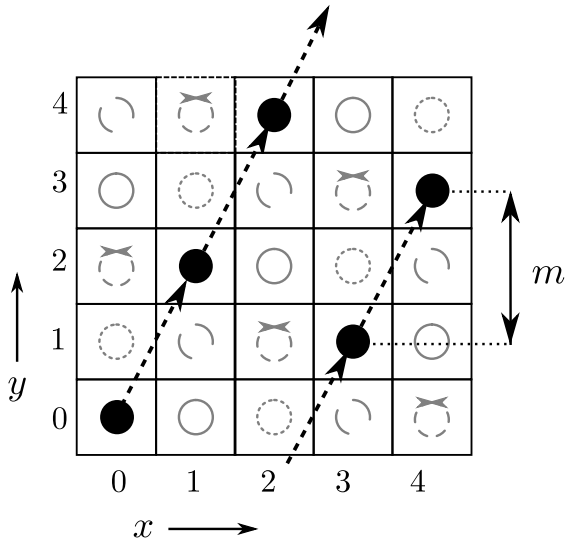


Figure 2.7 – Représentation géométrique des mesures de projection FRT $m = 2$. La mesure surlignée (en noir), d'index $t = 0$, correspond à la somme des éléments sur la droite $y \equiv 2x \pmod 5$ (i.e. $p = 5, m = 2, t = 0$). On observe que l'ensemble des mesures d'une projection traverse chaque pixel de l'image une seule fois. Cette figure est adaptée de [CSG08].

2.2.1 Transformation de Radon finie

Dans cette première section dédiée à la FRT, nous présenterons tout d'abord la définition de la transformation, puis nous nous intéresserons à la méthode de reconstruction de la grille à partir de la transformée, tel que le décrit Matúš et Flusser [MF93].

Transformation de Radon finie directe

La transformation de Radon finie, pour *Finite Radon Transform* (FRT), est une version discrète, exacte et périodique de la transformation de Radon continue, définie mathématiquement par Matúš et Flusser [MF93]. Considérons une fonction discrète $f : \mathbb{Z}^2 \to \mathbb{R}$, correspondant à une grille carrée de paramètre $p \in \mathbb{N}$, où p est premier. Les valeurs de projections $[Rf](m, t)$ sont définies ainsi :

$$[Rf](m,t) = \sum_{x=0}^{p-1} \sum_{y=0}^{p-1} f(x, (mx + t) \pmod p), \tag{2.13}$$

$$[Rf](p,t) = \sum_{x=0}^{p-1} \sum_{y=0}^{p-1} f(t, y), \tag{2.14}$$

avec $x, y, m, y \in \mathbb{Z}_p$. La variable t correspond à l'index de l'élément dans la projection. De manière géométrique, elle correspond à la valeur de translation de la projection

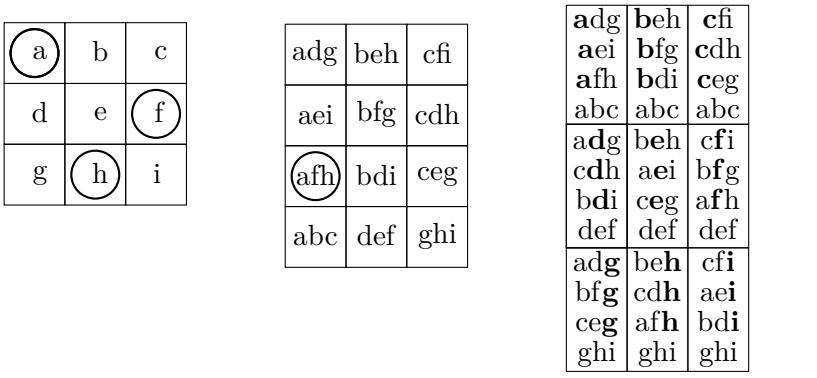


Image $f(x, y)$ (a) Radon $R(t, m)$ (b) Rétroprojection $f'(x, y)$ (c)

FIGURE 2.8 – Représentation de la FRT et de sa rétroprojection. (a) on applique la FRT sur une image carrée de paramètre $p = 3$. Les valeurs des pixels sont symboliques : $\{a, b, \ldots, i\}$. (b) correspond à la transformée, c'est-à-dire aux $p + 1 = 4$ projections. Un exemple de mesure est ici représenté par les valeurs encerclées. Il s'agit du calcul de FRT pour $t = 0$ et $m = 2$ (i.e. la pente vaut deux). La juxtaposition des lettres indique leur somme. Par exemple, afh correspond à la somme $a + f + h$. (c) représente la rétroprojection f'. On remarque que chaque élément est composé de I_{sum} et de p fois la valeur initiale du pixel.

suivant l'axe des y. Quant à m, elle représente la pente de la droite de projection. Notons que l'équation (2.13) qui calcule les n premières projections, contient un opérateur modulo permettant de rendre les droites de projection périodiques, de période p. Quand à l'équation (2.14), elle calcule la $(p + 1)$-ème projection qui correspond à la somme des éléments de la grille suivant l'horizontale. Dans le domaine de RADON, chacune des $(p + 1)$ projections indexées par m contient p valeurs correspondant à un décalage de t suivant l'axe des y. La figure 2.7, page 53 représente l'une de ces mesures pour la droite d'équation $y \equiv 2x \pmod 5$ sur un pavage carré défini par $p = 5$, avec $t = 0$. Dans le cas général, ces droites de projection ont pour équation :

$$y \equiv mx + t \pmod p \,. \tag{2.15}$$

MATÚŠ et FLUSSER [MF93] ont montré que les $(p + 1)$ projections définies aux équations (2.13) et (2.14) permettent d'échantillonner parfaitement l'image. En effet puisque p est premier, les mesures d'une projection parcourent l'ensemble des pixels une et une seule fois. La figure 2.8 illustre cela pour les 5 mesures de la projection $m = 2$. En conséquence, la somme des valeurs d'une projection correspond à la somme des pixels de l'image. Cette valeur, notée I_{sum} sera utilisée dans la suite.

La figure 2.8 montre la FRT d'une image (3×3) dans laquelle la valeur des pixels correspond à des lettres. En particulier, les pixels en jeu dans le calcul de la mesure $[Rf](m = 2, t = 0)$ sont encerclés. Pour l'inversion, il est nécessaire d'avoir $(p + 1)$

projections afin de pouvoir reconstruire une image $(p \times p)$ [MF93]. Dans la suite, nous détaillons une méthode de reconstruction de l'image algébrique.

Reconstruction de l'image par FRT inverse

La méthode de FRT inverse permet de retrouver l'image initiale à partir des données de projections. Chaque paire de pixels distincts n'apparaît que dans une mesure de projection puisque p est premier. Par exemple a et h n'apparaissent que dans la combinaison $a+f+h$. En conséquence, la somme de toutes les mesures issues d'un pixel contient : (i) $p+1$ fois ce pixel ($p+1$ étant le nombre de directions distinctes, donc le nombre de mesures qui contiennent ce pixel) ; (ii) et une fois et une seule chacun des autres pixels de l'image. MATÚŠ et FLUSSER [MF93] proposent une méthode d'inversion basée sur le fait que l'opérateur FRT est son propre dual. Dans ce cas, l'inversion implique de réaliser une rétroprojection. Cette opération consiste à appliquer l'opérateur FRT sur la transformée $[Rf](m,t)$, le long des droites de projection $\mathcal{L}_{m',t}$ d'angle $m' = p - m$ (i.e. opposé à m). On obtient une image f' dont chaque valeur des pixels correspond à $f'(x,y) = (f(x,y) \times p) + I_{sum}$, où $f(x,y)$ est la valeur d'origine de l'élément (x,y), et I_{sum} correspond à la somme de tous les pixels de l'image. Une représentation de la rétroprojection f' est donnée dans la partie droite de la figure 2.8. L'image initiale f est alors retrouvée en filtrant l'image de rétroprojection par la soustraction des valeurs de ses pixels par I_{sum}, puis par la division par p. L'équation correspondante à cette opération inverse est la suivant :

$$f(x,y) = \frac{1}{p}\left(f'(x,y) - I_{sum}\right), \tag{2.16}$$

$$= \frac{1}{p}\left(\sum_{\mathcal{L}_{m',t}} [Rf](m',t) + [Rf](p,x) - I_{sum}\right). \tag{2.17}$$

Bien que simple à mettre en œuvre, cette méthode n'est pas efficace puisque sa complexité algorithmique est $\mathcal{O}(p^3)$. MATÚŠ et FLUSSER [MF93] ont cependant démontré que la FRT conserve toutes les propriétés de la transformation continue de RADON. En particulier, ils définissent mathématiquement une version discrète du théorème de la tranche centrale. Comme vu précédemment dans le domaine continu, ce théorème permet de faire le lien entre la transformation de FOURIER 1D d'une projection, et la tranche orthogonale dans le domaine de FOURIER de f. La différence principale avec la transformation de RADON continue est que l'échantillonnage optimal de la FRT permet de recouvrir entièrement le domaine de FOURIER sans avoir besoin d'interpolation. La reconstruction consiste alors à (i) calculer la transformée 1D de FOURIER pour chaque projection afin de remplir l'espace de FOURIER ; (ii) calculer la transformée de FOURIER 2D inverse pour obtenir l'image. Il est ainsi possible de réduire la complexité à celle de la FFT, i.e. $\mathcal{O}(p^2 \log_2 p)$ [KS06]. CHANDRA et SVALBE [CS14] ont proposé une extension de cette méthode, basée sur la transformation de FOURIER modulaire. Cette méthode permet de remplacer une racine complexe de l'unité par une racine entière de l'unité.

2.2.2 Représentation partielle et fantôme discret

Les travaux de Matúš et Flusser [MF93] ont montré que les $(p+1)$ projections sont nécessaires pour déterminer de manière unique la solution du problème de reconstruction. En conséquence, une représentation partielle correspond au cas où l'ensemble de projections n'est pas suffisant pour reconstruire l'image de manière unique. Dans cette situation, le problème de reconstruction devient sous-déterminé et possède soit plusieurs solutions, soit aucune. Pour comprendre cette situation, nous nous intéresserons à l'*espace nul* qui correspond au noyau de l'opérateur. Plus particulièrement, nous étudierons les *fantômes* qui correspondent aux éléments de cet espace nul.

Introduction aux fantômes

Dans le domaine continu, Bracewell et Roberts [BR54] définissent le concept de *distribution invisible*, qui fait référence au terme *fantôme*, utilisé plus tard par Cornwell [Cor82] dans le cas de la transformation de Radon. D'une manière générale, on définit un fantôme comme une fonction $g\colon \mathbb{R}^2 \to \mathbb{R}$ telle que [Bra56] :

$$\int_{\mathcal{L}} g(x,y)d\ell = 0\ . \tag{2.18}$$

Les fantômes correspondent ainsi à des éléments dont la somme vaut 0 suivant la direction de projection $\mathcal{L}^2$. De manière analogue, ils correspondent à des éléments de l'image générés dès lors que des projections manquent. Dans la pratique, puisque la transformation de Radon continue est un problème *mal posé*, il existe toujours des projections manquantes (à cause de l'échantillonnage dû à la nature des capteurs). En conséquence, les fantômes empêchent l'aboutissement du processus de reconstruction vers une solution unique. De par sa définition, un fantôme correspond à une distribution de la valeur de projection manquante sur les pixels de l'image, selon un motif particulier.

Dans le domaine discret, on définit l'espace nul FRT comme étant le noyau de l'opérateur FRT $[R]$, c'est-à-dire, l'ensemble des éléments de l'image tel que leur projection suivant une pente m vaut 0 :

$$ker(R) = \left\{ g\colon Z_p^2 \to \mathbb{R} \mid [Rg](m,t) = 0 \right\}\ . \tag{2.19}$$

Les fantômes ont un rôle essentiel dans la compréhension des transformées et dans l'élaboration des méthodes de reconstruction à partir d'une représentation partielle. Nous nous intéresserons plus particulièrement aux travaux de Chandra et al. [CSG08] qui étudient la structure des fantômes afin de comprendre comment ils se superposent sur l'image. Nous verrons par la suite une méthode pour supprimer les fantômes de l'image et ainsi reconstruire sa valeur initiale.

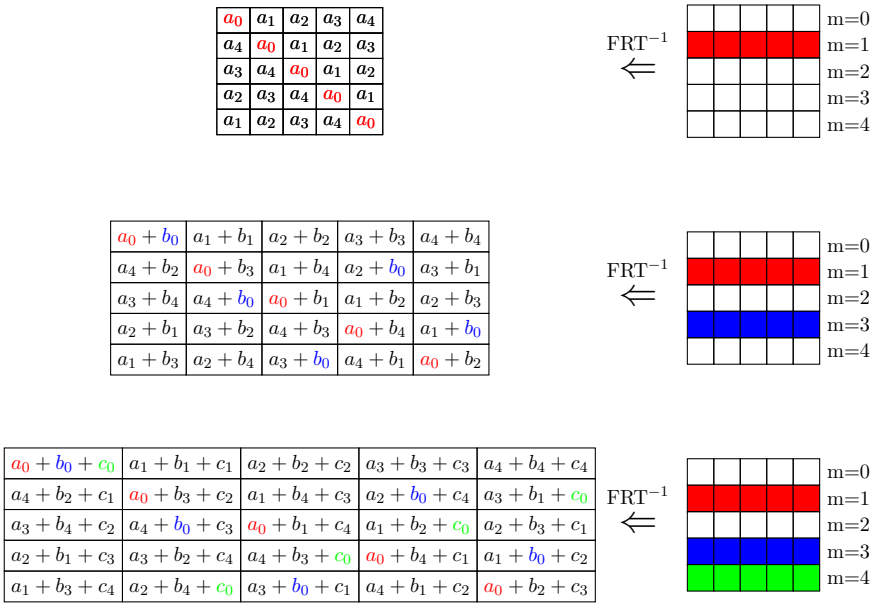


Figure 2.9 – Représentation des distributions circulantes des fantômes a, b, c générés par l'effacement respectif des projections $m = 1, 3, 4$. Les grilles de gauche correspondent à la superposition des fantômes sur une image (5×5) après reconstruction depuis les projections FRT de droite. Chaque étape correspond à un nouvel effacement, représenté par une ligne colorée. Figure inspirée de Chandra et al. [CSG08].

Structure des fantômes et distribution sur l'image

Par définition, l'effacement d'une projection FRT entraîne la création d'un fantôme dans l'image. Afin de déterminer des méthodes de reconstruction de l'image en cas de perte de projections, il est nécessaire d'en déterminer la structure. Soit une projection $a = \{a_0, \ldots, a_{p-1}\}$ d'index m_a de l'espace de Radon $[Rf]$. Chandra et al. [CSG08] ont montré que la reconstruction de l'image à partir d'un domaine de Radon partiel (où la projection a est manquante) entraîne une distribution circulaire des valeurs de a sur l'image. La figure 2.9 montre la distribution des valeurs du fantôme dans une image (5×5) lorsque la représentation de l'image par les projections est partielle. Dans cette figure, la première représentation correspond à la situation où la projection $m = 1$ est manquante. Dans ce cas, on remarque que les valeurs de la projection a se superposent à l'image, avec un décalage circulaire de m_a sur chaque ligne de l'image.

Chandra et al. [Cha+12] ont par la suite démontré que cette structure est caractérisée par une matrice circulante. Plus particulièrement, chaque projection manquante entraîne la génération d'une nouvelle distribution de fantômes dont les décalages sont caractérisés par l'index de cette projection. Ainsi, lorsque plusieurs projections manquent dans le domaine de Radon, l'image contient une superposition de ces distributions. La figure 2.9

[2] le nom "fantôme" fait référence au fait que ces éléments sont invisibles dans l'espace de Radon

montre le cas où deux puis trois projections sont manquantes. Dans la suite, nous nous intéresserons aux méthodes de résolution qui permettent de supprimer ces fantômes afin de reconstruire l'image à partir d'un espace de Radon partiel.

Reconstruire l'information manquante

Plusieurs algorithmes ont été proposés pour reconstruire l'image à partir d'une représentation partielle [CSG08 ; Nor+10]. La première méthode de reconstruction proposée par Chandra et al. [CSG08] consiste à supprimer les fantômes dans l'image afin de retrouver la valeur initiale. Cette méthode nécessite de connaître a priori une partie de l'image. Chandra et al. [CSG08] proposent de calibrer une zone de r lignes de l'image à zéro afin de pouvoir isoler la valeur des fantômes dans cette zone lors de la reconstruction. Il est ainsi possible de supporter la perte de r projections.

Dans le cas où une seule projection manque, la valeur de ses éléments est alors directement lisible sur chaque ligne de redondance. Il faut cependant considérer le décalage circulaire dû à la structure circulaire de la distribution des fantômes. Ce décalage est caractérisé par m_a et l'index de la ligne, comme le montre la première représentation de la figure 2.9. Lorsque plusieurs projections manquent, le processus de reconstruction proposé par Chandra et al. [CSG08] se complique. Il met en jeu trois opérations pour la détermination d'une projection : (i) un décalage cyclique sur chaque ligne afin de synchroniser le premier élément a_0 des fantômes sur le premier pixel de chaque ligne ; (ii) la soustraction des lignes afin d'enlever la contribution du fantôme dans ces lignes de redondance ; (iii) une intégration des valeurs obtenues afin de supprimer le décalage généré par la soustraction précédente. Pour le lecteur qui souhaite plus d'informations sur cette méthode, toutes les étapes sont clairement indiquées dans les travaux de Chandra et al. [CSG08].

2.2.3 Code à effacement par transformation de Radon finie

Cette section décrit à présent comment utiliser la FRT comme un code à effacement. Plusieurs travaux ont été réalisés pour définir ce code sous une forme systématique, et non systématique [Nor+10 ; Par+12 ; Per+12].

Forme non-systématique

La figure 2.10 représente la mise en œuvre de la FRT en non-systématique. Dans l'objectif d'obtenir un espace de Radon de même taille que l'image, on impose une contrainte de conception qui consiste à faire correspondre la dernière colonne à une colonne de parité. Cela entraîne deux conséquences : (i) la dernière projection est nulle de part la parité horizontale ; (ii) la dernière colonne du domaine transformé correspond également à un colonne de parité. En conséquence, il n'est alors pas nécessaire de garder ces informations.

Le code non-systématique est alors conçu de la manière suivante. La donnée est rassemblée dans une zone de taille $k \times (p-1)$. On représente r lignes (*imaginaires* dans l'implémentation) contenant des valeurs nulles. La colonne de parité permet de limiter

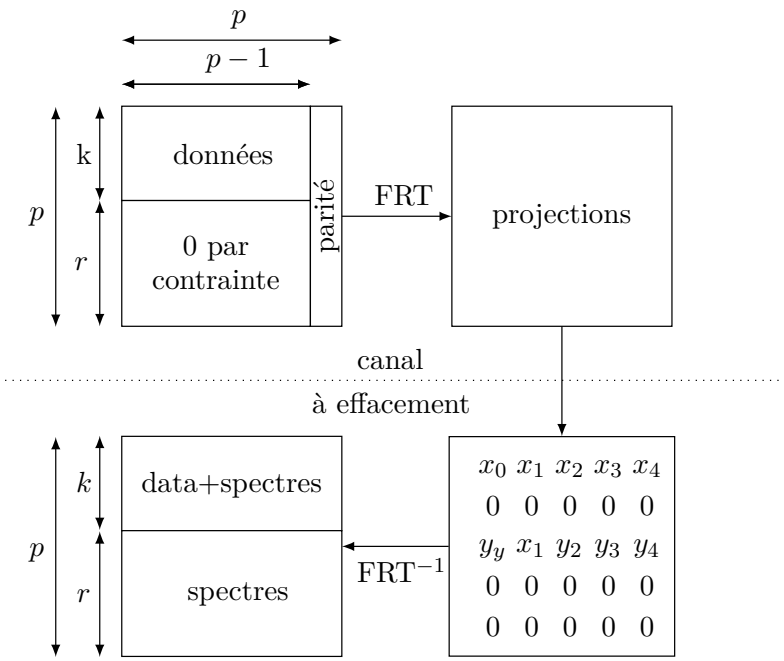


Figure 2.10 – Représentation de la mise en œuvre du code à effacement non-systématique basé sur la FRT. La colonne de parité correspond à une contrainte pour concevoir un code MDS. Les données encodées correspondent aux p projections de l'espace de Radon. Figure inspirée depuis [Nor+10].

notre représentation de l'image avec p projections, qui correspondent à l'ensemble des données encodées dans ce formalisme.

Lorsque des effacements surviennent sur ces données encodées, cela se traduit par la suppression de certaines projections. Comme expliqué précédemment, la reconstruction de l'image par cette représentation partielle va générer des fantômes sur l'ensemble de l'image. En particulier, la valeur de ces fantômes est contenue dans les r lignes de redondance (puisqu'elles contenaient uniquement des valeurs nulles précédemment). Si le nombre d'effacement est au plus r, on peut utiliser une technique de suppression des fantômes, telle que celle décrite précédemment, pour reconstruire la zone de $k \times (p-1)$ éléments de données.

Forme systématique

La mise en œuvre du code systématique est illustrée dans la figure 2.11. Le principe de cette mise en œuvre est d'intégrer les k lignes de données dans l'ensemble des p lignes encodées. La construction de cette version du code est semblable à ce que l'on a vu précédemment. La différence réside dans la zone de redondance. Il est nécessaire de définir r lignes de redondance de telle sorte que r projections contigües soient nulles. En conséquence, l'image devient un fantôme pour les projection. Puisque l'opérateur FRT

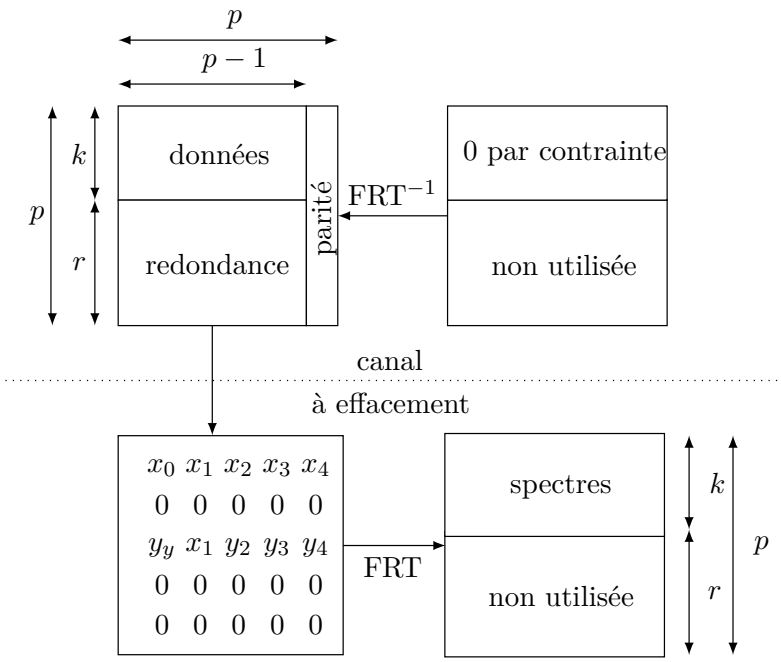


FIGURE 2.11 – Représentation de la mise en œuvre du code à effacement systématique basé sur la FRT. Les données encodées correspondent aux k lignes de données auxquelles on ajoute r lignes de redondance.

est son propre dual, cette opération correspond à générer des fantômes dans l'image. Une méthode pour déterminer la valeur de ces fantômes correspond à supprimer r fantômes dans le domaine de RADON. On peut utiliser la technique de suppression proposée par CHANDRA et al. [CSG08] par exemple.

Quand des effacements suppriment des lignes de données, des fantômes sont générés parmi les projections. On utilise alors le même algorithme que précédemment pour retrouver la valeur des données.

2.2.4 Relations avec d'autres codes MDS

La FRT possède des liens avec certains codes MDS. NORMAND et al. [Nor+10] ont montré que l'opérateur FRT peut être représenté sous la forme d'une matrice de VANDERMONDE. Cette représentation est également utilisée pour définir les codes de REED-SOLOMON [RS60]. Quant aux Array codes, nous verrons qu'au delà de partager une approche géométrique, ils représentent un cas particulier de la FRT.

Relation algébrique avec les codes de Reed-Solomon

Les travaux de NORMAND et al. [Nor+10] offrent une représentation polynomiale de l'image f et de l'opérateur FRT. Plus précisément, l'image est représentée un vecteur de

p polynômes, où chaque polynôme $P(x)$ de degré p représente une ligne. On note $\boldsymbol{f}$ le vecteur polynomial représentant l'image, défini tel que :

$$\boldsymbol{f}^\intercal = (P_0, \ldots, P_{p-1}) \ , \tag{2.20}$$

avec $P_i(x) = f(0,i)x^0 + \cdots + f(p-1,i)x^{p-1}$, et $\boldsymbol{f}^\intercal$ correspond à la transposée de $\boldsymbol{f}$.

Le support étant périodique de période p, on a $(p,p) = (0,0)$. Cela signifie que pour chaque projection $x^p \equiv x^0$. En conséquence, les polynômes $P_i \mid i \in Z_p$ sont définis (mod $x^p - 1$). Dans cette représentation, la multiplication d'un polynôme par x est équivalent à un décalage cyclique de la ligne associée. En particulier, l'opérateur FRT R_m correspond à une application qui somme les éléments de l'image après avoir appliqué des décalages circulaires ml sur chaque ligne l. On définit alors l'opérateur ainsi :

$$R_m(x) = P_0(x) + x^{-m}P_1(x) + \cdots + x^{-(p-1)m}P_{p-1}(x) \ , \tag{2.21}$$

avec $m \in \{0, \ldots, p-1\}$. Il est possible de poser ce problème sous une forme matricielle :

$$R_m(\boldsymbol{f}) = \begin{pmatrix} 1 & 1 & \cdots & 1 \\ 1 & x^{-1} & \cdots & x^{-(p-1)} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x^{-(p-1)} & \cdots & x^{-(p-1)^2} \end{pmatrix} \times \begin{pmatrix} P_0(x) \\ P_1(x) \\ \vdots \\ P_{p-1}(x) \end{pmatrix} \ , \tag{2.22}$$

avec $m \in \mathbb{Z}_p$ (voir équation (2.13)). Les coefficients de R_m correspondent aux coefficients d'une matrice de Vandermonde $\boldsymbol{R} = \boldsymbol{V}(x^{-0}, \ldots, x^{-(p-1)})$. Toute sous matrice d'une matrice de Vandermonde est inversible. En conséquence, on est capable de reconstruire r lignes manquantes en inversant la matrice dont les coefficients correspondent à ces lignes. Normand et al. [Nor+10] proposent d'inverser la matrice en utilisant la décomposition LU [Tur66].

Bilan de la FRT

Nous avons vu que la FRT correspond à une version discrète, exacte et périodique de la transformation de Radon continue. Cette propriété lui permet de fournir un code MDS en se basant sur une contrainte de construction mettant en jeu une colonne de parité, et un calibrage des données correspondant à des lignes de redondance. En particulier, nous avons vu deux algorithmes de reconstruction efficaces : (i) en utilisant la FFT depuis le théorème de tranche centrale ; (ii) par la méthode de suppression des fantômes. Ces méthodes ont ainsi permis de déterminer deux versions de code à effacement selon une construction systématique et non-systématique.

Dès lors, la FRT peut tout à fait être mise en œuvre dans un contexte de transmission et de stockage de l'information. Dans ce contexte, la propriété MDS du code à effacement par FRT permet de générer une quantité minimum de redondance pour fournir une certaine tolérance à l'effacement. Dans une autre mesure, l'efficacité des algorithmes proposés permettent une implémentation performante des opérations d'encodage et de décodage.

2.3 Code à effacement par transformation Mojette

Dans cette section, nous allons nous intéresser à un code à effacement basé sur la transformation Mojette. Tout comme la FRT, cette transformation correspond à une version discrète et exacte de la transformation de Radon continue définie dans Radon [Rad17]. Elle a été proposée pour la première fois par Guédon et al. [GBB95] dans le contexte du traitement et du codage psychovisuel. Depuis, cette transformation a été utilisée dans de nombreuses applications liées à l'imagerie numérique (codage, transmission, tatouage). Dans cette thèse, nous proposons de l'utiliser comme code à effacement pour le stockage et la transmission d'information.

Nous décrirons en section 2.3.1 la méthode de calcul des projections Mojette, qui se distinguent des projections FRT par leur géométrie apériodique. Nous verrons que ces projections peuvent fournir une représentation redondante de l'image, nécessaire pour tolérer des effacements. La section 2.3.2 présentera le critère défini par Katz [Kat78] permettant de garantir l'unicité de la solution de reconstruction d'une image rectangulaire, et nous verrons que ce critère est nécessaire pour définir la capacité de reconstruction du code à effacement. La méthode de reconstruction de Normand et al. [NKÉ06] sera également étudiée, et sera utilisée dans le processus de décodage afin de reconstruire l'information. Après cette présentation de la transformation Mojette, nous verrons comment ces éléments permettent de concevoir un code à effacement dans la section 2.3.3.

2.3.1 Transformation Dirac-Mojette directe

La transformation Mojette est une opération linéaire définie par Guédon et al. [GBB95] qui calcule un ensemble de I projections à partir d'une image discrète $f : \mathbb{Z}^2 \mapsto \mathbb{R}$. Bien que cette image discrète peut avoir n'importe quelle forme, nous considèrerons dans la suite une image rectangulaire, composée de $P \times Q$ pixels. Une projection Mojette $\mathcal{M}$ est un ensemble d'éléments appelés *bins*, qui est définie par une direction de projection (p, q), avec $p, q \in \mathbb{Z}$ premiers entre eux (comme expliqué en section 2.1.2). La transformée Dirac-Mojette $M[f]$ est définie par Guédon et al. [GBB95] comme l'ensemble de projections suivant :

$$M_{\{b,p_i,q_i\}}[f] = \sum_{k=0}^{Q-1} \sum_{l=0}^{P-1} f(k,l)\, \Delta(b - lp_i + kq_i) \ , \tag{2.23}$$

où $\Delta(\cdot)$ vaut 1 si $b = lp_i - kq_i$ et 0 sinon. Le paramètre b correspond à l'index du bin de la projection d'angle (p_i, q_i). Plus précisément, équation (2.23) signifie que la valeur des bins de la projection suivant la direction (p_i, q_i) résulte de la somme des pixels situés sur la droite discrète d'équation $b = -kq_i + lp_i$. Comme on peut l'observer, l'opération modulaire que l'on avait utilisée dans l'équation (2.13) de la FRT, n'est pas utilisée ici. Cette propriété apériodique distingue les deux transformations.

La figure 2.12 représente la transformée Mojette sur $\mathbb{F}_2$ d'une grille discrète (3×3) composée d'éléments binaires. Le traitement transforme une image 2D en un ensemble de

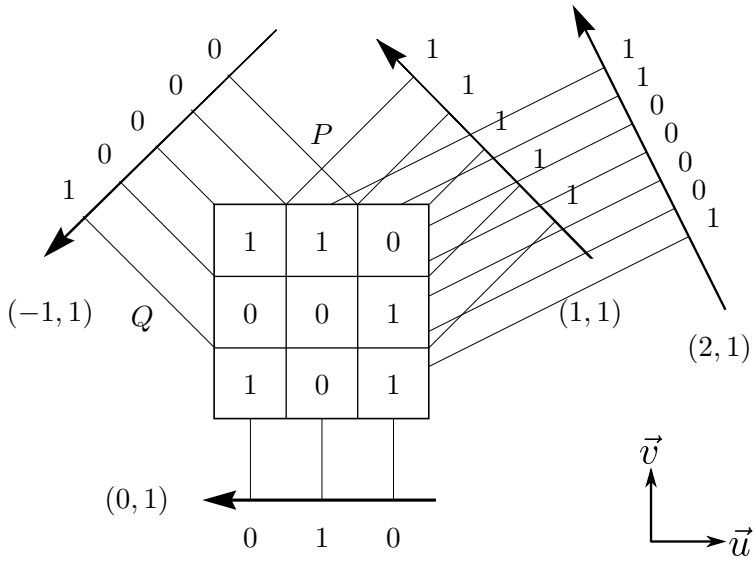


FIGURE 2.12 – Représentation de la transformée Mojette dans $\mathbb{F}_2$. On considère une grille d'image $P \times Q = 3 \times 3$ sur laquelle nous calculons 4 projections dont les directions (p_i, q_i) sont comprises dans l'ensemble $\{(-1,1),(0,1),(1,1),(2,1)\}$. La base utilisée est représentée par les deux vecteurs unitaires $\vec{u}$ et $\vec{v}$.

$I = 4$ projections dont les valeurs des directions sont comprises dans l'ensemble suivant : $\{(-1,1),(0,1),(1,1),(2,1)\}$. Dans l'objectif de simplifier la représentation de cet exemple et des suivants, nous avons choisi le corps $\mathbb{F}_2$, dont l'addition, noté $\oplus$, correspond à un OU exclusif. Cependant, on peut considérer n'importe quel groupe abélien.

Dans l'exemple de la projection de direction $(p = 0, q = 1)$, chaque bin résulte de l'addition des pixels de la grille suivant la direction verticale. Par exemple, le premier bin, situé tout à droite de la projection, vaut $1 \oplus 0 \oplus 1 = 0$. Les autres projections ont la particularité d'avoir une pente non nulle. Par exemple, la valeur des bins des projections d'angles $(-1,1)$ et $(1,1)$ correspond à la somme des pixels suivant les diagonales. En particulier, on remarque que les bins situés aux extrémités des ces projections sont entièrement définis par un seul pixel. Cette remarque sera nécessaire afin de comprendre comment s'applique l'algorithme de reconstruction de NORMAND et al. [NKÉ06] que l'on détaillera dans la prochaine section. Du point de vue de la complexité, puisqu'il faut $\mathcal{O}(I)$ opérations par pixel et que chaque pixel est parcouru, la complexité de la transformation Mojette vaut $\mathcal{O}(PQI)$. NORMAND et al. [Nor+96] précisent que si le nombre de projections I correspond à $log(PQ)$, alors la complexité de la Mojette correspond à celle de la FFT de COOLEY et al. [CLW69].

2.3.2 Reconstruction par transformation Mojette

Dans cette section, nous présenterons le critère défini par KATZ [Kat78] qui permet de déterminer si un ensemble de projections est suffisant pour reconstruire de manière unique

l'image. Par la suite, nous présenterons l'algorithme de reconstruction de Normand et al. [NKÉ06].

Critères de reconstruction

Le premier critère permettant de garantir l'existence d'une solution unique de reconstruction est le critère de Katz [Kat78]. Ce critère n'est défini que pour des images rectangulaires $P \times Q$. Étant donné un ensemble de directions de projection $\{(p_i, q_i)\}$, le critère de Katz déclare que si l'une des conditions suivantes est remplie :

$$\sum_{i=1}^{I} |p_i| \geq P \text{ ou } \sum_{i=1}^{I} |q_i| \geq Q \,, \tag{2.24}$$

alors il existe une unique solution de reconstruction, et cette solution contient les valeurs de la grille qui ont permis de calculer ces projections. Ce critère a été étendu par Normand et al. [Nor+96] pour n'importe quelle image convexe en utilisant la définition des fantômes.

Dans l'exemple de la figure 2.12, si l'on considère un sous-ensemble de 3 projections $\{(p_0, q_0), \ldots, (p_2, q_2)\}$, la seconde condition du critère de Katz donne $\sum_{i=0}^{2} |q_i| = 3$, puisque $q = 1$ pour chaque direction de projection. En conséquence, 3 projections parmi les 4 calculées dans cet exemple suffisent pour reconstruire l'image de manière unique. Autrement dit, la perte d'une projection n'influence pas le résultat du processus de reconstruction. On considère alors que les 4 projections calculées dans cette figure décrivent une représentation redondante de l'image.

Alors que le critère de Katz ne s'applique que sur une image rectangulaire, Normand et al. [Nor+96] ont étendu ce critère à n'importe quelle forme convexe.

Algorithme de reconstruction

Plusieurs algorithmes de reconstruction Mojette ont été proposés [Nor+96 ; NKÉ06 ; Ser+05a] et sont résumés dans [Nor+09]. Nous choisissons de décrire dans la suite, l'algorithme itératif d'inversion Mojette par balayage (*Balayage Mojette Inversion*, BMI) de Normand et al. [NKÉ06] pour son efficacité. Nous verrons en effet que la complexité de décodage d'un symbole est linéaire. Il sera en effet utilisé pour décoder l'information dans le code à effacement, et servira également dans la compréhension de génération de nouvelles projections dans le chapitre 6. Bien que le sens de reconstruction est arbitraire, nous considèrerons dans la suite, une reconstruction de gauche à droite comme proposé par Normand et al. [NKÉ06].

Chaque droite de projection correspond à une équation faisant intervenir un bin et un ensemble de pixels. Dans la section précédente, nous avions observé que les pixels situés dans les coins de la grille définissent entièrement les bins correspondants dans les projections dont la pente n'est pas nulle (i.e. tel que p ou q est différent de 0). Les autres pixels de la grille en revanche, interviennent avec d'autres éléments de l'image dans le calcul des bins de projections. Les droites de projections définissent des équations dont les

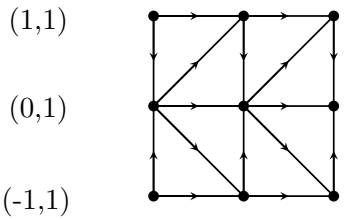


FIGURE 2.13 – Graphe de dépendance des pixels. On considère une image 3×3 ainsi qu'un ensemble de directions de projection (p_i, q_i) à valeur dans $\{(-1,1),(0,1),(1,1)\}$. Les nœuds correspondent aux pixels tandis que les arêtes représentent les dépendances entre eux. Un pixel peut être reconstruit par un bin lorsqu'aucune dépendance ne s'applique sur lui.

inconnues correspondent aux pixels qui n'ont pas encore été reconstruits. Il est possible de reconstruire la valeur d'un pixel lorsqu'il est le seul inconnu de l'équation définie par la droite de projection. En conséquence, il existe des dépendances entre les pixels.

Il est possible de représenter ces dépendances en utilisant un graphe. La figure 2.13 est un graphe orienté qui représente les dépendances entre les pixels d'une grille 3×3, étant donné un ensemble de directions de projection $\{(1,1),(0,1),(-1,1)\}$. Les nœuds de ce graphe correspondent aux pixels tandis que les arêtes représentent leurs dépendances. Il est ainsi possible de déterminer la valeur d'un pixel seulement lorsqu'aucune dépendance ne s'applique sur lui (i.e. aucune arête du graphe n'y parvient).

Afin d'assurer qu'aucune itération ne bloque l'algorithme, on affecte la reconstruction de chaque ligne à une seule projection. Par exemple dans la figure 2.13, la projection suivant la direction $(1,1)$ est affectée à la reconstruction de la première ligne. L'attribution des projections aux lignes se fait de sorte que p_i décroît en même temps que l'index de la ligne augmente. Il est alors possible de déterminer un chemin dans ce graphe qui décrit les différentes itérations qui permettent de reconstruire entièrement l'image (plusieurs chemins peuvent exister). En particulier, chaque itération permet de déterminer la valeur d'un pixel, permettant en conséquence de réduire les dépendances appliquées sur les autres. Dans l'exemple de la figure 2.13, une itération possible consiste à :

1. débuter en reconstruisant le pixel supérieur gauche par la projection suivant la direction $(1,1)$ puisqu'aucune dépendance ne s'y applique ;
2. récupérer la valeur du pixel inférieur gauche depuis un bin de la projection suivant $(-1,1)$ pour la même raison ;
3. puisque les deux précédents pixels ont été reconstruits, plus aucune dépendance ne s'applique sur le premier pixel de la deuxième ligne, il est alors possible de récupérer sa valeur à partir de la projection suivant $(0,1)$.

Une fois cette première colonne reconstruite, il est possible de répéter l'itération sur les colonnes suivantes (jusqu'à reconstruire entièrement la grille). En pratique, ces

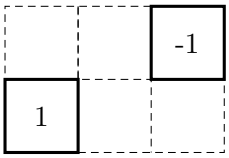


FIGURE 2.14 – Fantôme élémentaire selon la direction $(p, q) = (2, 1)$.

éléments correspondent à des zones mémoires contigües. En particulier, les bins des projections interviennent de manière séquentielle dans la reconstruction d'une ligne. Cette considération permet de favoriser la localité spatiale des données et donc les performances de l'implémentation.

Reconstruction à partir d'une représentation partielle

Lorsque l'ensemble de projections ne satisfait pas le critère de KATZ (équation (2.24)), le problème de reconstruction est sous-déterminé. En conséquence, la reconstruction par une représentation partielle conduit à plusieurs ou aucune solution. Comme nous l'avons vu précédemment avec la FRT, les fantômes correspondent aux éléments de l'espace nul. On définit un fantôme élémentaire ainsi :

$$g_{(p,q)} : p \mapsto \begin{cases} 1 & \text{si } p = (0,0) \\ -1 & \text{si } p = (p,q) \\ 0 & \text{sinon.} \end{cases} . \tag{2.25}$$

Sur la page suivante, la figure 2.14 représente le fantôme élémentaire selon la direction $(p, q) = (2, 1)$.

Soit un ensemble de projections suivant les directions de l'ensemble $\{(p_i, q_i)\}$, PHILIPPE [Phi98] a montré qu'il existe une unique décomposition de f telle que :

$$f = f^{\{(p_i,q_i)\}} + \sum_{j=1}^{r} a_i g_{(p_i,q_i)} , \tag{2.26}$$

où a_i correspondent aux inconnues.

2.3.3 Code à effacement Mojette

Nous avons vu que la transformation Mojette est capable de fournir une représentation redondante de l'image. Cette propriété essentielle forme la base de notre motivation pour concevoir un code à effacement à partir de la transformation Mojette. On rappelle ici qu'en théorie des codes, un code à effacement (n, k) transforme k blocs de données en un ensemble de n blocs encodés plus grand (i.e. $n \geq k$). Nous verrons dans la suite comment a été conçu le code à effacement Mojette sous sa forme non-systématique, puis nous verrons que le code obtenu n'est pas MDS.

Conception du code à effacement non-systématique

Contrairement à la FRT, il est nécessaire en Mojette de déterminer l'ensemble des projections que l'on souhaite calculer. Afin de concevoir simplement un code à effacement, on fixe l'un des paramètres (p, q) à 1. Considérons par la suite que pour chaque projection, $q_i = 1$ pour $i \in \mathbb{Z}_I$ comme cela à déjà été utilisé dans de précédents travaux [PNB01]. Soit une image discrète de taille $(P \times Q)$, avec Q le nombre de lignes de cette image, et soit un ensemble de I projections. Nous rappelons que la condition sur Q pour garantir l'unicité de la solution de reconstruction est $Q \leq \sum_{i=1}^{I} |q_i|$ (extrait de équation (2.24)). En conséquence, si l'on fixe $q_i = 1$, le critère de Katz est réduit au principe suivant : Q projections suffisent pour reconstruire l'information contenue dans une image de hauteur Q [Par01]. Dans ces conditions, Q projections constituent un ensemble suffisant pour reconstruire l'image. En particulier, la transformée Mojette correspond à un code à effacement (n, k) où k correspond à la hauteur de l'image Q, et n correspond au nombre de projections I. En conséquence, cette condition permet à la transformation Mojette de garantir une solution unique au problème de reconstruction lorsqu'un nombre maximal de $(n - k) = (I - Q)$ projections sont manquantes.

Un code à effacement "quasi MDS"

Il est important de considérer avec attention la taille des projections puisque cette taille a des conséquences sur la consommation mémoire. En particulier dans les applications de transmission et stockage, on souhaite réduire au maximum cette consommation. La taille B d'une projection, correspondant au nombre de bins qu'elle contient, est définie par Guédon et al. [GBB95] ainsi :

$$B(P, Q, p_i, q_i) = (Q - 1)|p_i| + (P - 1)|q_i| + 1 \ , \tag{2.27}$$

où (P, Q) correspond à la taille de l'image et (p_i, q_i) correspond à la direction de la projection étudiée. Dans le cas du code à effacement, où l'on fixe $q_i = 1$, l'équation (2.27) s'écrit :

$$B(P, Q, p_i, q_i) = (Q - 1)|p_i| + P \ . \tag{2.28}$$

En conséquence, la taille des projections augmente avec p_i. Rappelons que dans le cas d'un code MDS, la taille d'un bloc encodé correspond à la taille d'un bloc de données. Du point de vue des projections, la transformation Mojette permet de concevoir un code optimal puisqu'un ensemble de k projections suffisent pour reconstruire une image composée de k lignes. En revanche, du point de vue des bins, ce code n'est pas optimal puisque la taille des projections augmente avec la valeur de p_i. En conséquence, le code Mojette n'est pas MDS.

Choix de l'ensemble de projections Afin de réduire cet impact, la première étape consiste à déterminer la valeur des p_i permettant de générer l'ensemble des projections

le plus compact possible. Pour cela, il est nécessaire de choisir les valeurs de p_i dans l'ensemble des entiers issus de l'énumération des entiers alternativement positifs et négatifs[3] (i.e. $\{0, 1, -1, 2, \ldots\}$), comme proposé par Verbert et al. [VRG04].

Dans le cas d'un code MDS, la taille d'un bloc encodé correspond à la taille d'un bloc de donnée. Pour l'encodage Mojette d'une image rectangulaire, c'est le cas seulement pour la projection verticale ($p = 0, q = 1$) et horizontale ($p = 1, q = 0$). Dans le cas général, les projections sont plus grandes qu'une ligne de l'image. En conséquence, cette surpopulation du nombre de bins induit que le code à effacement Mojette n'est pas MDS. Parrein [Par01] définit ce coût de surpopulation, noté ϵ, comme étant le rapport du nombre de bins (défini dans équation (2.27)) sur le nombre de pixels :

$$\epsilon = \frac{\sum_{i=0}^{n-1} B(P, Q, p_i, q_i)}{P \times Q} \,. \tag{2.29}$$

En particulier, Parrein montre que cette valeur est significativement réduite à mesure que la largeur P de l'image augmente. Par conséquent, le code est MDS de façon asymptotique. On appelle ce genre de code sous-optimaux des codes $(1 + \epsilon)$-MDS tel que discuté par Parrein [Par01]. Dans le prochain chapitre de cette thèse, l'impact de l'encodage sera précisément évalué et comparé avec d'autres techniques (section 3.2).

Réduction du surcout de redondance Verbert et al. [VRG04] ont proposé une méthode basée sur l'analyse de l'algorithme de reconstruction afin de réduire la taille des projections. Prenons l'exemple d'une grille discrète ($P = 6, Q = 2$) sur laquelle on calcule trois projections suivant les directions $\{(1, 1), (0, 1), (-1, 1)\}$. Dans ce cas, puisqu'il est nécessaire d'avoir deux projections pour reconstruire la grille, il n'existe que 3 combinaisons possibles d'affection des projections aux lignes : $\{(1, 0), (0, -1), (1, -1)\}$. La première ligne ne peut alors être reconstruite que par les projections suivant ($p = 1$) ou ($p = 0$). De même, la seconde ligne ne peut être reconstruire que par les projections suivant ($p = 0$) ou ($p = -1$). En conséquence, quel que soit l'ensemble de projections utilisé lors de la reconstruction, le processus ne va utiliser que les $k = 6$ premiers bins de chaque projection. Ce qui signifie qu'il n'est pas nécessaire de calculer et stocker le septième et dernier bin des projections ($p = 1, -1$). Notons que dans ce cas particulier, le code Mojette est optimal puisque les projections font la même taille que les lignes de l'image.

Dans le cas général, Verbert et al. [VRG04] ont montré que cette réduction du nombre de bin ne peut amener le code à être optimal. Toutefois ils fournissent une méthode permettant de réduire la valeur d'ϵ. Cela permet deux choses : (i) l'opération d'encodage est plus performante puisque moins de calculs sont nécessaires ; (ii) la taille des projections est réduite ce qui améliore le transfert et le stockage des projections en pratique.

[3] séquence fournie par l'encyclopédie de Sloane et al. [Slo+96] `http://oeis.org/A001057`.

Conclusion du chapitre

La transformation de Radon pose les bases de la reconstruction tomographique. Son étude dans le domaine continu a ouvert la voie à de nombreuses techniques d'inversion. Nous avons vu que la tomographie discrète attaque le problème de la reconstruction tomographique par une représentation discrète des éléments et des algorithmes. Cette représentation a l'avantage de répondre au problème par une solution exacte par rapport aux solutions émanant du domaine continu, grâce à l'échantillonnage possible par la géométrie du problème.

Nous avons étudié la FRT et la transformation Mojette qui sont des versions discrètes et exactes de la transformation de Radon. La capacité d'inversion de ces transformations forme la base de la conception de nouveaux codes à effacement. Plus particulièrement nous avons étudié comment ces transformations peuvent être utilisées afin d'encoder et générer des informations redondantes, nécessaires pour tolérer l'effacement. Afin de déterminer la capacité des codes, nous avons énoncé les critères qui garantissent si un ensemble de projections donné permet de résoudre le problème de reconstruction. Les méthodes itératives et efficaces d'inversion ont été détaillées et permettent à ces codes de reconstruire l'information dans le cas où les données ont été altérées.

La transformation Mojette correspond à une version apériodique de la FRT, ce qui a pour conséquence de rendre le code à effacement associé non-MDS, contrairement au code fourni par la FRT. Toutefois ce surcout d'encodage est limité et la Mojette correspond à un code « quasi-MDS ». Ce léger surcout ouvre la voie à des algorithmes de reconstruction très efficaces. Actuellement disponible sous sa forme non-systématique, nous verrons qu'une mise en œuvre du code sous sa forme systématique peut améliorer les performances, et réduire la quantité de redondance. La conception de cette version systématique est l'objectif du chapitre suivant.

3

Code à effacement Mojette systématique

Sommaire

Introduction du chapitre

Le chapitre précédent a permis de définir les codes à effacement basés sur les transformations FRT et Mojette. Bien que le rendement de ce dernier soit sous-optimal (c'est à dire $(1+\epsilon)$-MDS), il a la particularité de disposer d'un algorithme de reconstruction itératif, de faible complexité [NKÉ06]. Puisqu'elle détermine le nombre d'opérations réalisées à l'encodage et au décodage, cette complexité joue alors un rôle important dans les latences du code, et donc, sur les latences du système de communication qui les utilise. Dans nos travaux de thèse, nous nous intéressons au cas des codes à effacement AL-FEC, utilisés par les couches « hautes » du modèle TCP/IP (i.e. transport et application), dont le rendement et la latence sont des critères essentiels. En effet, le rendement joue un rôle sur l'utilisation des ressources (e.g. bande passante sur un réseau, capacité d'un système de stockage). Quant à la latence, elle ne doit pas limiter les performances du système de communication, étant donné que l'encodeur et le décodeur sont des composants branchés en série dans la chaîne de traitement de l'information. Dans les systèmes de stockage distribués logiciels (*Software Defined Storage* ou SDS), le chemin de données (communément appelé *data path*) correspond à l'ensemble des composants par lesquels la donnée transite lors d'une communication. Dans ce contexte, le rendement et la latence ne doivent ni utiliser à outrance la capacité de stockage du système, ni limiter la latence des transferts de fichiers.

L'objectif de ce chapitre est de proposer une nouvelle conception du code à effacement Mojette sous sa forme systématique, qui permet en somme d'intégrer les symboles sources dans les mots de code générés à l'encodage. Cette construction sera présentée dans la section 3.1. Les sections 3.2 et 3.3 qui suivent, mettront respectivement en avant les deux principaux avantages de cette version, à savoir : (i) l'amélioration du rendement du code à effacement Mojette provoquée par la géométrie même de la transformation ; une évaluation de ce gain de rendement sera d'ailleurs présentée afin de positionner le rendement de notre code par rapport au rendement optimal des codes MDS ; (ii) la réduction du nombre d'opérations réalisées par le code.

3.1 Conception du code à effacement Mojette systématique

Cette section présente notre conception du code à effacement Mojette dans sa version systématique. La section 3.1.1 présentera dans un premier temps l'intégration des symboles sources dans le mot de code lors de l'opération d'encodage. Dans un second temps, nous proposerons un algorithme de reconstruction dans la section 3.1.2, afin de reconstituer les symboles sources en cas d'effacement. Nous verrons que l'algorithme d'inversion présenté ici correspond à une extension de l'algorithme inverse de Normand et al. [NKÉ06], qui a été étudié dans le chapitre précédent.

3.1.1 Construction du code systématique

Définissons tout d'abord la notion d'image dégradée. On appelle « image dégradée » (ou image partielle) f', une grille dans laquelle e lignes ont été effacées. Une mise en œuvre du code à effacement Mojette sous sa forme systématique précédemment a été proposée par David et al. [Dav+15]. Dans ce brevet, le procédé pour reconstruire une image f' repose sur les trois étapes suivantes :

1. calculer les valeurs des projections $M_{\{(p_i,q_i)\}}[f']$ de la grille partielle f'. Un ensemble suffisant de projections est défini par un ensemble $(p_i, q_i)_{i\in 1,\dots,e}$ de directions, nécessaire pour reconstruire une grille dont e lignes sont manquantes ;
2. calculer la différence $M_{\{(p_i,q_i)\}}[f] - M_{\{(p_i,q_i)\}}[f']$;
3. appliquer l'algorithme de reconstruction de Normand et al. [Nor+96] en utilisant les projections obtenues précédemment. Dans la suite, nous allons présenter une nouvelle mise en œuvre basée sur l'algorithme de Normand et al. [NKÉ06].

Ce que montre ce brevet, c'est que la construction de la version systématique du code à effacement Mojette est directe. Les n symboles du mot de code correspondent aux k lignes de la grille, auxquels on ajoute $(n-k)$ projections. Cette simplicité de conception en Mojette se distingue de la détermination d'une matrice d'encodage d'un code de Reed-Solomon systématique. Comme nous l'avions précisé dans le premier chapitre, cette détermination nécessite une élimination de Gauss-Jordan pour faire apparaître la partie identité de la matrice. Toutefois cette technique ne s'est limitée qu'au brevet. Dans la suite nous proposons une nouvelle méthode moins coûteuse en capacité et en accès mémoire.

3.1.2 Algorithme de reconstruction

L'algorithme de reconstruction présenté ici repose sur deux principales modifications de l'algorithme de [NKÉ06]. Ces modifications correspondent à : (i) une nouvelle détermination des décalages (i.e. appelés *offsets* dans [NKÉ06]), prenant en compte les lignes de la grille (i.e. les symboles sources) déjà présentes dans la grille. Cette modification sera le sujet de la section 3.1.2 ; (ii) une nouvelle méthode de calcul de la valeur d'un pixel à reconstruire, prenant en compte les pixels déjà présents sur la droite de projection. Cette méthode sera introduite dans la section 3.1.2.

L'algorithme 1 de reconstruction que nous avons conçu est présenté en page 74. Dans un soucis de cohérence avec la description de l'algorithme de Normand et al. [NKÉ06], nous utiliserons le terme « offset » pour parler des décalages nécessaires à la détermination du parcours de reconstruction dans la grille. Pour les mêmes raisons, une projection $M_{\{(p,q)\}}[]$ est désignée par $\text{Proj}_f(p,q)$.

Détermination des *offsets* pour la reconstruction

De manière comparable à ce qui a été réalisé dans l'algorithme de Normand et al. [NKÉ06], il est nécessaire de déterminer la valeur des *offsets*. Ces *offsets* désignent les

Algoritme 1 Algorithme d'inversion systématique, modification de [NKÉ06]

Entrée : Proj$(P_i, 1)$ triées par p_i croissants avec $i \in \mathbb{Z}_I$
Entrée : Eff(i) triés par ordre décroissant, avec $i \in \mathbb{Z}_e$

▷ Calcule $S-$, $S+$ et S
▷ équations (3.1) et (3.2)
$S_{\text{minus}} \leftarrow S_{\text{plus}} \leftarrow 0$
pour $i = 0$ à $Q - 2$ **faire**
    $S_{\text{minus}} \leftarrow S_{\text{minus}} + \max(0, -p_i)$
    $S_{\text{plus}} \leftarrow S_{\text{plus}} + \max(0, p_i)$
fin pour
$S \leftarrow S_{\text{minus}} - S_{\text{plus}}$
▷ Calcule la valeur de l'offset de la dernière ligne à reconstruire
▷ équation (3.3)
$\text{offset}(e-1) \leftarrow \max(\max(0, -p_r) + S_{\text{minus}}, \max(0, p_r) + S_{\text{plus}})$
▷ Calcule la valeur de l'offset des autres
pour $i \leftarrow (e-2)$ à 0 **faire**
    $\text{offset}(i) \leftarrow \text{offset}(i+1) + p_{i+1}$
    ▷ Mise à jour des offsets des lignes inférieures
    pour $j \leftarrow (i+1)$ à $e-1$ **faire**
        $\text{offset}(j) \leftarrow \text{offset}(j) - ((\text{Eff}(i+1) - \text{Eff}(i) - 1) \times p_{i+1})$
    fin pour
fin pour
▷ Détermination du plus petit offset
$\text{offset}_{\text{min}} = \text{offset}(0)$
pour $i \leftarrow 1$ à $e-1$ **faire**
    si $\text{offset}_{\text{min}} > \text{offset}(i)$ **alors**
        $\text{offset}_{\text{min}} = \text{offset}(i)$
    fin si
fin pour
pour $k \leftarrow -\max(\text{offset}(0), \text{offset}(e))$ à $(P - \text{offset}_{\text{min}} - 1)$ **faire**
    pour $l \leftarrow 0$ à $\text{offset}(e-1)$ **faire**
        $y = \text{Eff}(l)$
        ▷ Détermination de la valeur du pixel
        ▷ équation (3.4)
        $f(k,l) \leftarrow \text{Proj}_f(p_i, q_i) - \text{Proj}_{f'}(p_i, q_i)$
    fin pour
fin pour

décalages attribués à chaque ligne à reconstruire, dans le but de définir l'ordre des pixels à reconstruire. Cet ordre correspond donc au chemin de reconstruction de notre algorithme. L'ordre de reconstruction des pixels est primordial pour que l'algorithme de reconstruction ne soit pas bloqué. Plus particulièrement, cet ordre permet à chaque itération, de ne considérer que les pixels du graphe sur lesquels ne s'applique aucune dépendance.

Dans la version non-systématique, toutes les lignes de la grille doivent être reconstruites. En conséquence, pour déterminer la valeur de l'offset d'une ligne, il est nécessaire de connaitre son index, ainsi que la direction de la projection utilisée pour la reconstruire. Dans la version systématique, il est par ailleurs nécessaire de prendre en compte les lignes déjà présentes dans le calcul des *offsets* des lignes à reconstruire. On considère dans la suite l'ensemble des index $\mathrm{Eff}(i)$ des lignes effacées, triées par ordre décroissant, avec $i \in \mathbb{Z}_e$, tel que e désigne le nombre de lignes effacées. La détermination des valeurs des *offsets* se fera de la dernière ligne à reconstruire, jusqu'à la première. Il est ainsi nécessaire de calculer au préalable l'offset de la dernière ligne à reconstruire $\mathrm{Offset}(\mathrm{Eff}(e-1))$. Pour cela, nous définissons S_{minus} et S_{plus}, qui permettent de déterminer la valeur de $\mathrm{Offset}(\mathrm{Eff}(e-1))$, tel que :

$$S_{\mathrm{minus}} = \sum_{I=1}^{Q-2} \max(0, -p_i), \tag{3.1}$$

$$S_{\mathrm{plus}} = \sum_{I=1}^{Q-2} \max(0, p_i), \tag{3.2}$$

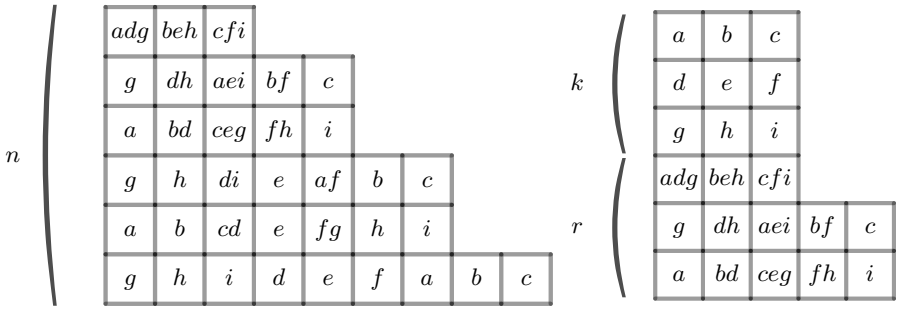
$$\mathrm{Offset}(\mathrm{Eff}(e-1)) = \max(\max(0, -p_r) + S_{\mathrm{minus}}, \max(0, p_r) + S_{\mathrm{plus}}). \tag{3.3}$$

La méthode pour déterminer la valeur de l'offset des autres lignes est décrite entre les lignes 9 et 19 de l'algorithme 1 de reconstruction (cf. page 74).

Calcul de la valeur du pixel à reconstruire

Comme nous l'avons vu dans le chapitre précédent, en non-systématique, lorsqu'aucune dépendance ne s'applique sur un pixel que l'on souhaite reconstruire, la valeur du pixel est directement lue dans le bin associé $\mathrm{Proj}_f(p_i, q_i, b)$. Dans la version systématique en revanche, lors de la reconstruction, les valeurs des pixels ne dépendent plus seulement des valeurs de bins, mais elles peuvent également dépendre des valeurs des pixels déjà présents dans la grille. Par définition de la transformation Mojette, un pixel participe à la valeur de $\mathrm{Proj}_f(p_i, q_i, b)$, comme élément de la somme des pixels situés sur la droite d'équation $b = -kq_i + lp_i$. Pour reconstruire sa valeur, on fait la différence de la valeur du bin de la projection de f avec le bin correspondant de la projection de f'. Plus précisément, cette opération est définie comme suit :

$$f(k, l) = \mathrm{Proj}_f(p_i, 1, k - lp_i) + \mathrm{Proj}_{f'}(p_i, 1, k - lp_i). \tag{3.4}$$



(a) Non-systématique (b) Systématique

FIGURE 3.1 – Comparaison entre l'encodage Mojette non-sytématique et systématique. L'ensemble $\{a, \ldots, i\}$ correspond aux valeurs des pixels de l'image 3×3 (i.e. de hauteur $k = 3$) qui ont permis de calculer les $n = 6$ blocs encodés de la version non-systématique.

où $Proj_{f'}(p_i, 1, k - lp_i)$ correspond à la somme des valeurs des pixels de l'image en reconstruction, selon la droite passant par le pixel de coordonnées (k, l), d'équation $b = -kq_i + lp_i$. Cette équation est utilisée pour reconstruire l'ensemble des pixels effacés de l'image partielle f', tel que décrit dans les instructions entre les lignes 27 et 34 de l'algorithme 1.

3.2 Évaluation du gain dans le rendement du code

Un code MDS génère la quantité minimale de redondance pour une tolérance aux pannes donnée. Dans le chapitre précédent, nous avons vu que le code à effacement Mojette n'est pas optimal et est considéré comme $(1 + \epsilon)$ MDS [Par01]. Cette désignation met en exergue deux types de redondance dans le code à effacement Mojette : (i) le premier type de redondance correspond à la définition du rendement du code $\frac{k}{n}$. En ce sens, le code Mojette définit un nombre de symboles optimal pour une capacité de correction donnée ; (ii) le second type de redondance provient de la géométrie même de la transformation. Nous avons vu en effet dans le chapitre précédent que la taille des projections augmente à mesure que la valeur de $|p_i|$ s'accroit. Rappelons à présent que la valeur d'ϵ désigne le facteur de la redondance présente à l'intérieur de k symboles (i.e. un ensemble suffisant pour reconstruire) [Par01]. Ce critère dépend ainsi du choix des k symboles étudiés parmi les n générés.

Afin de mieux correspondre au contexte du stockage, nous choisissons plutôt de nous intéresser à la redondance totale générée par le code, afin d'évaluer la taille totale occupée par les données encodées dans le cas de ce contexte. Pour cela, nous définissons μ comme le rapport entre le nombre d'éléments de symboles encodés (i.e. bins Mojette), et le nombre d'éléments sources (i.e. pixels de la grille). Dans la suite nous évaluerons dans un premier temps le gain de redondance induit par la version systématique du code Mojette, par rapport à sa version non-systématique. Une seconde étude permettra de positionner le coût de la redondance du code Mojette par rapport aux coûts qui résultent

		1024	2048	4096	8192	16384	32768	65536	131072
(3,2)	nsys	1,51	1,51	*1,50*	*1,50*	*1,50*	*1,50*	*1,50*	*1,50*
	sys	*1,50*	*1,50*	*1,50*	*1,50*	*1,50*	*1,50*	*1,50*	*1,50*
(6,4)	nsys	1,71	1,61	1,55	1,53	1,51	1,51	*1,50*	*1,50*
	sys	1,52	1,51	*1,50*	*1,50*	*1,50*	*1,50*	*1,50*	*1,50*
(12,8)	nsys	3,47	2,48	1,99	1,75	1,62	1,56	1,53	1,52
	sys	1,72	1,61	1,55	1,53	1,51	*1,50*	*1,50*	*1,50*

TABLE 3.1 – Résultats arrondis des coûts μ des données encodées pour les versions non-systématique (nsys) et systématique (sys) du code Mojette en fonction de différents paramètres de code (n, k) et de différentes tailles de $\mathcal{M}$ en octets. Les résultats en italique représentent la valeur optimale obtenue par un code MDS (i.e. 1.50).

des techniques de réplication et des codes MDS.

3.2.1 Réduction de la redondance en systématique

La figure 3.1 (page 76 illustre le gain de la version systématique d'un code Mojette $(6, 3)$ par rapport à la version non-systématique. L'objectif de cette section est d'analyser ce gain. Nous avons vu précédemment que la taille des projections varie en fonction des paramètres de la grille discrète P et Q, ainsi que des paramètres de l'ensemble des directions de projection $\{(p_i, q_i)\}$. Nous rappelons ici la formule permettant de déterminer la taille d'une projection :

$$B(P, Q, p, q) = |p_i|(Q - 1) + |q_i|(P - 1) + 1. \tag{3.5}$$

Puisque dans le cas des codes à effacement Mojette, la taille des blocs encodés (i.e. les projections) varie, nous allons étudier le nombre d'éléments de projection par rapport au nombre de pixels. Dans le cas du code à effacement non-systématique, la valeur de μ correspond au quotient de la somme du nombre de bins B de chaque projection de l'ensemble $\{(p_i, q_i)\}$, sur le nombre d'éléments de la grille :

$$\mu = \frac{\sum_{i=0}^{n-1} B(P, Q, p_i, q_i)}{P \times Q}. \tag{3.6}$$

Par rapport à la forme non-systématique, k projections sont remplacées par les k lignes de la grille discrète quand le code est systématique. En conséquence, la valeur de μ correspond au quotient de la somme du nombre de pixels et de bins produits, sur le nombre de pixels de l'image :

$$\mu = \frac{P \times Q + \sum_{i=0}^{n-k-1} B(P, Q, p_i, q_i)}{P \times Q}. \tag{3.7}$$

Puisque la taille d'une projection ne peut être inférieure à la longueur d'une ligne de la grille (i.e. $Q \leq B(P, Q, p_i, q_i)$), le coût μ des données encodées est moindre en systématique qu'en non-systématique. Dans la suite de notre évaluation, nous considérons un ensemble de projections de telle sorte que $q_i = 1$ pour $i \in \mathbb{Z}_Q$, on peut alors écrire l'équation (3.5) ainsi :

$$B(P, Q, p_i, 1) = (Q - 1)|p_i| + P. \tag{3.8}$$

La valeur de μ dépend naturellement de l'ensemble de projections choisi. En particulier, pour une taille de grille fixée, la valeur du paramètre p de direction de projection influence la valeur de μ. Afin de réduire cette valeur, nous choisirons alternativement des entiers positifs puis négatifs, dont la valeur croît à partir de zéro, comme valeurs de p_i. Par exemple, pour le code sous sa forme systématique, nous considérerons les ensembles de projection $S_{\left(\frac{n}{k}\right)} = \{(p_i, q_i)\}$ suivants :

1. $S_{\left(\frac{3}{2}\right)} = \{(0, 1)\}$,
2. $S_{\left(\frac{6}{4}\right)} = \{(0, 1), (1, 1)\}$,
3. $S_{\left(\frac{9}{6}\right)} = \{(0, 1), (1, 1), (-1, 1)\}$,
4. $S_{\left(\frac{12}{8}\right)} = \{(0, 1), (1, 1), (-1, 1), (2, 1)\}$.

Ces ensembles partagent le même taux de codage $r = \frac{3}{2}$ et fournissent respectivement une tolérance face à une, deux, trois et quatre pannes.

Le tableau 3.1 de la page 77 compare les résultats des coûts μ (à l'arrondi près) pour les deux versions du code à effacement Mojette avec les ensembles de projections proposés précédemment. Pour obtenir ces résultats, on a utilisé une taille de pixel de 64 bits. En conséquence, la valeur de P qui correspond à la largeur de la grille peut être obtenue ainsi :

$$P = \frac{\mathcal{M} \times 8}{k \times 64}, \tag{3.9}$$

avec $\mathcal{M}$ qui correspond à la taille des données traitées en octets. Ces résultats permettent d'observer que la valeur de μ croît lorsque les paramètres (n, k) du code augmentent. Plus précisément, en $(12, 8)$, la version non-systématique possède un coût élevé de $\mu = 3,47$, contre $1,72$ en systématique. Dans ce cas, $P = 16$, ce qui correspond à $(2 \times Q)$. Or, cette faible différence entraîne de grandes valeurs dans l'équation (3.8). Plus la valeur de P augmente, plus la valeur de μ diminue. C'est ce que l'on observe dans le tableau, où les valeurs de μ convergent vers la valeur optimale $\mu = 1,50$ qui correspond à la valeur atteinte par un code MDS. En conséquence, on tâchera de considérer des tailles de blocs $\mathcal{M}$ suffisamment grandes (dans la mesure du possible), afin de réduire au mieux la redondance contenue dans les symboles du mot de code.

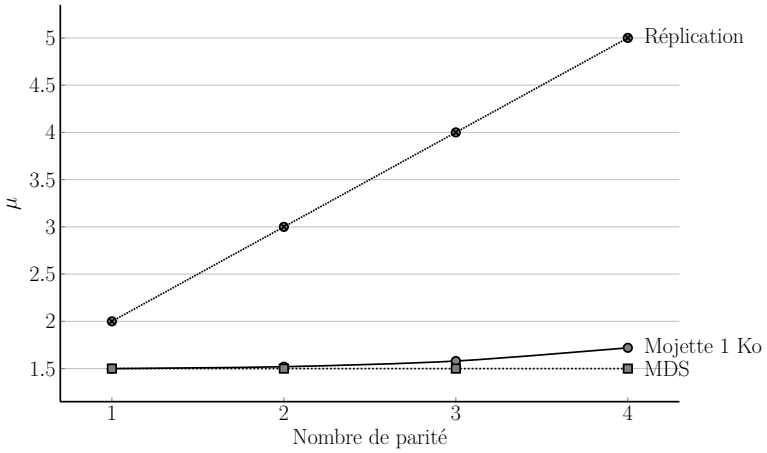


FIGURE 3.2 – Calcul de la valeur de μ pour différentes techniques de redondance en fonction de différents paramètres de code (n, k). Le taux de codage est fixé à $\frac{3}{2}$ tel que ces paramètres valent respectivement $(3, 2)$, $(6, 4)$, $(9, 6)$ et $(12, 8)$. Les codes issus de ces paramètres sont capables de supporter de une à quatre pannes respectivement. La valeur illustrée pour le code à effacement Mojette correspond à une taille de bloc de données de $\mathcal{M} = 1$ Ko.

3.2.2 Coût de la redondance par rapport à d'autres codes

Dans notre évaluation, nous allons considérer trois techniques qui permettent de générer de la redondance : la réplication, le code à effacement MDS, et le code à effacement Mojette dans sa version systématique. La figure 3.2 présente notre évaluation.

Dans le cas de la réplication, le facteur de redondance μ correspond au nombre de copies générées. Par exemple, dans le cas où l'on souhaite protéger la donnée face à deux pannes, il est nécessaire de générer deux copies en plus de l'information initiale. Dans cet exemple, le facteur de redondance μ vaut 3.

Dans le cas des codes à effacement, nous fixons le taux de codage de $r = \frac{3}{2}$ afin comparer la valeur de μ équitablement. Nous comparons ces techniques pour plusieurs paramètres de code (n, k) définis dans l'ensemble $\{(3, 2), (6, 4), (9, 6), (12, 8)\}$. Pour les codes MDS, la valeur du facteur de redondance μ correspond au taux de codage. En effet r correspond à la quantité de donnée en sortie n sur la quantité de donnée en entrée k. C'est pourquoi, si l'on fixe un taux de codage r, la quantité de redondance produite reste la même, indépendamment de la tolérance aux pannes du code. En conséquence dans la figure 3.2, la valeur de μ correspond à $r = \frac{3}{2} = 1,5$ quelle que soit la tolérance aux pannes fixée.

Bien que le code Mojette ne soit pas MDS, nous avons vu dans le tableau 3.1 que pour une taille de bloc de 4 Ko, le code Mojette systématique n'entraîne un surcout de seulement 3% (puiqsue $\mu = 1,55$). Ce qui le rend dans ce cas, quasi-MDS.

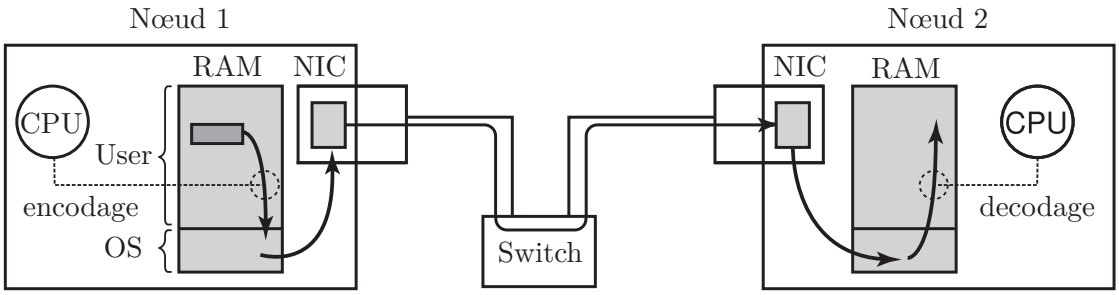


FIGURE 3.3 – Représentation du chemin de données dans la transmission entre deux terminaux. La donnée est présente dans la mémoire principale (RAM) du nœud 1. L'encodage est réalisé par le CPU de ce nœud avant de transmettre l'information de l'interface réseau (NIC). Cette interface transmet ensuite l'information encodée sur le canal à effacement. Après réception des données par le nœud 2, une opération de décodage est réalisée avant de restituer la donnée reconstruite à la mémoire. Cette figure est inspirée de [TB14].

3.3 Considérations sur la réduction du nombre d'opérations

Afin de ne pas former de goulot d'étranglement dans la chaine de transmission du système de communication dans lequel il est utilisé, un code doit fournir de bonnes latences en encodage et en décodage. La section 3.3.1 permet de comprendre la position du code dans la chaîne de traitement. Nous introduirons des rapports entre les différents maillons de cette chaîne et expliquerons quel doit être l'ordre de grandeur des performances du code pour ne pas former un goulot d'étranglement. Les deux analyses suivantes permettent de comprendre en quoi une version systématique du code peut améliorer ses performances. En particulier, la section 3.3.2 confirme que le gain en encodage est significatif. La section 3.3.3 montre quant à elle, que les performances en décodage dépendent du schéma de perte.

3.3.1 Contraintes en performances des codes à effacement

Pour comprendre l'enjeu des performances des codes à effacement, nous analyserons dans un premier temps son positionnement dans la chaîne de traitement des données. Nous verrons alors que le code doit être suffisamment performant pour ne pas former un goulot d'étranglement dans cette chaîne. Par la suite, nous déterminerons un ordre de grandeur des performances que notre code doit fournir.

Positionnement du code à effacement dans la transmission de l'information

Dans le contexte des télécommunications, les applications sont intrinsèquement liées au matériel qui traite et transporte la donnée, ainsi qu'aux techniques de codage qui permettent aux informations de transiter à travers un canal à effacement. La figure 3.3

Description	Temps d'accès	Temps Normalisé
1 cycle CPU	0,3 ns	1 s
Accès cache niveau 1	0,9 ns	3 s
Accès cache niveau 2	2,8 ns	9 s
Accès cache niveau 3	12,9 ns	43 s
Accès RAM	120 ns	6 min
E/S disque SSD	50-150 µs	2-6 jours
E/S disque dur	1-10 ms	1-12 mois
Internet : SF à NYC	40 ms	4 ans
Internet : SF à GB	81 ms	8 ans
Internet : SF à Australie	183 ms	19 ans
Redémarrage d'un SE virtuel	4 s	423 ans
Redémarrage d'une machine virtuelle	40 s	4000 ans
Redémarrage du système	5 m	32 millénaires

TABLE 3.2 – Comparaison des temps d'accès réels pour différentes opérations informatiques. La troisième colonne normalise ces temps sur la base d'un cycle CPU pour une seconde. Extrait de [Gre13].

représente une vue générale de la chaîne de transmission entre deux terminaux interconnectés. La donnée issue de la RAM du nœud 1 est traitée par le CPU afin de la transmettre sur le média de communication à partir de l'interface réseau. Sur le réseau, l'information passe au travers de composants gérant l'acheminement des données. Ce média représente un canal à effacement dans lequel les paquets peuvent être perdus. Une fois parvenue au destinataire, une opération inverse à la première étape est réalisée afin de stocker cette donnée dans la RAM du nœud 2. L'objectif du code à effacement est de traiter les données avant leur transmission ou leur stockage. En conséquence, l'encodage s'opère entre l'ensemble des composants de traitement (processeur, mémoire centrale), et l'ensemble des composants de communication (stockage de masse, réseau).

Dans notre cas, il est nécessaire de concevoir un code à effacement qui ne forme pas un goulot d'étranglement dans cette chaîne afin que ce soit le matériel qui limite les performances du système. Dans les systèmes distribués par exemple, on partage les ressources de différentes unités en concevant des algorithmes distribués pour exploiter au maximum la capacité du matériel.

Le tableau 3.2 donne une comparaison des délais observés pour certaines opérations informatiques. Afin de bien visualiser le rapport de différence entre ces opérations, la troisième colonne normalise ce délai sur la base d'un cycle CPU pour une seconde. S'il est nécessaire d'attendre une seconde pour un cycle CPU, le temps de récupération d'un bloc d'information sur le disque peut durer jusqu'à un an. En conséquence, la mise en œuvre d'un code à effacement doit favoriser les opérations proches du processeur. On travaillera ainsi à favoriser le stockage des données et des instructions dans les différents niveaux de cache, et éviter les interactions avec le disque.

Naturellement, il est impossible de garantir cette situation. Par exemple, lorsque la donnée n'est pas disponible dans le cache CPU, il est nécessaire de la faire "remonter" de la RAM. Si la taille des données ne peut entrer dans la RAM, il faudra nécessairement des interactions avec les disques de masse. Pour aller plus loin, si la donnée n'est pas disponible sur le nœud qui la demande, il faut la faire transiter par le réseau. En conséquence, il y a une forte dépendance entre le processeur, la RAM, le support de masse et le réseau.

Performance des codes

Matériel	Débits (Mo/s)
RAM (DDR3)	10000
Réseau (Fast Ethernet)	1000
Disque (SSD)	500

TABLE 3.3 – Ordre de grandeur de débits

Le tableau 3.3 donne l'ordre de grandeur des débits atteints par la RAM, les interfaces réseaux, et les disques. En particulier on observe que les disques et le réseau forment les éléments limitants. Afin d'améliorer les performances d'un système composé de ces différents éléments, il existe deux possibilités : (i) attendre puis acheter au prix fort la nouvelle génération de matériel ; (ii) agréger plusieurs composants ensemble afin de partager leurs ressources. Bien que la seconde option apporte une complexité de mise en œuvre, son prix est nettement plus accessible. En ce qui concerne le stockage de masse, cette agrégation a vu le jour avec la conception des différents niveaux de techniques RAID par PATTERSON et al. [PGK88]. Par exemple, l'utilisation du RAID-0 permet généralement d'augmenter les débits d'un facteur 2. Côté réseau, des techniques d'agrégation des liens réseaux existent, comme proposé par ADISESHU et al. [APV96]. Il est alors possible d'améliorer les performances en exploitant plusieurs interfaces et liens réseaux.

La présence du code au sein de la chaîne de traitement entraîne un calcul intermédiaire entre le processeur et les médias de communication. Afin de ne pas former un goulot d'étranglement dans cette chaîne, le code doit être suffisamment performant pour supporter les débits montants des disques et/ou du réseau. Une évaluation des performances des codes à effacement utilisés dans les applications de stockage a été réalisée par PLANK et al. [Pla+09]. Dans cette étude, les auteurs montrent que les débits d'encodage et de décodage observés par les meilleurs codes sont de l'ordre d'un gigaoctet par seconde. Par conséquent, il est possible que ces codes forment un goulot d'étranglement dans le cas où un agrégat de disques ou de liens réseaux sature le nœud.

Il est alors essentiel qu'un code puisse fournir de bonnes performances. C'est pourquoi ce chapitre s'intéresse à l'optimisation des performances du code à effacement basé sur la transformation Mojette. Nous verrons dans les deux parties qui suivent, les bénéfices d'une version systématique sur l'encodage et le décodage.

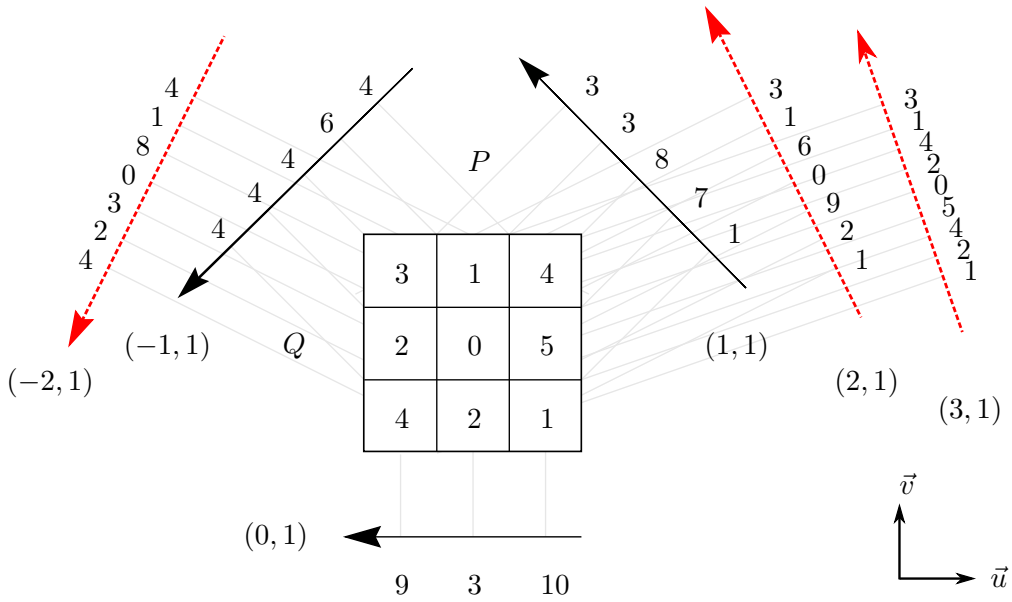


FIGURE 3.4 – Comparaison du nombre de projections calculées entre la forme systématique et non-systématique pour un code Mojette $(6, 3)$. En particulier, les projections suivant les directions de l'ensemble $\{(p_i \geq 2, 1)\}$ (en pointillés rouge) correspondent aux projections supplémentaires qu'il est nécessaire de calculer sous la forme non-systématique.

3.3.2 Bénéfices de cette nouvelle technique sur l'encodage

Par rapport à la version non-systématique, cette nouvelle technique permet de réduire significativement le complexité à l'encodage. En version non-systématique, il est nécessaire de calculer n projections à partir d'une grille discrète constituée de k. Dans cette nouvelle version systématique, nous considérons les k lignes de cette grille comme faisant partie des données encodées. En conséquence, il suffit de calculer $(n-k)$ projections pour fournir la même protection qu'avec l'approche classique du code à effacement Mojette (n, k). D'une manière générale, le rapport g de blocs de parité générés entre les deux versions s'exprime ainsi :

$$g = \frac{n}{n-k} \tag{3.10}$$

Prenons l'exemple d'un code avec un taux $r = 2$, comme un code $(6, 3)$ fournissant de la protection face à trois effacements. La figure 3.4 représente la comparaison entre les deux techniques pour cet exemple. En version systématique, l'ensemble des données encodées correspond aux k lignes de la grille, auxquelles on ajoute $r = 3$ projections calculées. Dans notre exemple, ces projections sont construites suivant les directions $\{(p_i, q_i)\} = \{(-1, 1), (0, 1), (1, 1)\}$. Sous sa forme non-systématique, le code à effacement Mojette doit calculer trois projections supplémentaires afin de fournir la même disponibilité des données. Sur la figure 3.4, ces projections supplémentaires sont représentées en rouge. Cette nouvelle version systématique nécessite de calculer deux fois moins de projections

dans cet exemple. Cette réduction du nombre d'opérations dépend des paramètres du code. Si l'on prend le cas d'un code $(6, 4)$, l'encodage génère trois fois moins de projections. Nous verrons dans la suite l'impact sur le décodage.

3.3.3 Bénéfice de cette technique sur le décodage

Dans cette partie nous allons étudier le comportement du code systématique en fonction du schéma de perte. On distingue trois schémas de pertes : (i) le cas optimal correspond à la situation où la grille n'a subi aucun effacement ; (ii) le cas où la grille est dégradée (i.e. elle subit un nombre e d'effacements, où $e < k$) ; (iii) le pire cas où toute la grille est effacée.

Accès direct sans dégradation

Lorsqu'aucune ligne de la grille n'est effacée, il s'agit du meilleur cas. C'est dans cette situation que réside le principal avantage de cette technique puisqu'il n'est pas nécessaire d'exécuter d'opération de décodage. Si aucune des k lignes de données ne subit d'effacement, la donnée est immédiatement accessible en clair. En conséquence aucun calcul n'est réalisé et les performances sont optimales.

En comparaison, dans le cas non-systématique, il est toujours nécessaire de reconstruire la grille entière, même lorsqu'aucun effacement ne survient. Quel que soit le schéma de perte, le décodage met en jeu k projections pour reconstruire les k lignes. En conséquence, le décodage nécessite toujours un travail calculatoire dont le coût est significatif. Dans la suite, nous analysons le cas où des effacements se produisent.

Dégradation partielle des données

Une dégradation des données entraîne nécessairement une opération de décodage afin de restaurer la donnée perdue. Nous considérons à présent que le nombre de lignes de grille discrète effacées e est inférieur à k. Dans ce cas, l'opération de décodage est possible dès lors que l'on accède à un ensemble suffisant de e projections pour reconstruire les e lignes effacées. Plus précisément, ce problème correspond à reconstruire une grille partiellement remplie. valeur %des bins des projections, mais également la valeur des pixels déjà présents %dans la grille. Nous verrons en détail ce nouvel algorithme dans la prochaine %partie.

En comparaison avec la version non-systématique, cette nouvelle mise en œuvre est plus performante. En effet, quelque soit le schéma de perte en non-systématique, il est nécessaire d'utiliser k projections pour reconstruire l'ensemble de la grille tout entière. Toutefois, cette nouvelle technique correspond à la reconstruction d'une grille partiellement reconstruite. En conséquence, l'ensemble des pixels à reconstruire est moins important que dans le cas non-systématique et donc, moins d'opérations sont nécessaires pour le décodage.

Perte complète des données

Dans le cas où $(e = k)$, l'ensemble des lignes de la grille est effacé. Il est alors nécessaire de décoder l'information à partir de k projections. L'opération de décodage est alors identique, que le code soit systématique ou non.

Bilan de l'impact en décodage

L'avantage principal de la version systématique est de fournir des performances optimales quand la grille ne subit aucune dégradation. Il n'y a pas besoin dans ce cas de décoder l'information, qui est disponible en clair dans la grille. Lorsque des effacements apparaissent, les performances décroissent puisqu'il est nécessaire de déclencher l'opération de décodage. Ces performances se dégradent alors avec le nombre de lignes effacées, et le pire cas est obtenu quand toute la grille est effacée. Dans cette situation, l'opération de décodage correspond à l'opération effectuée en non-systématique, et les performances sont en conséquence semblables pour les deux techniques. Rappelons que la forme non-systématique fournit les mêmes performances quel que soit le schéma de perte puisqu'il est nécessaire de reconstruire la grille entière à partir de k projections. En comparaison, cette situation correspond au pire scénario de perte dans le cas systématique. Dans tous les autres cas, les performances en systématique sont meilleures.

Conclusion du chapitre

Dans ce chapitre, nous avons présenté un moyen permettant d'améliorer le code à effacement Mojette selon deux critères parmi la liste élaborée dans le premier chapitre, à savoir : (i) le rendement ; (ii) le nombre d'opérations nécessaires pour l'encodage et le décodage. Cette méthode consiste à utiliser le code à effacement Mojette sous sa forme systématique. Une construction de cette version a été proposée dans la section 3.1. Cette construction (simple à mettre en œuvre) consiste à considérer que les n symboles d'un mot de code, sont composés des k lignes de la grille, et de $n-k$ projections Mojette. Pour accompagner cette proposition de conception, un algorithme d'inversion a été proposé afin de reconstruire la grille dans le cas systématique. Par la suite, nous avons analysé les conséquences de cette construction, et avons déterminé deux améliorations.

La première amélioration concerne le rendement du code Mojette. La section 3.2 a permis de montrer que le surcout de redondance de ce code provient de la géométrie de la transformation. Plus précisément, les projections sont de taille variable, et cette taille augmente avec l'index de la projection. La version systématique permet de réduire la quantité de redondance générée par le code en diminuant la quantité de projections à générer. Bien que le rendement converge vers l'optimal (au sens MDS) à mesure que la largeur de la grille augmente, nous avons pu observer une réduction de la quantité de données encodées par deux dans le cas extrême de notre évaluation. Ce cas met en jeu une grille de faible largeur (i.e. $P = 16, Q = 8$). Dans la pratique, nous avons vu qu'un code $(12, 8)$ qui opère sur des blocs de 4 Ko nécessite un surcout que de 3%, ce qui correspond à un code quasi-MDS.

La seconde amélioration, traitée dans la section 3.3 s'intéresse à l'amélioration du critère lié au nombre d'opérations nécessaires pour l'encodage et le décodage. Pour les mêmes raisons que pour le rendement (i.e. moins de projections à calculer), le nombre d'opérations nécessaires en encodage est réduit. Par exemple, dans le cas d'un code de rendement $\frac{3}{2}$, il y a trois fois moins de projections à calculer. Le décodage nécessite également moins d'opérations, dans le cas où certaines lignes de la grille sont disponibles. En particulier, cette amélioration permet un accès immédiat aux données lorsqu'aucun des symboles sources (i.e. les lignes de la grille) n'est effacé.

Conclusion de la partie

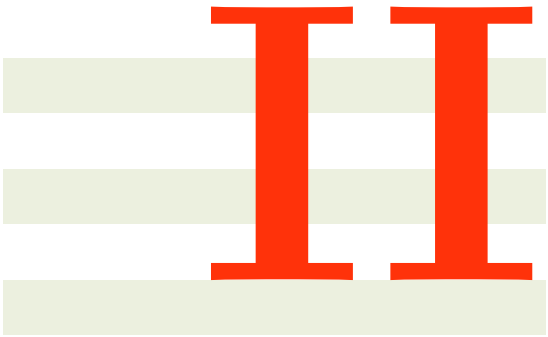
Le lien entre la théorie des codes et les transformées discrètes a été établi dans cette partie. Nous avons vu dans le chapitre 1, les notions de la théorie des codes nécessaires à la compréhension des codes à effacement, ainsi qu'à l'élaboration d'une liste de critères de comparaison. Cette étude, qui a conduit à l'analyse des codes MDS de Reed-Solomon, et des codes LDPC, a montré qu'aucun des codes étudiés n'est capable de satisfaire l'ensemble des critères proposés.

Le chapitre 2 propose une nouvelle approche à base de transformées discrètes, dans l'objectif de concevoir de nouveaux codes. Nous y avons notamment introduit les codes à base de FRT et de transformation Mojette. Nous avons tout d'abord vu que la FRT peut fournir un code à effacement MDS. L'étude qui a suivi a permis de montrer que malgré son rendement sous-optimal, le code à effacement Mojette possède l'avantage d'avoir un algorithme de décodage itératif efficace. Notre motivation a donc été d'utiliser ce dernier en tâchant d'améliorer son rendement.

Dans le chapitre 3, la construction d'une version systématique, et l'élaboration d'un algorithme de décodage adapté, ont permis d'améliorer le rendement et les performances de notre code. Jusque là, le code à effacement Mojette satisfait les critères suivants :

1. une complexité théorique faible grâce à l'algorithme de décodage itératif ;
2. une complexité des opérations faibles, correspondant à des additions ;
3. l'indépendance des paramètres du code (la hauteur et le nombre de projections peuvent être choisis arbitrairement) ;
4. une forme systématique possible, et dont la mise en œuvre est simple.

L'efficacité des opérations d'encodage et de décodage du code à effacement Mojette se fait au détriment d'un faible surcout de redondance nécessaire (désigné par μ dans notre étude). Toutefois, nous avons vu que ce surcout par rapport aux code MDS, se limite à 3% en pratique. Le code Mojette est donc le code approprié pour concevoir un système de stockage capable de gérer à la fois, les données froides, et les données chaudes. Restent alors à déterminer les performances du code dans le contexte du stockage, notamment à travers son intégration dans un système de fichiers distribué, tel que RozoFS. Cette application au stockage distribué est le sujet de la seconde partie.



Application au stockage distribué

Introduction de la partie

La partie précédente a permis de définir le code à effacement Mojette permettant de répondre efficacement au problème de la transmission d'informations sur un canal non-fiable. Dans cette nouvelle partie, nous nous intéresserons à l'application de ce code dans un système de stockage distribué (NDSS). Dans ce contexte, le phénomène de panne est considéré comme la norme plutôt que l'exception. L'objectif de cette nouvelle partie est de concevoir un NDSS capable de gérer à la fois un seuil de redondance (capacité nécessaire à l'archivage de données froides), et de fournir de bonnes performances (nécessaire pour le traitement de données chaudes). Nous verrons en détail, les éléments suivants dans cette partie :

1. le chapitre 4 présente l'utilisation des codes à effacement dans le contexte du stockage distribué. En particulier, le cas du RAID-6, protégeant les données face à une panne, est introduit avant de généraliser l'étude. Ce chapitre contient les deux contributions suivantes : (i) l'évaluation théorique des performances des codes à effacement (ii) la comparaison des performances d'encodage et de décodage des implémentations des codes Mojette par rapport aux codes MDS de référence (de Reed-Solomon) ;

2. le chapitre 5 étudie l'intégration du code à effacement Mojette dans RozoFS, le DFS développé par Rozo Systems. Une évaluation des latences de lecture et d'écriture enregistrées par RozoFS sera comparée à celles fournies par le DFS référent : CephFS, qui utilise des techniques de réplication ;

3. dans le chapitre 6, nous proposons une nouvelle méthode distribuée pour ré-encoder de nouveaux symboles de mots de code (i.e. projections Mojette), sans avoir à reconstruire explicitement la donnée initiale. Cette technique sera particulièrement utile dans le cas de la réparation de supports de stockage, ou d'allocation dynamique de redondance.

4

Les codes à effacement dans le stockage distribué

Sommaire

Introduction

Dans les systèmes de stockage, les codes à effacement sont connus comme une alternative efficace aux techniques de réplication. Ils permettent notamment de réduire significativement la quantité de redondance nécessaire pour fournir une tolérance aux pannes. Une panne est un phénomène qui entraine l'indisponibilité d'une donnée. En pratique, ces indisponibilités sont les conséquences de problèmes de différentes natures (e.g. logicielle, matérielle, réseau, ...). Afin d'augmenter la disponibilité des données, il est nécessaire de distribuer ces informations de manière redondante, sur plusieurs supports de stockage. Ainsi, il est possible de supporter l'indisponibilité d'une partie de ces données en utilisant cette information redondante.

En 1988, Patterson et al. [PGK88] publient leurs travaux qui présentent cinq techniques d'agrégation de disques durs. Ces techniques d'agrégation sont regroupées sous la désignation *Redundant Array of Independant Disks* (RAID), qui peut se traduire par « Matrice redondante de disques indépendants ». Chacune de ces techniques, identifiées par un numéro, permet d'exploiter un ensemble de disques afin d'améliorer les performances et/ou la disponibilité des données. D'un côté, les performances de lecture et d'écriture sont améliorées grâce à la distribution des données qui permet d'exploiter l'ensemble des disques simultanément (*stripping*), cette technique est notamment à la base du RAID-0. De l'autre, des techniques permettent d'améliorer la disponibilité des données en générant de la redondance au sein du système de stockage. Il est par exemple possible de répliquer l'information sur l'ensemble des disques disponibles, comme en RAID-1. D'autres méthodes permettent de calculer r disques de parité à partir de k disques de données, ce qui permet de supporter la panne de r disques. C'est le cas en RAID-4 et RAID-5 pour $r = 1$, et RAID-6 pour $r = 2$. Ces techniques ont ainsi permis de démocratiser l'utilisation des codes à effacement au sein de systèmes de stockage. Elles sont encore aujourd'hui largement utilisées.

À l'origine, l'agrégation des ressources matérielles comme utilisée dans le stockage par les méthodes RAID, est rendue possible dans les années 80 avec la réduction significative de la taille et du prix des composants informatiques [Bel84]. En parallèle du monde du stockage, les microprocesseurs apparaissent à la même époque et permettent de fournir une puissance de calcul bon marché et compacte. Ces réductions permettent d'agréger les ressources de calcul au sein d'un système multiprocesseur. Krajewski [Kra85] utilise notamment l'expression *Array Processing* pour désigner une matrice de processeurs exécutant la même instruction sur un jeu de données.

Dans les systèmes de stockage organisés en RAID-4 et RAID-5, qui protègent face à une panne, les informations du disque de parité correspondent à la somme des informations des disques de données. En revanche, il devient plus compliqué de fournir des disques de parité supplémentaires. Il existe plusieurs méthodes pour calculer un deuxième disque de parité en RAID-6 [Per+15b]. Par exemple, les codes de Reed et Solomon [RS60] peuvent être utilisés pour calculer ces disques de parité. Cependant, plusieurs méthodes optimisées pour cette configuration ont été conçues et permettent de calculer ce deuxième disque de façon plus performante. En particulier, les *Array* codes rassemblent une famille

de codes qui ont été conçus dans cette optique. Les codes EVENODD de Blaum et al. [Bla+95] et les codes *Row Diagnoal Parity* (RDP) de Corbett et al. [Cor+04] font partie de cette famille de codes. Bien que plus performantes par rapport aux codes de Reed-Solomon, ces méthodes sont limitées à deux voir trois disques de parité, et se généralisent difficilement au delà [BBV96]. En conséquence, il n'existe pas de solution parfaite et les concepteurs de systèmes de stockage doivent privilégier soit les performances, soit la haute disponibilité des données.

Dans ce chapitre, une de nos contributions sera de fournir une étude sur les performances théoriques des codes conçus pour RAID-6 (i.e. $r = 2$). Plus spécifiquement, les critères de comparaison porteront sur des métriques adaptées au contexte du stockage telles que les performances d'encodage, de décodage et de mise à jour des données. La section 4.1 fournit une comparaison des codes RAID-6 traditionnels avec le code Mojette dans sa version systématique, tel que défini précédemment dans le chapitre 3. La section 4.2 compare de manière théorique les codes de Reed-Solomon et Mojette dans le cas général. Enfin, la section 4.3 présente une évaluation des performances des implémentations du code Mojette face aux meilleures implémentations des codes de Reed-Solomon fournies par Intel® Corporation [Int15]. Nous montrerons en particulier que le code Mojette bénéficie de meilleures performances théoriques dans l'ensemble des métriques, au prix d'un léger surcout de données à stocker. Ces aspects théoriques seront validés dans l'expérimentation en dernière partie.

4.1 Cas des codes à effacement $(k+2,k)$ pour le stockage : RAID-6

Cette section offre une comparaison des codes RAID-6. Pour cela, nous détaillerons tout d'abord l'architecture en RAID-6 dans la section 4.1.1. Puis, afin de réaliser cette comparaison, nous fournirons une liste de métriques dans la section 4.1.2, que nous exploiterons dans la section 4.1.3.

4.1.1 Architecture en RAID-6

La particularité de l'organisation des disques en RAID-6 est non seulement d'améliorer les performances de lecture et d'écriture, mais également d'apporter de la tolérance aux pannes. L'amélioration des performances provient de la distribution des données sur un ensemble de disques. Quant à la protection des données, qui se limite au cas où deux disques sont en panne, repose sur l'exploitation de deux disques de parité. On représente généralement cette organisation sous la forme d'une matrice de n disques possédant la même capacité de stockage. Ces n disques sont divisés en deux parties : (i) un ensemble de k disques de données ; (ii) un ensemble de $(n-k)$ disques de parité contenant les données de redondance calculées depuis les disques de données. Chaque disque est divisé en w blocs d'information de β bits. La figure 4.1 (cf. page 96) représente une matrice RAID-6 avec $w = 2$ blocs. En pratique, un bloc correspond à un mot de β bits (e.g. $\{8,\ldots,256\}$). En réalité, si l'on divise un disque en blocs de β bits, la valeur de w serait trop grande.

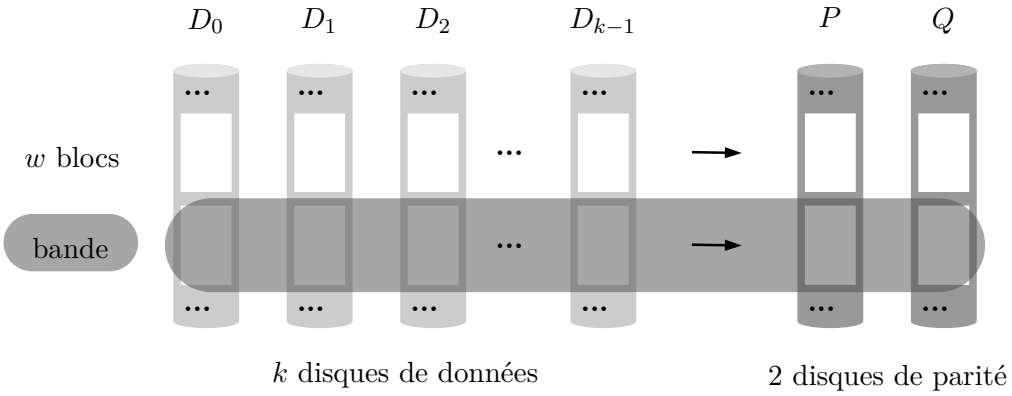


FIGURE 4.1 – Représentation d'une matrice de disques organisés en RAID-6. Un ensemble de k disques de données est utilisé pour encoder 2 disques de parité : $\mathcal{P}$ et $\mathcal{Q}$. Les disques sont partitionnés en w blocs. Une bande correspond à un ensemble de n blocs impliqués dans un processus d'encodage. Cette figure est extraite de [Pla+09].

C'est pourquoi, on applique la technique de codage sur des sous-ensembles de w blocs adaptés pour le code. Nous verrons dans la suite quelle valeur de w est utilisée pour chaque code. Les codes RAID-6 correspondent à une famille de codes généralement MDS, de paramètres $(n = k + 2, k)$. Puisque les codes sont MDS, la quantité de données de parité est optimale. En conséquence, le nombre de blocs w des disques de parité équivaut au nombre de blocs w des disques de données.

4.1.2 Métriques d'analyse de performance

Le contenu du premier disque de parité $\mathcal{P}$ est toujours calculé de la même manière. Les w blocs qu'il contient correspondent à des informations de parité horizontale des blocs des n disques de données. En revanche, plusieurs méthodes présentées dans ce chapitre permettent de calculer les données du disque $\mathcal{Q}$. Nous allons ainsi comparer ces techniques selon les métriques suivantes qui sont classiquement utilisées dans les travaux des codes étudiés :

- Le **coût à l'encodage** correspond au nombre d'opérations nécessaires pour encoder un disque de parité ;
- Le **coût de mise à jour** correspond au nombre d'opérations nécessaires pour modifier un bloc de données et mettre à jour les données associées dans les disques de parité. Par rapport aux études habituelles, nous considérerons une mise à jour différentielle dans notre étude. C'est-à-dire qu'au lieu de ré-encoder complètement les informations, nous calculerons seulement la différence avec la valeur d'origine. Par la suite, nous appliquerons par linéarité de l'opération d'encodage cette différence sur la donnée de parité, tel que proposé par ZHANG et al. [ZHX12]. Nous verrons

en particulier que dans la cas de certains codes, la position du bloc modifié dans la matrice a un impact significatif sur les performances ;

- Le **coût au décodage** correspond au nombre d'opérations nécessaires pour reconstruire l'information d'un disque de données qui subit une panne. On considère dans la suite qu'une panne entraîne l'indisponibilité totale d'un disque de données. Nous verrons durant notre analyse que certains codes possèdent de meilleures performances en reconstruisant l'information depuis le disque $\mathcal{Q}$ plutôt que $\mathcal{P}$.

4.1.3 Analyse des performances des codes RAID-6

Par rapport aux codes de Reed-Solomon qui peuvent fournir un taux de codage arbitraire, les *Array* codes sont conçus et optimisés pour gérer une quantité limitée de redondance. Dans la suite, il s'agira de décrire et d'analyser les performances des codes de Reed-Solomon, ainsi que deux *Array* codes (EVENODD et RDP). Enfin nous apporterons une comparaison des performances de ces codes avec le code Mojette. Nous verrons que ce dernier apporte de meilleures performances dans les différentes métriques définies précédemment.

Codes de Reed-Solomon

Les codes de Reed-Solomon peuvent être utilisés pour calculer les disques de parité. Contrairement à la construction d'une matrice d'encodage systématique dans le cas général, la matrice d'encodage dans le cas particulier du RAID-6 est simple et scindée en deux partie : $G = [I_k|R]$. La première partie correspond à la matrice carrée identité I_k de taille $(k \times k)$ et permet d'intégrer l'information à traiter dans les données encodées. La seconde partie correspond à la matrice de redondance R de taille $(2 \times k)$ permettant de calculer les valeurs des disques de parité $\mathcal{P}$ et $\mathcal{Q}$. La matrice R correspond à une matrice de Vandermonde de la forme :

$$\begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ \alpha^0 & \alpha^1 & \alpha^2 & \cdots & \alpha^{k-1} \end{pmatrix}, \tag{4.1}$$

avec $\alpha \in \mathrm{GF}(q)$ un générateur du corps. Si l'on utilise par exemple le corps de Galois $\mathrm{GF}(2^8)$ et un générateur de ce corps tel que $\alpha = 2$, l'opération d'encodage d'une bande correspond alors à l'équation matricielle suivante :

$$\begin{pmatrix} 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \ddots & 1 \\ 1 & 1 & 1 & \cdots & 1 \\ 1 & 2 & 4 & \cdots & 2^{k-1} \end{pmatrix} \times \begin{pmatrix} d_0 \\ d_1 \\ d_2 \\ \vdots \\ d_{k-1} \end{pmatrix} = \begin{pmatrix} d_0 \\ d_1 \\ d_2 \\ \vdots \\ d_{k-1} \\ p \\ q \end{pmatrix}, \tag{4.2}$$

où $p, q \in \mathrm{GF}(2^8)$ correspondent aux valeurs des données de parité (respectivement stockées dans les disques $\mathcal{P}$ et $\mathcal{Q}$), impliquées dans le calcul d'une bande. Dans la suite, nous allons uniquement considérer le calcul de p et q. Puisque dans notre représentation, un disque est composé de w bandes, l'encodage nécessite w fois cette opération. L'équation (4.2) montre ainsi que l'encodage du disque $\mathcal{P}$ ne requiert que $(k-1)w$ additions, tandis que le calcul des valeurs du disque $\mathcal{Q}$ nécessite $(k-1)w$ additions et kw multiplications. Les mises en œuvre des codes Reed-Solomon souffrent notamment de l'implémentation des opérations dans les corps de Galois. La multiplication est en effet une opération coûteuse qui peut être implémentée de plusieurs manières. Par exemple, afin d'améliorer les performances de l'encodage, Anvin [Anv04] propose de décomposer le calcul de q en utilisant la méthode de Ruffini-Horner, illustrée dans l'équation (4.4) :

$$q = d_0 + (2 \times d_1) + (4 \times d_2) + (8 \times d_3) + (16 \times d_4) \tag{4.3}$$
$$= d_0 + (2 \times d_1 + (2 \times d_2 + (2 \times d_3 + (2 \times d_4)))). \tag{4.4}$$

Dans cette représentation, il suffit alors de développer une implémentation efficace de la multiplication par deux.

La modification d'un bloc de données entraîne la mise à jour de deux blocs de parité. Une addition est réalisée pour calculer la différence, deux opérations supplémentaires sont nécessaires pour mettre à jour le bloc correspondant dans $\mathcal{P}$ puis dans $\mathcal{Q}$. Enfin, une multiplication permet de finir la mise à jour du bloc dans $\mathcal{Q}$. En conséquence, la modification d'un bloc entraîne trois additions et une multiplication.

Quant au décodage, il nécessite autant d'opérations que l'encodage en considérant que la matrice inverse est pré-calculée. Pour reconstruire un disque, dans le cas où il y a une seule panne, le processus de décodage favorise l'utilisation de $\mathcal{P}$ puisque cela entraînerait le recours à $(k-1) \times w$ additions. En revanche, l'utilisation du disque $\mathcal{Q}$ nécessite des opérations de multiplication supplémentaires, plus complexes à mettre en place. Dans le cas de deux pannes, l'opération nécessite autant d'opérations qu'à l'encodage, c'est à dire : $2 \times (k-1)w$ additions et $k \times w$ multiplications.

Les *Array* codes

Les *Array* codes ont été conçus à l'origine comme une alternative aux codes de Reed-Solomon afin d'éviter les opérations de multiplication dans les corps de Galois. Bien que ces codes fournissent une quantité de redondance limitée, ils ne réalisent que des opérations d'addition, et sont en conséquence performants. Les disques $\mathcal{P}$ et $\mathcal{Q}$ sont respectivement calculés en utilisant des bandes horizontales et diagonales. Le calcul de $\mathcal{P}$ correspond également à un mot de code de parité. On s'intéressera en particulier à la conception du disque $\mathcal{Q}$ pour les codes EVENODD [Bla+95] et RDP [Cor+04].

Les codes EVENODD Les codes EVENODD ont été conçus en 1995 par Blaum et al. [Bla+95]. Les paramètres du code sont contraints de telle manière que $(w+1)$ soit premier, et que $k \leq w+1$. Lorsque le disque $\mathcal{Q}$ est impliqué, que ce soit pour l'encodage

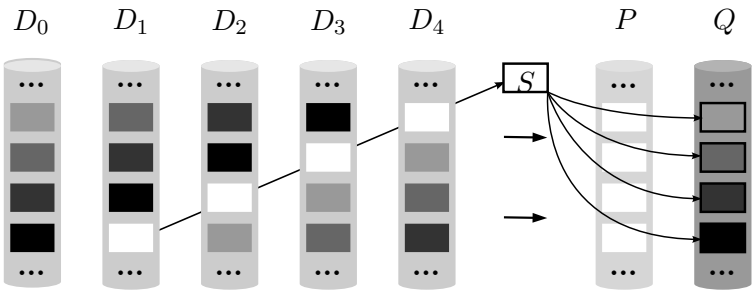


FIGURE 4.2 – Représentation d'un code EVENODD sur une matrice $(k=5, w=4)$. La figure s'intéresse en particulier au calcul des valeurs du disque $\mathcal{Q}$, basé sur la valeur des données des disques D_i. L'ajusteur S correspond à la somme des éléments de la diagonale blanche. Sa valeur est additionnée à chaque blocs de $\mathcal{Q}$, qui sont déterminés par une parité diagonale. Figure inspirée de [Pla+09].

ou le décodage, une valeur intermédiaire S (nommée *ajusteur*) doit être déterminée afin de garantir la propriété MDS du code. La figure 4.2 représente le processus pour calculer $\mathcal{Q}$. La valeur de l'ajusteur correspond à la somme des éléments suivant la diagonale blanche. La valeur des blocs de $\mathcal{Q}$ se calcule à partir de la somme des éléments des disques de données suivant différentes diagonales (représentées par des couleurs différentes sur la figure 4.2), à laquelle on rajoute la valeur de S.

Lors de l'encodage, cet ajusteur nécessite $(k-2)$ additions. L'encodage nécessite $(k-1)\times w$ additions pour calculer $\mathcal{P}$ et $(k-1)w+k-2$ pour calculer les valeurs de $\mathcal{Q}$. En conséquence, le disque $\mathcal{Q}$ requiert plus d'opérations que le disque $\mathcal{P}$.

Les conséquences de la modification d'un bloc de données dépendent de la position de ce bloc. Dans la plupart des cas, cette modification entraîne la mise à jour d'une valeur optimale de 2 blocs de parité. On calcule ainsi la différence entre l'ancienne et la nouvelle valeur du bloc, puis on met à jour un bloc de $\mathcal{P}$ et un bloc de $\mathcal{Q}$, ce qui coûte trois additions. En revanche, lorsque la modification affecte un bloc de la diagonale qui définit la valeur de S, l'ensemble des blocs du disque $\mathcal{Q}$ est affecté. Par conséquent, w additions sont réalisées, une autre est nécessaire pour calculer la différence, et une dernière pour la mise à jour du bloc du disque $\mathcal{P}$. Cette situation correspond au pire cas, et nécessite alors $(w+2)$ additions.

Concernant la reconstruction d'un disque, plusieurs scénarios sont également possibles. Quand une panne affecte un seul disque, il est recommandé de reconstruire en utilisant le disque $\mathcal{P}$ pour ne pas avoir à calculer la valeur de S. Une panne est alors réparée en utilisant $(k-1)\times w$ additions. Lorsque les disques de données subissent deux pannes, il est tout d'abord nécessaire de recalculer la valeur de S. Le coût du calcul de S dépend de la position des disques en panne. Soit $\gamma(S)$ le nombre d'opérations nécessaires pour recalculer sa valeur, et soit deux entiers $i,j \in \mathbb{Z}_w$ tel que $i \neq j$, correspondant respectivement à l'index du premier et du second disque en panne. On distingue alors les trois cas suivants :

1. Si $i=0$ et $j \geq k$, alors la méthode utilisé lors de l'encodage permet de recalculer la

valeur de S, puisque le disque en panne n'impacte pas la diagonale blanche. En conséquence, $\gamma(S) = k - 2$ opérations ;

2. Si $i < k$ et $j = k$, alors il est nécessaire de calculer S à partir de n'importe quelle autre diagonale, donc $\gamma(S) = k - 1$ opérations ;
3. Si $i, j < k$, alors S peut être calculée en additionnant l'ensemble des valeurs des disques de parité. Cela nécessite $\gamma(S) = 2(k - 2)$ opérations. Cette situation correspond au pire cas.

Une fois que la valeur de S est calculée, il est possible de reconstruire les éléments des disques effacés par un processus itératif. La première itération consiste à reconstruire la valeur d'un bloc effacé en utilisant une bande diagonale. La seconde itération utilise la bande horizontale de ce bloc afin de reconstruire un nouveau bloc. En utilisant alternativement les parités diagonales et horizontales, on parvient à reconstruire l'ensemble des données effacées. Le coût total de la reconstruction est $2(k - 1)w + \gamma(S)$. Le pire cas correspond à la perte de deux disques impliqués dans la diagonale de S. Dans cette situation, la reconstruction nécessite $2(k - 1)w + 2(k - 2)$ additions.

Les codes RDP Les codes RDP conçus par Corbett et al. [Cor+04] ont également une contrainte géométrique telle que $k \leq w$. Ces codes sont similaires aux EVENODD mais ne nécessitent pas d'ajusteur. Les informations du second disque de parité sont également calculées en utilisant des parités diagonales. En revanche, la diagonale spéciale précédemment utilisée pour calculer S n'intervient pas dans ce calcul. De plus, les informations du disque $\mathcal{P}$ interviennent dans le calcul des valeurs du disque $\mathcal{Q}$, ce qui entraîne une dépendance et une séquentialité dans la mise en œuvre de l'encodage. En particulier, les codes RDP requièrent moins d'opérations pour l'encodage, le décodage et la mise à jour, que les codes de Reed-Solomon et EVENODD.

Le nombre d'opérations nécessaires pour le calcul des informations de $\mathcal{P}$ et de $\mathcal{Q}$ est identique et correspond à $(k - 1)w$ additions. De manière similaire aux EVENODD, le coût de la mise à jour d'un bloc dépend de la position du bloc. Lorsque celui-ci est situé sur la première ligne ou sur la diagonale spéciale, seulement deux blocs de parité sont impactés sur $\mathcal{P}$ et $\mathcal{Q}$ respectivement. En conséquence, trois opérations sont nécessaires. Dans tous les autres cas, trois blocs sont impactés du fait de l'interdépendance entre les disques de parité. La reconstruction d'un disque de données est quant à elle réalisée en $(k - 1)w$ opérations quel que soit le disque de parité utilisé. En conséquence, $2(k - 1)$ opérations sont nécessaires pour reconstruire l'information lorsque deux disques sont en panne.

Le code Mojette

Dans la représentation des codes $(k + 2, k)$, le code à effacement Mojette, sous forme systématique, calcule deux projections à partir d'une grille de hauteur k. Il est ainsi possible de reconstruire deux lignes de la grille à partir de ces projections. Le choix des directions de projection sera motivé par la réduction du coup de la redondance. On

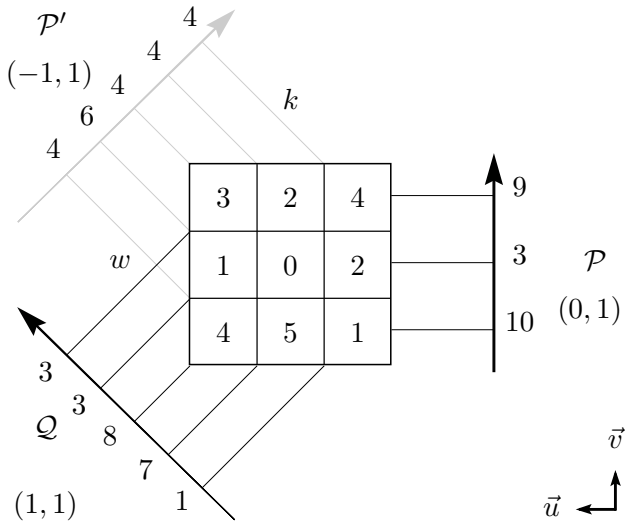


FIGURE 4.3 – Code à effacement Mojette d'une image (3×3) pour les directions $\{(p,q)\}$ dans l'ensemble $\{(-1,1),(0,1),(1,1)\}$. La base utilisée est représentée par $\vec{u}$ et $\vec{v}$. $\mathcal{P}$, $\mathcal{Q}$, et $\mathcal{P}'$ correspondent à des disques de parité.

choisira ainsi comme ensemble de projections $\{(p,q)\} = \{((0,1),(\pm 1,1)\}$, où le disque $\mathcal{P}$ contiendra les éléments de projections suivant la direction $(0,1)$, et $\mathcal{Q}$ suivant $(\pm 1,1)$. La figure 4.3 montre cette représentation pour une grille (3×3). En particulier le choix du signe pour la valeur de p_Q n'a pas d'impact ni sur la complexité des opérations, ni sur le coût de la redondance.

Pour l'encodage, le nombre d'opérations nécessaires pour le calcul d'une projection dépend de la taille de la grille et de la direction de projection. On le définit ainsi :

$$\gamma(k,w)^{(p,q)} = k \times w - B(k,w,p,q), \tag{4.5}$$

où $B(k,w,p,q)$ correspond au nombre de bins de la projection (voir équation (2.27), page 67). Puisque le disque $\mathcal{P}$ correspond à la projection suivant l'horizontal, le nombre de bins $B(k,w,0,1)$ vaut w. Le calcul de cette projection nécessite alors $(k-1)w$ opérations. Le nombre d'opérations pour le disque $\mathcal{Q}$ correspond quant à lui à $(k-1)w-k+1$. On remarque ici que pour la première fois, le nombre d'opérations nécessaires est inférieur pour calculer le disque $\mathcal{Q}$ que pour le disque $\mathcal{P}$. On peut également noter que si l'on utilise le disque $\mathcal{P}'$ (correspondant à la projection suivant la direction $(-1,1)$) à la place du disque $\mathcal{P}$, on réduit le nombre d'opérations global au prix d'un coût de redondance plus élevé.

Les mises à jour dans le cas du code Mojette sont optimales. En effet, la modification d'un bloc de donnée impacte uniquement un bloc de parité correspondant dans chaque disque. Dans le cas étudié, on modifie uniquement deux bins quelle que soit la position du pixel modifié.

Dans le cas de la perte d'un disque, il est cette fois plus avantageux de reconstruire

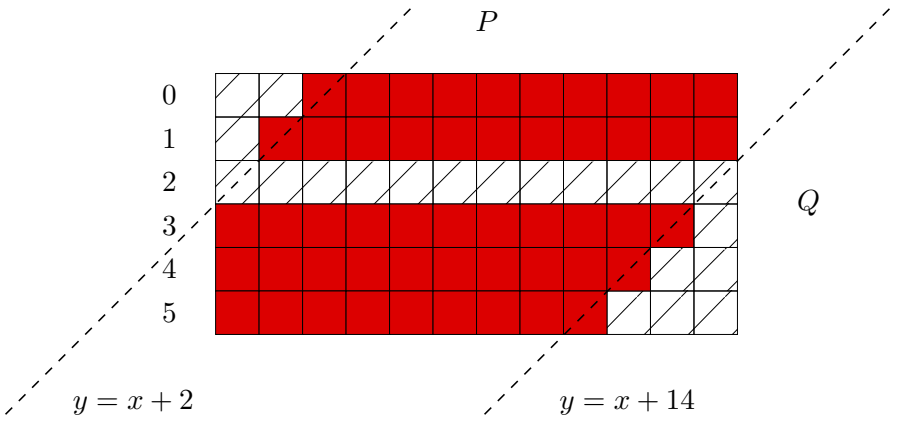


FIGURE 4.4 – Représentation de la méthode de détermination du nombre d'opérations nécessaires pour reconstruire une ligne l à partir d'une projection. Cet exemple représente une grille ($P = 12, Q = 6$). On cherche à reconstruire la ligne $l = 2$ à partir de la projection suivant la direction ($p = 1, q = 1$). Les éléments en rouge représentent les éléments impliqués dans la reconstruction de la ligne.

la donnée en utilisant le disque $\mathcal{Q}$ qu'en utilisant le disque $\mathcal{P}$. Rappelons que le nombre d'opérations nécessaires à la reconstruction d'un pixel dépend de sa position dans la grille. En effet, un pixel situé dans un coin de la grille nécessitera en général moins d'opérations qu'un pixel situé au milieu de la grille. De plus, ce nombre va dépendre de la projection utilisée pour la reconstruction. Si l'on reprend l'exemple d'un pixel situé dans un coin de la grille, aucune opération n'est nécessaire si $(p, q) \neq (0, 0)$. En revanche, si $(p, q) = (0, 1)$, alors $(Q - 1)$ opérations seront nécessaires.

Soit l l'index d'une ligne à reconstruire. La figure 4.4 représente la situation où l'on souhaite reconstruire la ligne $l = 2$ d'une grille ($P = 12, Q = 8$) en utilisant la projection suivant la direction ($p = 1, q = 1$). Les éléments de la grille en rouge représentent les pixels utilisés dans la reconstruction de la ligne l. Le nombre d'opérations nécessaires à la reconstruction d'une ligne l est défini par l'ensemble des pixels représentés en rouge. Cet ensemble correspond aux éléments de la grille discrète (hors éléments de la ligne l) contenus entre les deux droites de projection $y = p_i x + l$ et $y = p_i x + l + P$, qui passent par les extrémités de la ligne l. Dans le cas général, ce nombre correspond à la surface de la grille ($P \times Q$) à laquelle on soustrait le nombre d'éléments de la ligne à reconstruire P, et auquel on soustrait la surface de deux triangles. Dans l'exemple de la figure 4.4, le triangle supérieur possède une base et une hauteur de l, et le triangle inférieur possède une base et une hauteur de $(Q - l)$. Le nombre d'opérations nécessaires pour reconstruire cette ligne correspond à :

$$\gamma(P, Q, l)^{(1,1)} = (Q - 1)P - \frac{l(l + 1)}{2} - \frac{(Q - l - 1)(Q - l)}{2}. \tag{4.6}$$

La figure 4.5 représente le nombre d'opérations nécessaires pour reconstruire une ligne depuis la projection $(1, 1)$ en fonction de l'index de cette ligne. On peut remarquer

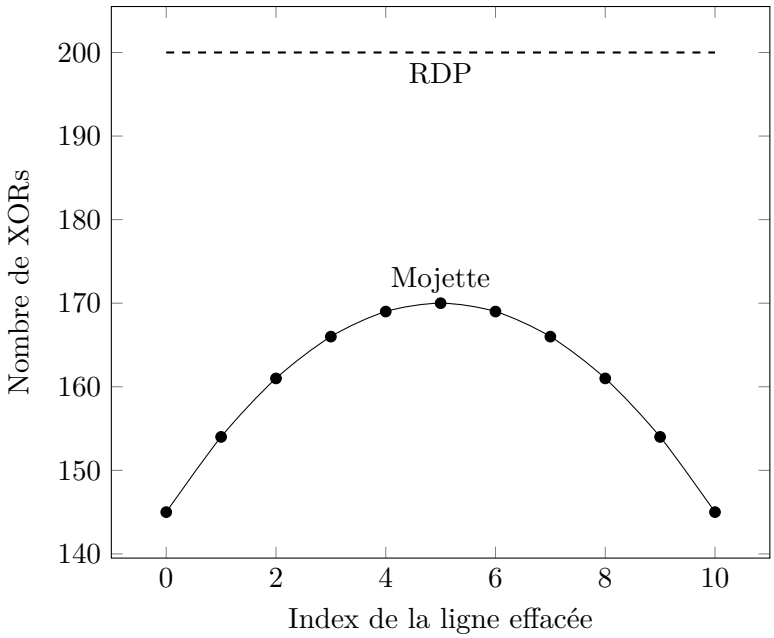


FIGURE 4.5 – Nombre d'opérations nécessaires pour la reconstruction Mojette depuis la projection de direction $(1,1)$, en fonction de la position de la ligne effacée. La grille utilisée correspond à $k = 11$ et $w = 20$. La ligne en tirets représente les performances obtenues par les codes RDP (i.e. $(k-1)w$).

le comportement quadratique de cette courbe. En particulier, les lignes situées aux extrémités (i.e. $l = 0$ et $l = k-1$) nécessitent moins d'opérations pour être construites. En effet, la surface des pixels impliqués dans la reconstruction est moins importante pour ces lignes, alors qu'elle est maximale pour la ligne centrale. En particulier, ces lignes bénéficient du fait qu'elles possèdent des pixels proches des coins de la grille, et dont le nombre de dépendances est réduit.

Bilan et discussion

Le tableau 4.1 (cf. page 104) résume les résultats obtenus précédemment. On y exprime le nombre d'opérations nécessaires pour chaque métrique détaillée en première section, en fonction des différents codes. En particulier, on distingue les métriques d'encodage et de décodage en fonction de l'utilisation du disque $\mathcal{P}$ ou $\mathcal{Q}$. On remarque que la génération ou la réparation relative au disque $\mathcal{P}$ nécessite toujours $(k-1)w$ opérations. En effet, ce disque correspond à la parité horizontale des disques de données. En revanche, le nombre d'opérations nécessaires quand on interagit avec $\mathcal{Q}$ forme un critère intéressant puisque celui-ci varie pour chaque code. Le critère de mise à jour d'un bloc est également décisif et est optimal pour la Mojette.

Code	*Encoder P*	*Encoder Q*	*Mise à jour*	*Décoder depuis P*	*Décoder depuis Q*
RS[a]	$(k-1)w$	$(k-1)w + (kw)_{\otimes}$	$3 + 1_{\otimes}$	$(k-1)w$	$(k-1)w + (kw)_{\otimes}$
EVENODD	$(k-1)w$	$(k-1)w + k - 2$[b]	$w + 2$[b]	$(k-1)w$	$(k-1)w + 2(k-2)$[b]
RDP	$(k-1)w$	$(k-1)w$	4[b]	$(k-1)w$	$(k-1)w$
Mojette	$(k-1)w$	$(k-1)w - k + 1$	3	$(k-1)w$	$\rho(k, w, l)^{(1,1)}$[c]

[a] Pour les codes de Reed-Solomon, les opérations de multiplication sont symbolisées par $\otimes$.

[b] Le nombre d'opérations correspond au pire cas (par exemple, ça peut dépendre de $\gamma(S)$ pour EVENODD).

[c] Cette valeur dépend également de l'index l de la ligne perdue, voir l'équation (4.6).

Table 4.1 – Tableau de comparaison du nombre d'opérations nécessaires pour différents code à effacement selon les métriques définies dans la section 4.1.2. Les paramètres k et w correspondent respectivement au nombre de disques de données, et au nombre de blocs qu'ils contiennent.

Les spécificités des codes de Reed-Solomon Les performances théoriques des codes de Reed-Solomon sont compliquées à définir. En effet, comme spécifié précédemment, la multiplication dans les corps de Galois peut être implémentée de différentes manières : par exemple Blaum et Roth [BR93] proposent d'utiliser des permutations circulaires, Rizzo [Riz97a] propose des tables de correspondance et Blömer et al. [Blö+95] utilisent des matrices binaires. C'est pourquoi dans le tableau 4.1, nous distinguons les opérations de multiplication des opérations d'addition en utilisant le symbole $\otimes$.

Analyse des performances Afin de comparer les performances d'encodage et de décodage, prenons $(k-1)w$ opérations, comme référence. Cette référence correspond au meilleur résultat obtenu par les codes MDS RDP. Concernant l'encodage, les codes de Reed-Solomon et EVENODD ont tous les deux un coût supplémentaire par rapport à cette référence : (i) les codes de Reed-Solomon nécessite deux multiplications supplémentaires ; (ii) les EVENODD nécessitent $(k-2)$ additions supplémentaires à cause du calcul de S. Le code Mojette en revanche, requiert moins d'opérations pour calculer le disque Q. On peut également rappeler qu'il s'agit du seul code qui nécessite moins d'opérations pour calculer $\mathcal{Q}$ que pour calculer $\mathcal{P}$.

Côté reconstruction d'un disque en utilisant les données du disque de parité $\mathcal{Q}$, le même constat peut être fait. Les codes RDP nécessitent encore $(k-1)w$ opérations. De manière similaire à l'encodage, les codes de Reed-Solomon et EVENODD ont un surcout calculatoire respectivement dû aux multiplications et au calcul de S. Pour le code Mojette, le nombre d'opérations à réaliser dépend du disque inaccessible (voir équation (4.6)). Cependant, quel que soit le disque inaccessible, sa reconstruction par le disque $\mathcal{Q}$ nécessitera toujours un nombre d'opérations inférieur à $(k-1)w$ comme le montre la figure 4.5.

Quant aux mises à jour d'un bloc de données, bien que tous les codes soient capables d'atteindre la meilleure performance de 3 additions dans le meilleur des cas, seul le code Mojette les atteint quelle que soit la position du bloc modifié. En conclusion, les

codes Mojette nécessitent moins d'opérations pour l'ensemble des métriques étudiées par rapport aux codes MDS utilisés pour le stockage distribué en RAID-6. Notons toutefois que ce gain en performance nécessite une quantité de redondance supplémentaire. Nous avons cependant montré dans le chapitre précédent que ce coût est modéré et tend vers l'optimal quand la largeur de la grille augmente.

Bien que les performances théoriques soient liées au nombre et à la nature des opérations réalisées durant l'encodage et le décodage, d'autres critères entrent en jeu en pratique. Comme nous l'avons vu dans le chapitre précédent, les opérations doivent être réalisées au plus près du processeur. Un autre enjeu correspond à la localité spatiale de nos données. Cette considération correspond à la caractéristique de notre algorithme à traiter les données de manière séquentielle. L'intérêt d'un accès séquentiel aux données repose sur deux considérations : (i) il n'est pas nécessaire de calculer la position de la donnée et d'atteindre sa position (les disques durs mécaniques par exemple perdent un temps considérable à déplacer les têtes de lecture pour accéder à des données distantes) ; (ii) il est possible de charger en avance la donnée pour la placer dans les zones mémoires près du processeur (c'est notamment l'objectif de l'appel système *readahead* sur Linux). Bien qu'il soit possible de favoriser ces comportements, il est compliqué de les quantifier. Dans le cas du code à effacement Mojette, nous avons vu que la géométrie du problème permet de favoriser la localité spatiale.

4.2 Généralisation de la construction des codes

Après avoir comparé ces différents codes dans le cas du RAID-6, nous nous intéressons à présent au cas général. Dans la suite, nous comparerons donc les performances des codes de Reed-Solomon dans la section 4.2.1, au code Mojette, qui sera détaillé dans la section 4.2.2.

4.2.1 Reed-Solomon

Les codes à effacement de Reed et Solomon correspondent à des codes (n, k) MDS, dont la valeur des paramètres peut être arbitrairement choisie. En particulier, cette application calcule n mots encodés à partir de k mots de données tel que $k \geq n$. Pour cela, une matrice d'encodage de taille $n \times k$ est utilisée pour la transformation. Pour définir un code MDS, cette matrice doit nécessairement posséder la propriété suivante : toute sous-matrice de taille $k \times k$ doit être inversible. À l'origine, une matrice de Vandermonde était utilisée puisque qu'elle possède une telle propriété. Blömer et al. [Blö+95] ont par la suite utilisé des matrices de Cauchy qui partage cette propriété. La complexité d'inversion d'une telle matrice est de $\mathcal{O}(k^2)$. Dans leurs travaux, Blömer et al. [Blö+95] ont également proposé une représentation particulière de cette matrice permettant d'implémenter les opérations de multiplication par des fonctions de OU-exclusif.

Vandermonde-RS Les matrices de Vandermonde V sont définies par un vecteur de n éléments $(\alpha_0, \alpha_1, \ldots, \alpha_{n-1})$. Ce type de matrice s'écrit sous la forme suivante :

$$V = \begin{pmatrix} 1 & 1 & \cdots & 1 & 1 \\ \alpha_0^1 & \alpha_1^1 & \cdots & \alpha_{k-2}^1 & \alpha_{k-1}^1 \\ \alpha_0^2 & \alpha_1^2 & \cdots & \alpha_{k-2}^2 & \alpha_{k-1}^2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \alpha_0^{n-1} & \alpha_1^{n-1} & \cdots & \alpha_{k-2}^{n-1} & \alpha_{k-1}^{n-1} \end{pmatrix} \in F_q^{k \times n}, \tag{4.7}$$

où α_i correspondent aux coefficients d'une matrice de Vandermonde. Pour que l'ensemble des sous-matrices $k \times k$ soit inversible, il est nécessaire que les éléments α_i du corps fini soient deux à deux distincts. Dans le cas d'un code de Reed-Solomon non-sytématique, la matrice d'encodage correspond à une matrice de Vandermonde telle que $G = V$. Pour obtenir un code de Reed-Solomon systématique avec des paramètres (n, k) arbitraires, il n'est pas possible de construire une matrice d'encodage de la forme $G = [I_k|V]$ (comme nous l'avons vu précédemment dans le cas particulier de RAID-6). Dans une telle matrice, toute sous-matrice carrée n'est pas inversible [LF04]. On peut cependant obtenir un code de Reed-Solomon systématique à partir d'une matrice de Vandermonde en appliquant un algorithme d'élimination de Gauss sur les colonnes de V afin de faire apparaître une matrice identité. Cette méthode est décrite dans Plank et Ding [PD03]. Une autre méthode est proposée par Lacan et Fimes [LF04]. Une fois que l'on a déterminé la matrice d'encodage G, l'opération d'encodage correspond à calculer $Y = GX$.

Toute sous-matrice carrée G' d'une matrice de Vandermonde est inversible (en particulier une sous-matrice $k \times k$). Cette caractéristique des matrices de Vandermonde permet d'inverser l'opération réalisée à l'encodage, et de reconstruire l'information perdue. Quand l'information subit des effacements, les lignes correspondantes dans la matrice d'encodage sont supprimées pour donner une sous-matrice G' de taille $(k \times k)$. L'inverse de cette matrice est déterminée afin de retrouver le message initial : $X = G'^{-1}Y'$, où Y' correspond à un mot de code dégradé. Considérons deux ensembles $i \in \mathbb{Z}_k$ et $j \in \mathbb{Z}_w$. Ainsi, $d_{i,j}$ correspond au bloc de données situé dans la colonne i et à la ligne j. Dans ce cas, les données d'un disque de parité $\mathcal{R}_j$ sont calculées ainsi :

$$\mathcal{R}_j = \bigoplus_{i=0}^{k-1} d_{i,j}\alpha^i. \tag{4.8}$$

L'équation (4.8) montre que $(k-1)$ additions et k multiplications sont nécessaires pour calculer un bloc d'un disque de parité.

Les codes de Reed-Solomon souffrent des multiplications de l'équation (4.8) réalisées dans un corps de Galois. De nombreux travaux ont été réalisés afin de produire des implémentations efficaces de l'opération de multiplication. Par exemple, Blaum et Roth [BR93] ont proposé une méthode pour remplacer ces multiplications par des opérations de soustractions et de rotations cycliques. Par la suite, Rizzo [Riz97a] a proposé d'utiliser des tables de correspondance.

Cauchy-RS Une version des codes de Reed-Solomon basée sur des matrices de Cauchy a été proposée par Blömer et al. [Blö+95]. Un avantage de cette technique

est de permettre une inversion matricielle plus efficace en $\mathcal{O}(n^2)$. De plus, les travaux de Blömer et al. ont permis de remplacer les opérations de multiplications par des additions. Pour cela, chaque élément de la matrice d'encodage est étendu par β dans les deux directions. Ses performances sont ainsi liées au nombre de 1 présents dans la matrice d'encodage ou de décodage. Des travaux ont été menés afin de rendre les matrices les plus creuses possible. En effet, étant donné les paramètres (n, k) d'un code défini dans un corps de Galois, il existe une quantité importante de matrices de Cauchy permettant d'encoder l'information. Plank et Xu [PX06] ont déterminé que chaque matrice n'est pas égale en matière de performance. Si l'on souhaite trouver la meilleure matrice, il faut énumérer tous les cas possibles, dont le nombre croît de manière exponentielle avec n. En conséquence, cette méthode peut convenir pour des codes avec des paramètres (n, k, w) de faibles valeurs (e.g. $w \leq 4$). Plank et Xu donnent toutefois un algorithme pour déterminer une « bonne » matrice de Cauchy. Cependant, le nombre de 1 dépend des ensembles d'éléments du corps de Galois choisis pour construire la matrice de Cauchy. Or, aucune méthode efficace n'existe aujourd'hui afin de définir ce nombre minimum de 1 dans la matrice d'encodage.

4.2.2 Généralisation du code Mojette

Voyons à présent les performances de l'encodeur, puis du décodeur Mojette dans le cas général.

Performances de l'encodeur Mojette

Nous allons ici analyser le nombre d'opérations nécessaires pour le calcul d'une projection. Bien que la génération d'une projection met en jeu l'ensemble des éléments de la grille discrète une et une seule fois (voir équation (2.23), page 62), le nombre γ d'opérations nécessaires pour l'encodage varie en fonction de deux paramètres : la taille de la grille, et la direction de projection. Le nombre d'additions nécessaires pour générer une projection $\text{Proj}_f(p, q, b)$ correspond à :

$$\begin{aligned} \gamma(P,Q)^{(p,q)} &= P \times Q - B(P,Q,p,q), \\ &= P \times Q - ((Q-1)|p| + (P-1)|q| + 1)\,, \end{aligned} \tag{4.9}$$

et représente le nombre d'éléments de la grille discrète $(P \times Q)$ auquel on soustrait le nombre de bins de la projection, tel que défini dans l'équation (2.27) (cf. page 67). Considérons à présent que l'on fixe la taille de la grille, ainsi qu'un paramètre de projection. Si on considère comme précédemment que $q_i = 1$, cette équation devient :

$$\gamma(P,Q)^{(p,1)} = P \times Q - ((Q-1)|p| + P), \tag{4.10}$$

$$= (Q-1)(P-|p|). \tag{4.11}$$

En conséquence, quand les dimensions de la grille sont fixées, si la valeur de $|p|$ augmente, le nombre d'opérations nécessaires pour générer une projection $\gamma(P,Q)^{(p,q)}$ diminue. Cela

signifie que pour une taille de grille fixée, plus une projection est grande, moins elle nécessite d'opérations d'addition pour être calculée. Aussi, si seules les performances sont essentielles pour une application, on choisira des projections avec de grandes valeurs de $|p|$. Pour un ensemble de projections $\{(p_i, 1)\}$ donné, l'ensemble des opérations nécessaires pour l'encodage correspond à $\sum_i \gamma(P,Q)^{\{(p_i,1)\}}$.

Performances du décodeur systématique

Le nombre d'opérations nécessaires $\gamma(k, w, l)$ pour reconstruire un pixel d'index i depuis une projection de direction $(p, 1)$ est défini par :

$$\gamma(k,w)_i^{(p,1)} = \left\lceil \sum_{a=0}^{w-1} \sum_{b=0}^{k-1} \Delta\left(i + a - (b \times |p|)\right) \right\rceil - 1. \tag{4.12}$$

En conséquence, le nombre d'opérations nécessaires pour reconstruire une ligne d'index l correspond à la somme des valeurs calculées à partir de l'équation (4.12) pour chaque pixel de la ligne :

$$\gamma(k,w)_l^{(p,1)} = \sum_{i=l-w+1}^{i=l} \left(\gamma(k,w)_i^{(p,1)}\right). \tag{4.13}$$

Dans le cas où e lignes sont effacées, le nombre total d'opérations nécessaires correspond à la somme des opérations nécessaires pour chaque ligne. Ainsi, une projection différente, issue d'un ensemble suffisant $\{(p_i, 1)\} \mid i \in \mathbb{Z}_e$, est associée à la reconstruction d'une ligne. On détermine alors le nombre d'opérations nécessaires ainsi :

$$\gamma(k,w)_e^{\{(p_e,1)\}} = \sum_e \left(\gamma(k,w)_{l_e}^{(p_e,1)}\right). \tag{4.14}$$

4.3 Expérimentations

Dans cette section, nous évaluons les performances du code à effacement Mojette et comparons ces résultats avec les performances des meilleures implémentations des codes de Reed-Solomon. Nous détaillons en section 4.3.1 les caractéristiques des implémentations étudiées. La section 4.3.2 présente la mise en œuvre de l'expérimentation nous permettant de mesurer les performances de ces implémentations. Enfin, dans la dernière section 4.3.3, nous exposerons puis analyserons les résultats obtenus.

4.3.1 Les implémentations à comparer

Nous avons choisi de comparer les implémentations du code à effacement Mojette avec une implémentation des codes de Reed-Solomon. De par leur popularité et leur accessibilité, les codes de Reed-Solomon représentent un compétiteur essentiel pour notre comparaison. Ces codes sont en effet, largement distribués à travers de nombreuses bibliothèques.

Implémentations Mojette

Nous avons implémenté une version systématique du code à effacement Mojette en langage C. Le choix de ce langage est judicieux lorsque l'on développe une technique de codage devant fournir de hautes performances. En effet la possibilité de gérer la mémoire, ainsi que le recours à diverses instructions particulières du processeur, permettent d'atteindre d'excellentes performances [Hun11].

Dans la suite, nous reprenons la terminologie utilisée dans le chapitre 2. En pratique, la taille des pixels et des bins doivent correspondre à un mot machine. Un mot correspond à l'unité de base, exprimée en bits, manipulée par un processeur. Pour les architectures classiques, la taille d'un mot correspond à 32 ou 64 bits. Il s'agit plus exactement de la taille des registres du processeur. Par conséquent, un processeur est d'autant plus rapide que ses mots sont longs puisqu'une plus grande quantité d'information est traitée à chaque cycle. Nous avons alors fixé la taille des bins et pixels à 64 bits. Cette valeur correspond à la taille des registres des architectures usuelles aujourd'hui.

La plupart des processeurs proposent des extensions de leur jeu d'instructions afin d'améliorer les performances de certains traitements. Les instructions *Single Instruction, Multiple Data* (SIMD) correspondent à un mode de fonctionnement du processeur qui permet de profiter de parallélisme. Plus particulièrement, il s'agit d'appliquer en parallèle la même instruction sur plusieurs données afin d'obtenir plusieurs résultats. Bien que proposées par Flynn en 1966 et utilisées pour la première fois dans le supercalculateur Cray-1 en 1976, les instructions SIMD ont été disponibles dans les processeurs grand public d'Intel et AMD qu'à partir de 1997. Les processeurs Intel par exemple, proposent des extensions pour flux SIMD (*Streaming SIMD Extensions* SSE) qui ajoutent jusqu'à 16 registres de 128 bits et 70 instructions supplémentaires pour les processeurs x86. Ce mode de fonctionnement permet donc de traiter 2048 octets en parallèle, en un cycle processeur. Les applications peuvent alors bénéficier de cela dès lors qu'une instruction peut être réalisée sur plusieurs données. En pratique, ce mode est largement utilisé dans les applications multimédias, scientifiques ou financières. Dans le stockage, il permet notamment d'augmenter les performances de traitement du RAID logiciel implémenté dans Linux en réalisant plusieurs instructions en parallèle [Anv04].

Ce mode de fonctionnement est donc très intéressant pour notre code à effacement étant donné que la quantité de données traitées pour un cycle CPU est cruciale dans les systèmes d'information. Les algorithmes d'encodage et de décodage Mojette sont propices à ce fonctionnement puisque nous appliquons une instruction, qui correspond à l'addition, sur une multitude de données, représentées par les éléments de la grille discrète et des projections. En conséquence, dans notre mise en œuvre, l'addition est implémentée par des opérations de OU exclusif (XOR), correspondant à des additions modulo deux, sur des données de 128 bits.

Dans cette partie, il s'agira de comparer les débits observés dans une évaluation des performances d'encodage et de décodage des deux implémentations de notre code à effacement Mojette et des codes de Reed-Solomon. En particulier pour le code Mojette, nous évaluerons la version non-systématique, que l'on appellera *NS-Mojette*, ainsi que l'implémentation systématique que l'on désignera simplement par *Mojette*.

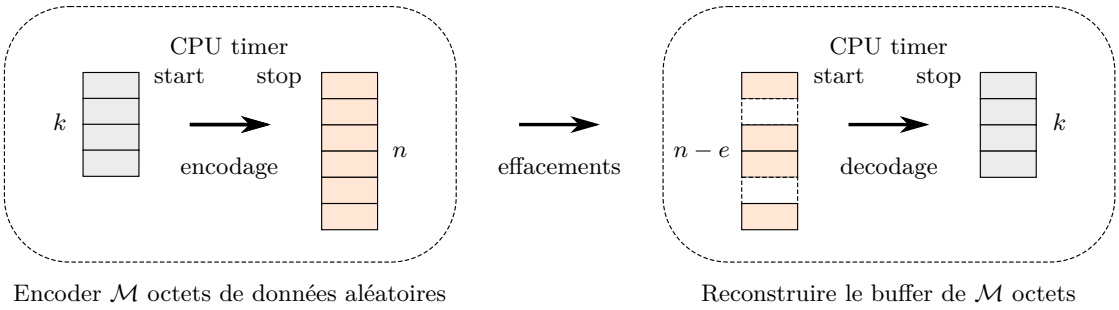


FIGURE 4.6 – Représentation de l'expérimentation. Les données aléatoires sont stockées dans un *buffer* indexé par k adresses. Le décodage est réalisé après avoir supprimé $e \times \frac{\mathcal{M}}{k}$ octets. On enregistre le compteur de cycles avant et après la fonction d'encodage et de décodage pour mesurer les performances.

Implémentation des codes Reed-Solomon

ISA-L (pour INTEL *Storage Acceleration Library*) est une bibliothèque *open-source* développée par INTEL® CORPORATION [Int15] fournissant une implémentation des codes de REED-SOLOMON. Cette bibliothèque fournit des codes optimisés pour les applications de stockage. Dans la limite de notre expérience orientée dans le contexte du stockage, et des paramètres utilisés dans notre expérimentation, cette implémentation fournit de meilleurs résultats que d'autres bibliothèques fournissant des codes pour de la transmission, telles que *OpenFEC.org* et JERASURE [Ope10 ; PG14]. En particulier, cette implémentation des codes de REED-SOLOMON utilise le polynôme irréductible $x^8 + x^4 + x^3 + x^2 + 1$.

4.3.2 Configuration de l'expérimentation

Dans cette partie, nous allons évaluer les performances d'encodage et de décodage des implémentations des codes à effacement Mojette et REED-SOLOMON, présentées précédemment.

Les tests réalisés dans cette partie mettent en jeu plusieurs paramètres. Ainsi nous allons faire varier les paramètres n et k des codes à effacement, qui définissent implicitement la tolérance aux pannes que fournit le code. En pratique, ce facteur dépend de la nature des données, des applications et du support sur lesquels transite la donnée. Les fournisseurs de service web proposent en général une protection face à quatre pannes. C'est le cas de FACEBOOK, qui utilise des codes de REED-SOLOMON au sein de leurs grappes de stockage [Sat+13]. Un second paramètre concerne la taille des données $\mathcal{M}$ que nous allons traiter. Dans la terminologie Mojette, cette taille correspond au nombre de pixels de la grille discrète. Ce paramètre dépend de l'application utilisée. Dans le cadre de stockage de données POSIX, on choisira une taille $\mathcal{M}$ correspondant à la taille des blocs du système de fichiers. Dans l'exemple d'*ext4*, cette taille de blocs est de 4 Ko. En revanche, dans des applications mettant en jeu des accès séquentiels sur de grands fichiers, on choisira une taille de bloc beaucoup plus importante afin de limiter le nombre d'entrées/sorties. C'est le cas du système de fichiers *Hadoop Distributed File Systems*

HDFS, qui met en jeu des applications d'analyse distribuées grâce à *Hadoop Map-Reduce* sur des blocs de 128 Mo par défaut [Shv+10].

La figure 4.6 représente l'expérimentation que l'on réalise. Notre mesure lors de l'encodage correspond aux performances du CPU lors de la génération de n blocs encodés à partir de k blocs de données. Ces k blocs totalisent $\mathcal{M}$ octets. Plus particulièrement dans notre mise en œuvre, ces k blocs correspondent à une zone mémoire de $\mathcal{M}$ octets de données aléatoires, dont on représente chaque bloc par k pointeurs vers l'adresse de début de ces blocs. L'encodage non-systématique consiste alors à générer n blocs de données encodés à partir de ces données d'entrées. En revanche, pour les versions systématiques, l'encodage correspond aux opérations suivantes : (i) copie des k blocs de données ; (ii) puis génération de $(n - k)$ blocs de parité. Le critère de comparaison de performance entre les différents codes correspond donc à l'effort du CPU pour offrir une certaine tolérance aux pannes.

Concernant le décodage, les performances enregistrées correspondent à la reconstruction des k blocs de données. Un nouveau paramètre entre en jeu dans les opérations de décodage puisque le schéma de perte influence les performances du code à effacement. En conséquence, nous enregistrons les performances du CPU pendant le décodage tout en augmentant le nombre de pannes jusqu'à la tolérance limite qu'offre le code. Dans notre cas, une panne correspond à l'absence d'information dans un bloc de données si le code est systématique, sinon il s'agira de la perte d'un bloc encodé.

Nos tests sont exécutés sur une seule machine, un seul processeur et un seul *thread*. Toutes les opérations sont réalisées en mémoire, en prenant soin de ne pas créer d'interactions avec le disque dur. Plus exactement, nous ne considérons pas certains pré-traitements tels que la génération des matrices d'encodage dans le cas des codes de Reed-Solomon, ou la détermination du chemin de reconstruction dans le cas des codes Mojette.

Puisque l'on mesure des fonctions d'encodage et de décodage hautement optimisées pour nos architectures processeurs dont les temps d'exécution sont de l'ordre de la nanoseconde, il est imprécis, voire impossible, de mesurer le temps d'exécution de ces fonctions de nos implémentations. En revanche, puisque ces calculs sont bornées par les considérations vues dans la partie précédentes, et puisque nos instructions sont réalisées au sein d'un *thread* sur un processeur, il est possible d'obtenir une mesure sur le nombre de cycles du processeur. Plus précisément, on utilise le compteur temporel (*Time Stamp Counter* TSC) qui est un registre spécial qui s'incrémente à chaque cycle CPU. Pour cela, on utilise l'instruction *ReaD Time Stamp Counter* (ou RDTSC) qui permet de récupérer la valeur de ce registre. Il suffit alors d'enregistrer sa valeur avant et après nos fonctions d'encodage et de décodage et d'afficher la différence. Intel propose une mise en œuvre afin de filtrer les résultats aberrants [Int98].

Enfin, nous affichons la valeur moyenne qui résulte de 100 itérations. L'écart type n'est pas présenté puisqu'il est négligeable (et correspond à moins d'un pour-cent des valeurs présentées). La machine utilisée provient de la plate-forme *FEC4Cloud* située à Polytech Nantes. Cette machine dispose d'un processeur Intel Xeon à 1, 80 GHz, de 16 Go de mémoire RAM et de caches processeurs de 128 Ko, 1 Mo et 10 Mo pour les niveaux *L1*, *L2* et *L3* respectivement. Notons que cette plate-forme a été utilisée dans

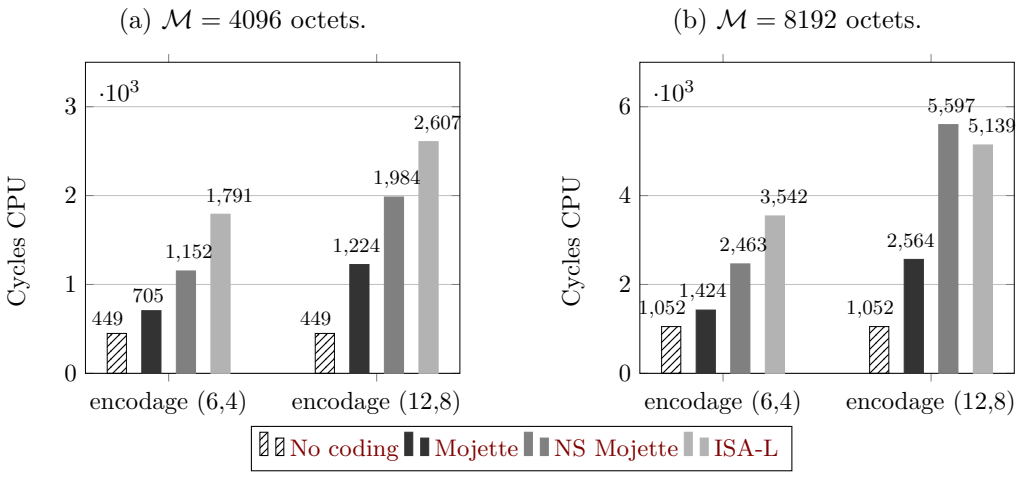


FIGURE 4.7 – Comparaison des performances d'encodage sur des blocs de données de 4 Ko (a) et 8 Ko (b). Les performances des codes Mojette et REED-SOLOMON sont comparées par le nombre de cycles CPU nécessaires pour réaliser l'opération d'encodage (plus le résultat est petit, plus il est bon). Deux paramètres de codage ont été utilisés : $(6, 4)$ et $(12, 8)$. Les performances optimales sont représentées par « no coding ».

des expérimentations soumises dans plusieurs publications [PPN14 ; Per+15a ; Par+15].

4.3.3 Résultats de l'expérimentation

Dans la suite, nous exposerons les performances obtenues pour l'encodage dans la section 4.3.3, puis pour le décodage dans la section 4.3.3. Nous analyserons en particulier l'influence de la taille des blocs et des effacements dans les sections sections 4.3.3 et 4.3.3, respectivement.

Performances à l'encodage

Nous analysons dans cette partie les résultats à l'issu de notre expérimentation sur les performances d'encodage, qui mesure le nombre de cycles CPU nécessaires pour encoder la donnée. Les figures 4.7a et 4.7b montrent les résultats obtenus pour des tailles de blocs $\mathcal{M}$ équivalent à 4096 et 8192 octets respectivement. Nous avons représenté en hachuré sur ces courbes, les performances optimales obtenues par une opération équivalente sans encodage. Plus précisément, ces performances correspondent à la copie de n blocs de données. Dans le cas où $\mathcal{M}$ vaut 4 Ko, cette opération correspond à copier 6 Ko. Pour une taille de bloc de $\mathcal{M} = 8$ Ko, les performances optimales représentées correspondent à la copie de 12 Ko de données. Dans notre expérimentation, cette opération de copie est implémentée à partir de la fonction *memcpy()* de la bibliothèque standard du C.

La première observation générale que l'on peut faire sur ces courbes d'encodage est que les performances de la Mojette non-systématique sont comparables à celles fournies par l'implémentation des codes de REED-SOLOMON d'ISA-L. Plus précisément, ces derniers

sont moins performants lors de trois tests sur quatre (la tendance s'inverse dans le test $(12, 8)$ présenté dans la figure 4.7b). En réalité, il est important de rappeler que cette implémentation du code Mojette doit calculer trois fois plus de données que les autres implémentations testées dans notre expérimentation. En effet, puisque cette version est non-systématique, elle doit calculer 12 projections Mojette dans le cas d'un code $(12, 8)$, tandis que le code de Reed-Solomon doit calculer seulement 4 blocs de parité. On constate cependant dans le cas du test de la figure 4.7a, qu'il nécessite plus de 30% de cycles supplémentaires par rapport à NS Mojette, pour protéger la donnée face à 4 pannes. On observe donc que malgré le désavantage calculatoire de notre code en version non-systématique, il parvient dans le cadre de nos tests à être compétitif face à des codes systématiques.

Une deuxième observation est que la version systématique du code Mojette est plus performante que sa version non-systématique. Ce résultat était attendu puisque cette dernière version doit calculer trois fois plus d'information lors de l'encodage. Notons cependant que la différence observée entre les résultats de ces deux implémentations n'est pas un facteur trois. Lors de nos tests d'encodage, le nombre de cycles CPU mesuré des implémentations systématiques correspond à : (i) la copie des k blocs d'informations en clair ; (ii) plus le calcul des $(n - k)$ blocs de parité. Les résultats observés correspondent donc à la somme de cette copie et de l'encodage. En revanche, si l'on prend l'exemple des résultats du code Mojette $(6, 4)$ sur des blocs de 4 Ko, présentés dans la figure 4.7a, on observe que $(705 - 321) \times 3 = 1152$, où 321 correspond aux nombres de cycles CPU nécessaires pour copier 4096 octets, et où 1152 correspond à la valeur observée dans les résultats de la version non-systématique de la même courbe. Ces résultats nous permettent donc de valider que l'encodage systématique est trois fois plus performant que l'encodage non-systématique dans le cas où nos codes sont paramétrés sur un taux $r = \frac{3}{2}$.

En conséquence, nos courbes de résultats montrent que pour les paramètres choisis dans nos expériences, l'encodage de l'implémentation non-systématique offre des performances comparables à la meilleure implémentation des codes de Reed-Solomon développée par Intel. De plus, la version systématique du code Mojette que nous avons développée offre des performances d'encodage largement supérieures à ce que proposent les autres codes utilisés dans nos tests. On observe ainsi une réduction de la latence par deux environ dans le cas de l'encodage du code Mojette par rapport aux codes de Reed-Solomon. Dans une autre mesure, les résultats atteint par notre nouvelle mise en œuvre sont proches des résultats optimaux, correspondant à la copie de l'information, sans opération d'encodage. Ceci montre que le surcout calculatoire de cette nouvelle version est particulièrement réduit.

Performances au décodage

Nous analysons dans cette partie les résultats à l'issu de notre expérimentation sur les performances de décodage en matière de cycles CPU nécessaires pour reconstruire la donnée initiale. Les figures 4.8a et 4.8b donnent le nombre de cycles CPU nécessaires pour le décodage des codes $(6, 4)$ pour des blocs de 4 Ko et 8 Ko respectivement. De manière similaire, les figures 4.8c et 4.8d concernent des codes $(12, 8)$. Nous avons représenté sur

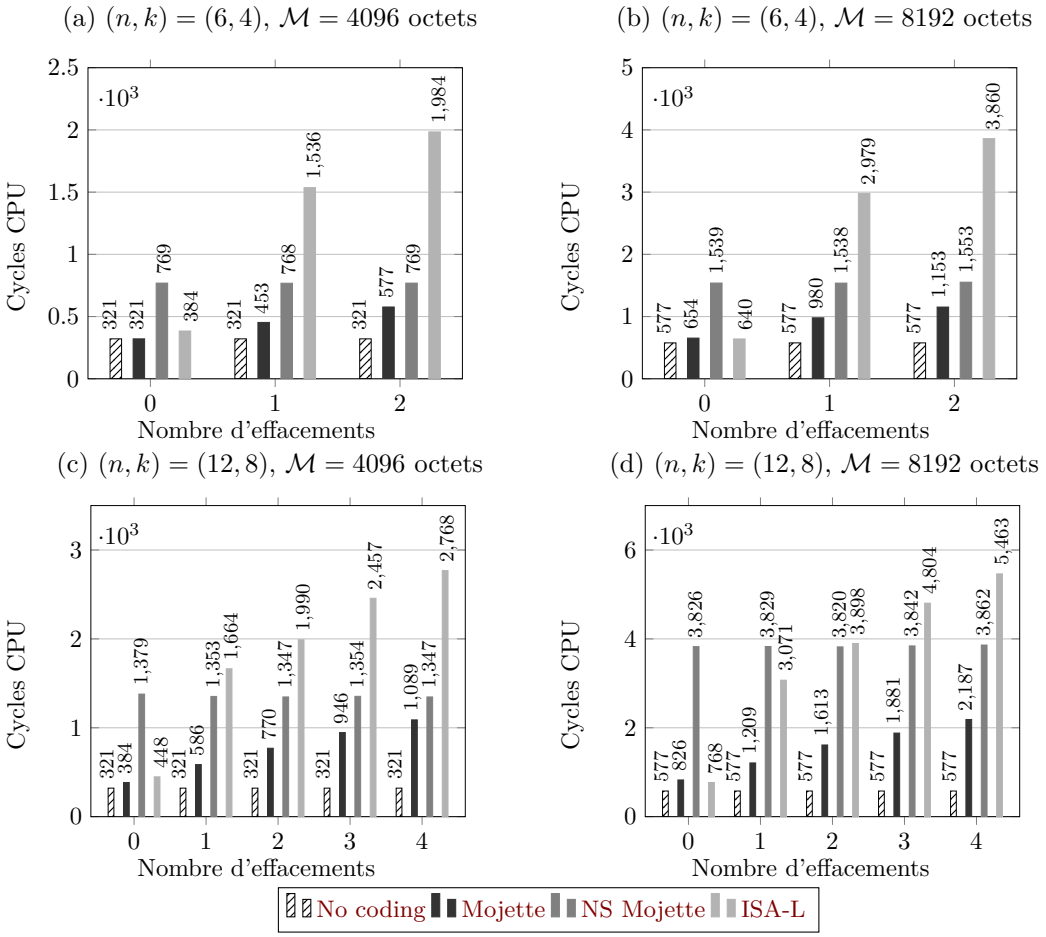


FIGURE 4.8 – Comparaison des performances de décodage pour des paramètres de codes $(6, 4)$ (figures 4.8a et 4.8b) et $(12, 8)$ (figures 4.8c et 4.8d). Les courbes à gauche montrent les résultats pour des tailles de blocs de 4 Ko (figures 4.8a et 4.8c) tandis que les courbes de droite concernent des blocs de 8 Ko (figures 4.8b et 4.8d). Les performances des codes Mojette et REED-SOLOMON sont comparées par le nombre de cycles CPU enregistrés durant l'opération de décodage (plus c'est bas, mieux c'est) alors que l'on augmente progressivement le nombre d'effacements. Un effacement correspond à la perte d'un bloc encodé. Les valeurs optimales sont représentées par « No coding ».

ces courbes les performances optimales de décodage correspondant à la copie de k blocs d'information.

La première observation générale que l'on peut faire concerne le cas où aucun effacement ne survient lors du décodage. Dans ce cas, les deux codes systématiques Mojette et Reed-Solomon atteignent les performances optimales représentées sur nos courbes par « no coding ». Ce résultat était attendu puisque dans le cas des codes systématiques, lorsqu'aucun des k blocs de données n'est effacé, le décodage correspond à la lecture direct de ces k blocs. Au niveau de l'implémentation, cette lecture correspond à la copie de cette information en clair.

À présent, lorsque des effacements surviennent, des opérations de décodage sont déclenchées. On remarque que l'influence des effacements n'est pas la même selon que le code est systématique ou non. Pour NS Mojette, le nombre d'effacements e n'a pas d'influence sur les performances de décodage. Ce résultat provient du fait que le décodage des codes non-systématiques correspond à la reconstruction entière des informations utilisateurs. Ainsi le nombre d'opérations est comparable quel que soit l'ensemble des blocs encodés utilisé pour cette reconstruction.

Dans le cas des codes systématiques en revanche, le décodage consiste à reconstruire un ensemble partiellement reconstruit de la donnée. En conséquence, le nombre de cyles CPU nécessaires pour le décodage augmente au fur et à mesure que l'on augmente le nombre de blocs de données effacés. En particulier, la différence entre les performances de l'implémentation systématique du code Mojette et des valeurs optimales augmente avec le nombre d'effacements puisque l'on supprime progressivement des lignes de la grille discrète. En effet, puisque l'on considère une grille de moins en moins remplie, et puisque les opérations d'additions nécessaires à la reconstruction Mojette sont plus coûteuses que la copie utilisée dans *memcpy()*, les performances décroissent.

Notons cependant que malgré la baisse de performances du décodage observée lorsque l'on augmente le nombre d'effacement pour le code systématique Mojette, les valeurs enregistrées sont d'une part toujours meilleures que celles observées pour la version non systématique (puisqu'il s'agit du cas où la grille doit être entièrement reconstruite). D'autre part, ces performances sont significativement meilleures que les performances observées par l'implémentation systématique des codes de Reed-Solomon. On peut ainsi observer dans nos tests, une réduction par trois de la latence de décodage en utilisant le code Mojette.

Impact de la taille des blocs

L'influence de la taille des blocs $\mathcal{M}$ est étudiée ici. Dans nos tests, nous n'utilisons que de petites tailles de blocs (i.e. $\mathcal{M}$ vaut 4 Ko ou 8 Ko), correspondant à l'application de stockage visée. Pour les valeurs de nos tests, on observe que le nombre de cycles double en même temps que la valeur de $\mathcal{M}$, pour les deux versions du code Mojette. Cette observation est la même dans le cas de l'encodage, que du décodage. Cela confirme la complexité linéaire de la transformation Mojette.

Influence de la tolérance aux pannes

Nous analysons à présent l'impact du paramétrage (n, k) du code sur les performances des codes systématiques. Bien que pour les valeurs des paramètres utilisés dans cette expérimentation, le code Mojette fournisse de meilleures performances que l'implémentation des codes de Reed-Solomon, l'écart des performances entre les deux méthodes semble diminuer à mesure que seuil maximum de tolérance aux pannes augmentent. En conséquence, il est possible que pour très grandes valeurs de paramètre, la situation s'inverse. Toutefois, comme précisé auparavant, une protection face à quatre pannes apporte déjà une protection importante [Sat+13].

Conclusion

Nous avons vu dans ce chapitre une évaluation théorique des performances du code à effacement Mojette sous sa forme systématique, dans le contexte du stockage distribué. Pour cela, nous avons défini trois métriques : (i) le nombre d'opérations nécessaires à l'encodage ; (ii) le nombre de blocs modifiés lors d'une mise à jour ; (iii) le nombre d'opérations nécessaires au décodage. En particulier, nous avons montré que le code à effacement Mojette nécessite moins d'opérations dans ces trois métriques, comparé aux *Array* codes et codes de Reed-Solomon, dans un contexte RAID-6 où la tolérance aux pannes est limitée à deux (i.e. paramètres de code $(k + 2, k)$). Par la suite, nous avons étendu le même constat pour des paramètres de code (n, k) arbitraires, en comparant les codes Mojette et Reed-Solomon. Dans une dernière section, nous avons évalué par la pratique les performances de notre implémentation. En particulier, notre expérimentation a permis de montrer le gain significatif de notre nouvelle mise en œuvre du code systématique par rapport à la version non-systématique. Il est intéressant de remarquer toutefois, que malgré ce gain, le code sous sa forme non-systématique parvient tout à fait à fournir de bonnes performances. De plus, dans le cadre de notre expérimentation, notre implémentation encode deux fois vite, et décode jusque trois fois plus vite que la meilleure implémentation des codes de Reed-Solomon actuelle, qui est contenue dans la bibliothèque développée par Intel® Corporation [Int15].

Rappelons cependant que les bonnes performances obtenues par le code à effacement Mojette nécessitent davantage d'information encodée que ce qui est produit dans le cas des codes MDS. Toutefois, nous avons montré dans le chapitre 3 que ce coût est modéré, et tend vers la borne minimale quand la taille des blocs augmente. Plus particulièrement, nous y avons analysé qu'en définissant des blocs de $\mathcal{M} = 4$ Ko, comme utilisés dans nos tests, un surcout de données de seulement 3% est nécessaire. N'ayant pas d'impact significatif sur un système de stockage, ce faible surcout est largement contrebalancé par l'amélioration des performances que nous avons observée ici.

Ce chapitre a permis de mettre en avant le fait que le code à effacement Mojette (systématique ou non-systématique) est suffisamment efficace pour ne pas former un goulot d'étranglement dans la chaîne de transmission des données. Dans le chapitre suivant, nous allons nous intéresser à l'intégration de ce code au sein d'un système de

fichiers distribué (DFS). Cette intégration vise à ce que le DFS puisse tirer parti du code à effacement pour protéger les données face aux pannes, tout en fournissant de bonnes performances.

5

Application au système de fichiers distribué RozoFS

Sommaire

Introduction du chapitre

Les systèmes de fichiers distribués (DFS) permettent d'agréger en un volume unique différents supports de stockage interconnectés par un réseau. Plusieurs processus exécutés sur des serveurs de ce réseau peuvent ainsi accéder simultanément à ce volume afin de partager et d'interagir avec les données qui y sont stockées. Comme nous l'avons énoncé précédemment, ces systèmes reposent sur la redondance d'information afin de supporter l'indisponibilité des données. Dans la pratique, l'apparition de pannes sur de tels systèmes est considérée comme une norme et non une exception [Wei+06]. De par leur simplicité de mise en œuvre, les techniques de réplication sont largement utilisées pour fournir cette redondance. Cependant, ces techniques impliquent de stocker une quantité importante de redondance par rapport à la donnée à protéger (e.g. 200% dans le cas de la réplication par trois). Les codes à effacement offrent une alternative permettant de fournir la même tolérance aux pannes, tout en diminuant très largement cette quantité de redondance (généralement par un facteur 2) [WK02 ; OD12]. En conséquence, la transition vers un système de données encodées réduit significativement la consommation énergétique du système de stockage.

À l'origine, les systèmes RAID-5 tolèrent la perte d'un disque, moyennant un volume de données qui reste moins important par rapport au volume induit par les données répliquées dans le cas de RAID-1. Côté performance, le calcul des données de parité est une opération relativement simple pour le contrôleur RAID. Les additions utilisées ont en effet un impact modéré sur les performances du système en écriture, à condition que ces données soient distribuées sur l'ensemble des disques de la matrice. Cette considération permet en effet de répartir la charge afin qu'un disque ne forme un goulot d'étranglement. Toutefois, cette technique se retrouve rapidement limitée à mesure que la taille des matrices de disques augmente. En effet, plus le nombre de disques augmente, plus le risque qu'une double panne survienne est important. La famille RAID s'est alors agrandie grâce à de nouvelles techniques permettant une protection face à la perte d'un second disque (RAID-6). Pour un ensemble de disques donné, cette technique améliore la tolérance aux pannes du système, au prix d'une diminution de la capacité de stockage. Une première construction repose sur les codes de « Reed-Solomon $\mathcal{P} + \mathcal{Q}$ » [Che+94]. Le principal problème de cette méthode correspond à la complexité des calculs des blocs de parité, qui impacte significativement les performances en écriture et lors des opérations de reconstruction. En conséquence, plusieurs méthodes de codage ont été proposées pour améliorer cette complexité, telles que l'utilisation combinée de parités horizontales et verticales [Gib+89], les codes EVENODD [Bla+95] d'IBM, les codes RDP de NetApp [Cor+04], ou encore des codes à densité minimale [BR99]. Le milieu scientifique s'est donc intéressé à l'efficacité des implémentations de ces codes [Pla+09]. Par la suite, les systèmes de stockage ont évolué en tirant parti d'un ensemble de machines interconnectées. Des techniques de distribution RAID sont alors apparues en exploitant la communication à travers un réseau, tout d'abord au niveau des blocs [LMC94], puis au niveau logiciel sous le terme de *Reliable Array of Independant Nodes* (RAIN) [Boh+01]. Apparaissent également des mises en œuvre dans les infrastructures pair à pair [RD01 ; WK02]. Aujourd'hui, grâce

aux codes à effacement, ces techniques sont largement étendues dans les systèmes de fichiers distribués tels que HDFS [Fan+09], Ceph [Wei+06] ou Tahoe [WW08].

Nous débuterons ce chapitre par une étude détaillée des systèmes de fichiers distribués en section 5.1. La section 5.2 présentera en détail RozoFS : le DFS qui intègre le code à effacement Mojette. Puis nous mesurerons les performances en matière de lecture et d'écriture de RozoFS dans une évaluation réalisée dans la section 5.3.

5.1 Systèmes de fichiers distribués tolérants aux pannes

Dans cette section, nous allons étudier les systèmes de fichiers distribués qui proposent des mécanismes permettant de supporter l'inaccessibilité d'une partie de la donnée. Il existe plusieurs façons de concevoir un système de fichiers distribué. Pour comprendre cela, nous verrons dans la section 5.1.1 qu'il existe plusieurs architectures pour partager des données, allant d'une conception centralisée à un schéma complètement décentralisé. Par la suite, nous présenterons dans la section 5.1.2, les principales fonctions qui rentrent en jeu dans la conception d'un DFS (e.g. la gestion de la cohérence des données). La section 5.1.3 se focalisera quant à elle aux aspects de redondance dans ce genre de système.

5.1.1 Architectures

Nous allons ici explorer les différentes architectures utilisées dans le partage des données entre plusieurs clients. Nous verrons tout d'abord le modèle client-serveur, utilisé notamment dans NFS. Par la suite nous verrons une solution dont la distribution des données sur une grappe de serveur est gérée par un serveur de métadonnées. Enfin, nous verrons une architecture entièrement décentralisée.

Architectures client-serveur

Plusieurs DFS sont définis sur un modèle client-serveur. C'est notamment le cas de *Network File System* (NFS) qui est le plus utilisé par les systèmes basés sur UNIX [HN15]. Le principe de ces DFS est que le serveur offre une vision uniformisée d'une partie de son système de fichiers local (quel que soit le type de ce système de fichiers). NFS dispose d'un protocole de communication qui permet aux clients d'accéder aux fichiers du serveur. Plus précisément, le client ne sait pas comment sont implémentées les opérations pour interagir sur le système de fichiers du serveur distant. En revanche, le serveur, lui, propose une interface accessible par des requêtes *Remote Procedure Calls* (RPC), pour réaliser ces opérations. Et le serveur, qui maîtrise la façon dont ces opérations sont implémentées, les applique. Ainsi, il est possible que des processus installés sur différentes machines, fonctionnant sur différents systèmes d'exploitation, puissent interagir avec un système de fichiers virtuel partagé.

Côté client, NFS permet de monter le système de fichiers distant. Comme le représente la figure 5.1, cette opération est possible grâce au *Virtual File System* (VFS) qui remplace l'interface du système de fichiers local afin d'interfacer plusieurs systèmes de

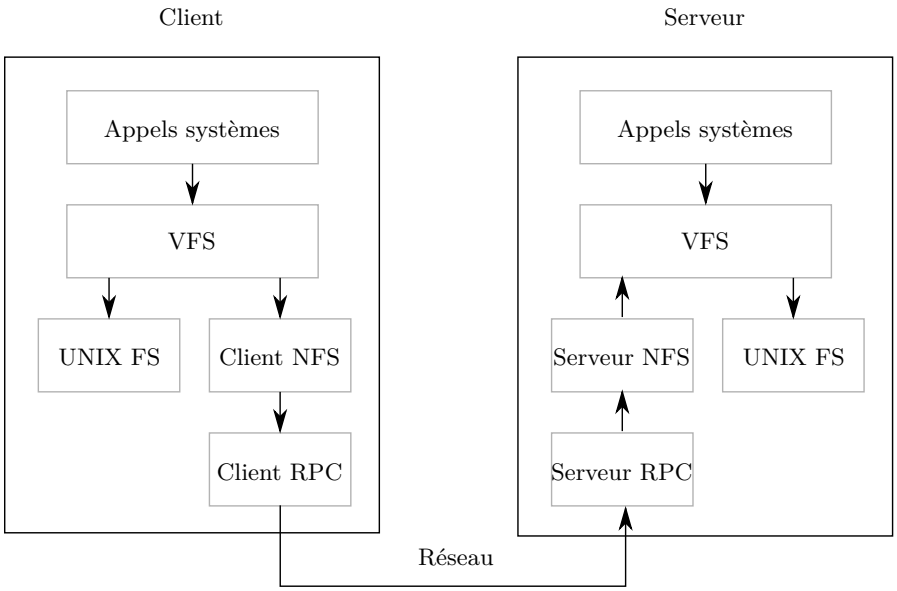


FIGURE 5.1 – Représentation d'une architecture client-serveur, utilisée notamment dans NFS.

fichiers [Kle86]. En particulier, il intercepte les appels systèmes pour les transmettre au client NFS.

Architectures en grappe centralisées

Les architectures en grappe sont une extension du modèle client-serveur. Les grappes de serveurs sont généralement utilisées dans le cas d'applications parallèles. Dans ce modèle, les fichiers sont distribués sur un ensemble de nœuds de la grappe. Puisque plusieurs éléments disposent de la donnée, il est alors possible de distribuer également les applications. Dans cette approche, proposée par GOOGLE pour son *Google File System* (GFS) [GGL03], chaque grappe correspond à un nœud maître et plusieurs nœuds de stockage. La figure 5.2 illustre cette approche. Le nœud maître maintient et transmet les informations liées aux métadonnées d'un fichier. Pour y accéder, un client doit lui fournir l'identifiant du fichier afin qu'il lui transmette les informations permettant d'atteindre les données relatives à ce fichier, qui sont réparties sur les serveurs de stockage.

Architectures en grappe complètement distribuées

Le modèle précédent n'est pas adapté pour certains types d'infrastructures tels qu'en pair à pair. Il existe des moyens pour ne pas avoir à garder un index contenu dans un nœud *master*. Pour cela, on combine un mécanisme clé-valeur avec un système permettant de calculer de façon unique la position des données dans une grappe. Il est par exemple possible d'utiliser le protocole Chord pour déterminer de manière décentralisée la position des données dans un anneau [Sto+01]. Cette technique est par exemple utilisée dans le

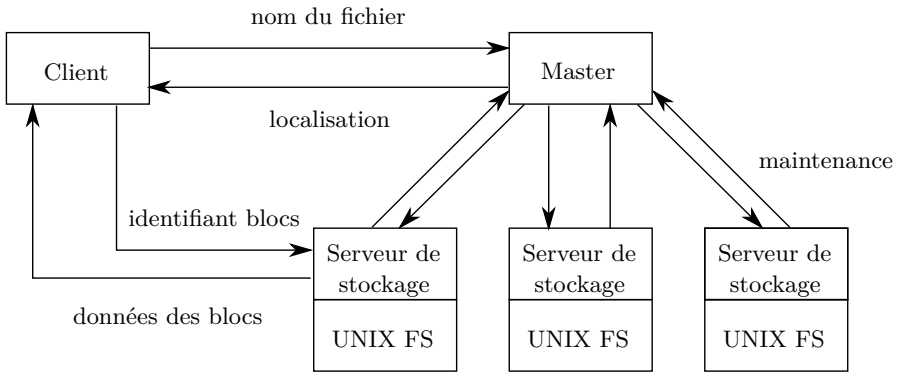


FIGURE 5.2 – Représentation d'une architecture en grappe centralisée comme utilisée dans GFS. Un client contacte le nœud *master* afin de déterminer la position des blocs d'un fichier. Il contacte ensuite les serveurs de stockage appropriés pour récupérer l'information voulue.

système de fichiers pair à pair Ivy [Mut+02]. D'autres méthodes existent telles que, la distribution des données issue de l'algorithme CRUSH utilisé dans Ceph [Wei+06].

5.1.2 Principes de conception

Cette section permet de comprendre les enjeux d'un système de fichiers distribué. En particulier, nous étudierons une liste non exhaustive d'éléments à prendre en compte lors de la conception d'un DFS, telles que la gestion de la distribution des données, ou de la cohérence des données. Bien que les mécanismes de tolérance aux pannes puissent correspondre à cette étude, ils seront plus précisément décrits dans la section suivante.

Transparence des opérations

L'utilité d'un système de fichiers distribué est de pouvoir interagir avec un volume de stockage distribué sur un ensemble de supports de stockage. Du point de vue fonctionnel, la mise en œuvre doit permettre à l'utilisateur d'avoir l'impression d'interagir avec un système de fichiers UNIX local. C'est le rôle du VFS que d'intercepter les appels systèmes depuis les applications et de les fournir à la couche du DFS, afin que celui-ci applique les opérations nécessaires sur l'ensemble des supports de stockage en jeu.

Espace de nommage

La capacité à résoudre la correspondance entre les éléments de l'organisation hiérarchique proposée aux client, et les données distribuées au sein du système est relative à l'espace de nommage. Lorsqu'un client monte un système de fichier, il dispose d'une arborescence de fichiers et de répertoires dans laquelle il peut naviguer. Dans un système de fichiers UNIX classique, un fichier est visible par un utilisateur comme un élément inclus au sein d'un répertoire, et identifié par un nom. Quand le VFS recherche ce nom, celui-ci inclut

le chemin depuis la racine pour y accéder, sauf si le chemin est relatif. Cette recherche de nom permet de déterminer le numéro d'inode qui est un identifiant unique utilisé par le système de fichiers. Une fois que ce numéro est déterminé, il est possible d'accéder à la structure de l'inode, et donc aux données du fichier.

Dans le cas des DFS, le montage d'un système de fichiers permet de placer le volume distant dans l'arborescence locale (le volume est dit *exporté*). Par exemple dans NFS, le serveur peut définir d'exporter un répertoire de son système de fichiers local. Un espace de nommage supplémentaire est alors nécessaire afin de réaliser la correspondance du nom d'un fichier proposé à l'utilisateur dans le volume exporté, avec la position des données sur l'ensemble des nœuds de stockage de la grappe. Dans le cas de GFS, c'est le serveur maître qui réalise cette correspondance.

Cohérence

Dans un système UNIX classique, lorsqu'une demande de lecture survient après deux écritures successives sur une même partie de la donnée, il est garanti que le système retourne la donnée qui résulte des opérations d'écriture successives.

Dans un système distribué, plusieurs problèmes surviennent. Tout d'abord le délai nécessaire pour que la donnée transite d'un serveur vers un client peut être suffisant pour délivrer une version dépassée. De plus, pour des raisons de performance, les clients tendent à utiliser des techniques de cache afin de garder une version des données localement, mais qui implique une complexité de gestion des ressources partagées. Il est possible de contourner ce deuxième problème de deux manières. La première solution consiste à mettre à jour le serveur à chaque modification du cache, ce qui n'est pas performant. La seconde consiste à relaxer la contrainte de cohérence en affirmant que dans un premier temps, les modifications ne seront visibles qu'à partir du client, mais qu'à terme (par exemple lors de la fermeture du fichier) ces modifications seront accessibles à tous. Bien que cela ne résout pas le problème de partage des ressources, on considère dans ce cas que la lecture d'une version dépassée est valide.

Une autre solution pour résoudre ce problème est de rendre les fichiers immuables. C'est le cas des premières versions de HDFS puisque ce fonctionnement est cohérent avec les traitements *Map-Reduce* qui lisent des fichiers en entrée et écrivent les résultats dans de nouveaux fichiers. Une dernière façon de gérer le partage de données est d'utiliser des transactions atomiques.

5.1.3 Redondance dans les DFS

Dans cette section, nous nous intéresserons à la gestion de la redondance dans les DFS afin de supporter l'inaccessibilité d'une partie de la donnée. Pour cela nous chercherons à estimer la disponibilité d'une donnée dans un système de stockage en fonction de la technique utilisée (réplication ou codage à effacement). Par la suite, nous donnerons un état de l'art de l'utilisation de ces techniques dans les DFS.

Disponibilité de la donnée

L'utilisation des techniques de réplication permet de garantir la disponibilité des données d'un système de stockage en conservant plusieurs copies distribuées sur différents supports de stockage. Il faut toutefois relativiser la disponibilité de cette donnée puisque les probabilités qu'une panne survienne dans un système composé d'une grappe importante de serveurs sont significatives.

Soit ρ_F la probabilité qu'un disque devienne inaccessible. Une estimation de la valeur de ρ_F dépend du *Mean Time Between Failures* (MTBF), c'est-à-dire le temps moyen qui s'écoule entre deux pannes, ainsi que le *Mean Time To Repair* (MTTR) qui correspond au temps nécessaire pour remplacer le disque. Par exemple, pour un disque qui fonctionne pendant 1000 jours, et qui peut être remplacé, formaté et fonctionnel en un jour, $\rho_F = 0.001$ [CPK14].

Considérons un système de stockage qui réplique chaque objet à stocker d'un facteur n. Ce système peut perdre de la donnée lorsque l'ensemble de ces n disques tombent en panne. En supposant que les pannes correspondent à des phénomènes *indépendants*, la probabilité de perdre de la donnée ρ_P vaut ρ_F^n. En comparaison, dans un système utilisant un code à effacement (n, k), on perd de la donnée lorsque plus de k disques tombent en panne. Par conséquent, la probabilité de perdre de la donnée vaut :

$$\rho_P = \sum_{i=n-k+1}^{n} \binom{n}{i} \rho_F^i (1-\rho_F)^{n-i}. \tag{5.1}$$

En conclusion, pour une une même disponibilité, un code à effacement permet à un système de stockage de disposer d'une capacité de stockage plus importante qu'en utilisant de la réplication.

Gestion de la redondance dans les DFS

La réplication par trois est configurée par défaut dans la plupart des systèmes de stockage tel que *Hadoop Distributed File System* (HDFS) [Shv+10], ou encore *Google File System* (GFS) [GGL03]. Cependant, le facteur de réplication entraîne une augmentation significative des volumes de ces systèmes de stockage. C'est pourquoi, la réduction de la quantité de redondance est devenue une préoccupation importante dans le milieu scientifique et industriel.

HDFS est l'une des solutions de stockage les plus populaires. Cette solution offre un système de fichiers extensible et tolérant aux pannes. Le modèle *Map-Reduce* utilise HDFS afin de diviser et distribuer les tâches sur les différents nœuds de travail [DG08]. En particulier, ce système de fichiers est conçu pour accomplir des tâches d'analyse en parallèle sur de très importants volumes de données immuables (i.e approche *write-once-read-many*). Par exemple, les plus grosses grappes de nœuds expérimentées reposent sur 4000 serveurs qui agrègent une capacité totale de $14,25$ pétaoctets exploitée par 14000 clients simultanément [SM08]. Côté performances, l'exploitation du parallélisme par *Map-Reduce* permet de trier un téraoctet de données en une minute [OM09]. HDFS est adapté pour ce genre d'applications puisque ce système de fichiers travaille de manière

séquentielle sur d'importants blocs de données de 128 Mo (valeur fixée par défaut). Un bloc correspond à la plus petite quantité de données sur laquelle un système de fichiers peut lire ou écrire. L'ensemble des opérations qu'il réalise est effectué sur une quantité de données multiples de la taille d'un bloc. Cette taille a donc une influence sur le système de stockage. L'utilisation de grands blocs entraîne trois conséquences : (i) la réduction des interactions entre les clients et le serveur de métadonnées ; (ii) la réduction de la quantité de métadonnées stockées ; (iii) la réduction du trafic réseau en gardant des connexions TCP persistantes [GGL03]. Globalement, des blocs de tailles importantes permettent d'augmenter les performances de lecture et d'écriture dans le cas où de larges fichiers sont parcourus de manière séquentielle (e.g. lecture d'une vidéo de haute qualité). En effet, le support de stockage peut lire ou écrire la donnée de manière continue, sans avoir à localiser le bloc suivant. Cependant, une part significative du bloc peut être perdue dans le cas où le fichier est plus petit que la taille d'un bloc. En comparaison, les systèmes de fichiers compatibles POSIX[1] reposent sur de petites tailles de blocs (e.g. 4 Ko dans le cas de *ext4* [Mat+07]). Dans la suite, nous nous intéresserons à la conception d'un système de fichiers distribué compatible POSIX qui travaillera sur des blocs de l'ordre de 4-8 Ko comme compromis entre la capacité d'accéder de manière séquentielle aux données, et le gaspillage d'espace au sein des blocs [Gia98, p. 34].

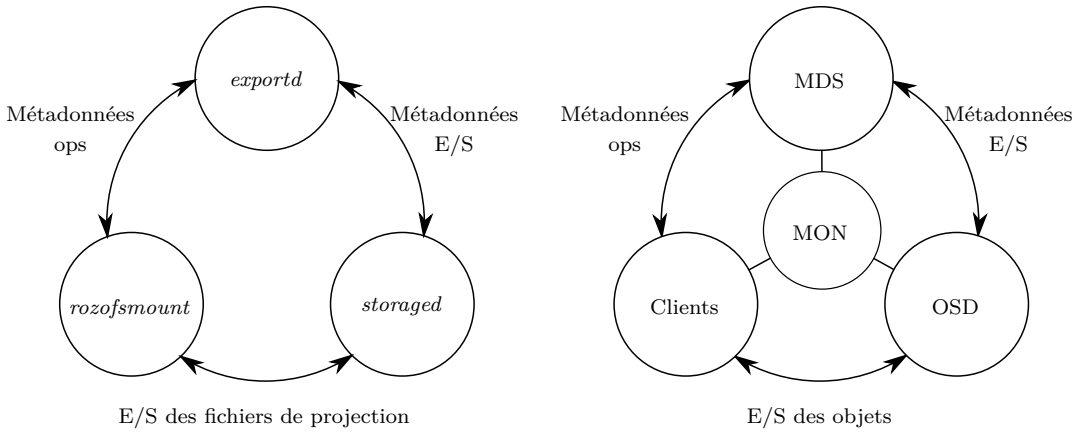
Alors que la réplication par trois est le paramètre de haute disponibilité par défaut dans HDFS, Fan et al. [Fan+09] ont développé une version modifiée basée sur les codes de Reed-Solomon, dénommée *DiskReduce*. Dans leur publication, l'utilisation des codes à effacement permet de réduire de 2, 4 fois le volume de stockage. Cependant, cette technique est mise en œuvre en tâche de fond, et permet d'encoder des blocs d'information de 64 Mo issus des répliquas, ce qui n'est pas adapté dans le cas des applications compatibles avec POSIX. Muralidhar et al. [Mur+14] ont également utilisé les codes de Reed-Solomon au sein des grappes de stockage de Facebook pour des données considérées « tièdes » (i.e. 80 lectures par seconde). GlusterFS[2] et Ceph [Wei+06] sont d'autres exemples de systèmes de stockage distribués populaire. Ceph a la particularité de stocker les données sous la forme d'objets. Toutefois, il dispose d'un système de fichiers, nommé CephFS (pour *Ceph File System*), permettant de créer une interface POSIX entre les applications et les volumes de stockage objet. Par rapport à HDFS qui est spécialisé pour des applications spécifiques, CephFS et GlusterFS sont destinés à toute application. Ces DFS fournissent des techniques de tolérance aux pannes, dont la réplication est la technique proposée par défaut. Le codage à effacement, bien que disponible, n'est en revanche recommandé que dans le cas d'applications liées aux données froides, telles que l'archivage.

5.2 RozoFS : le DFS basé sur le code à effacement Mojette

Dans cette section, nous présenterons la conception des systèmes de fichiers distribués RozoFS et CephFS. Ces deux systèmes sont à « code source ouvert » et compatibles

[1] POSIX est le standard des interfaces de programmation utilisés par les logiciels conçus pour les systèmes de type UNIX.

[2] http ://www.gluster.org/



(a) Architecture de RozoFS (b) Architecture de CephFS

FIGURE 5.3 – Représentations des architectures de RozoFS (a) et CephFS (b). Les similitudes correspondent aux : serveur de métadonnées (*exportd* et MDS), serveur de stockage (*storaged* et OSD) et aux clients (*rozofsmount*). De plus, CephFS utilise un service de monitorage (MON). Alors que les serveurs de stockage de RozoFS stockent des projections Mojette, les OSD contiennent des objets pour CephFS.

avec POSIX. En particulier, ils reposent sur un ensemble de systè.es de fichiers locaux sous-jacents. Une fois montés, ils fournissent un espace de noms global (i.e. un ensemble de répertoires et fichiers structurés comme une arborescence) qui repose sur un ensemble de nœuds de stockage standards (*commodity*). Alors que traditionnellement, CephFS repose sur des techniques de réplication, la spécificité de RozoFS est d'être conçu sur la base du code à effacement Mojette. La figure 5.3 montre les similitudes entre l'architecture de RozoFS en figure 5.3a, et de CephFS en figure 5.3b, et met en avant la correspondance des noms entre les différents services. Dans la suite, nous décrirons les composants de RozoFS et leurs interactions, avant d'analyser les principales différences avec CephFS.

5.2.1 L'architecture de RozoFS

L'architecture de RozoFS est bâtie de trois composants qui sont représentés dans la figure 5.3a :

1. le service de gestion des métadonnées, appelé *exportd* ;
2. le service de stockage des projections, appelé *storaged* ;
3. les clients qui utilisent le processus *rozofsmount* pour monter RozoFS.

Bien qu'il soit possible de combiner différents services au sein d'un même serveur, nous décrirons dans la suite ces différents composants de telle manière qu'ils soient distribués sur différents serveurs.

Layout	k	n	nœuds de stockage	capacité de correction
0	2	3	4	1
1	4	6	8	2
2	8	12	16	4

TABLE 5.1 – Caractéristiques des trois *layouts* fournis par RozoFS. Les paramètres du code à effacement Mojette sont donnés par (n, k). Le nombre minimum de supports de stockage correspond à $n + (n - k)$ afin de garantir une certaine capacité de stockage durant les opérations d'écriture.

Serveur de métadonnées Ce serveur gère le processus *exportd* qui stocke les métadonnées, et gère les différentes opérations qui sont liées. Les métadonnées correspondent à une petite quantité d'information qui décrit la donnée elle-même. Elles sont composées d'attributs POSIX (e.g. horodatages des fichiers, permissions, ...), auxquelles s'ajoutent des attributs étendus définis par RozoFS, tels que l'identification ou la localisation des fichiers sur le système de stockage. Ce serveur garde des statistiques sur la capacité des nœuds de stockage afin de répartir la charge sur les différents disques. En particulier, dans le cas d'une demande de lecture ou d'écriture, il fournit au client la liste des supports de stockage relatifs aux informations d'un fichier. La haute disponibilité de ce service n'est pour l'instant pas gérée par RozoFS. En pratique, des logiciels tiers tels que DRBD[3] (pour la réplication du service) et Pacemaker[4] (pour la haute disponibilité) sont utilisés.

Nœud de stockage Cette machine gère le service *storaged*. Deux fonctions sont gérées par ce service : (i) la gestion des requêtes vers *exportd* ; (ii) les *threads* d'E/S des données qui gèrent les requêtes issues des clients en parallèle. De plus, ce service est en charge de la réparation des nœuds de stockage. Lorsqu'un nœud est définitivement perdu, il est nécessaire de reconstruire de la redondance afin de rétablir la tolérance aux pannes du système. Nous verrons plus en détail cette considération dans le chapitre 6.

Les clients RozoFS Un client utilise le processus *rozofsmount* afin de monter localement un volume défini par RozoFS. Ce processus utilise FUSE[5] (*Filesystem in UserSpacE*) pour intercepter les appels systèmes et définir les opérations distribuées de RozoFS. Les clients gèrent deux types d'opération : (i) les opérations de métadonnées (*lookup*, *getattr*, ...) en interaction avec *exportd* ; (ii) les opérations d'E/S des données avec *storaged*. Les clients sont responsables de l'encodage et du décodage lors des opérations d'écriture et de lecture respectivement.

5.2.2 Distribution de l'information

La distribution des données sur plusieurs supports de stockage est définie par les paramètres de l'encodage Mojette. Tel qu'on a pu le voir dans les chapitres précédents, un sous-ensemble de k parmi les n blocs encodés est suffisant pour reconstruire l'information initiale. Les paramètres de protection, appelés *layouts* dans RozoFS, définissent les valeurs de k, de n, ainsi qu'un nombre minimal de supports de stockage requis afin de fournir une capacité de correction face aux pannes (en lecture comme en écriture).

Actuellement, trois *layouts* sont définis dans RozoFS. Le tableau 5.1 représente les informations relatives à ces trois configurations. Chaque *layout* correspond à un taux de codage de $r = \frac{3}{2}$. En conséquence, le volume de données stockées vaut un peu plus de $1,5$ fois la taille du volume d'entrée. Lors d'une opération d'écriture, au moins $(n - k)$ nœuds de secours sont nécessaires en plus des n supports désignés pour recevoir la donnée, afin de supporter les éventuelles pannes durant l'opération. Par exemple, en *layout* 2, le nombre minimum de supports de stockage correspond à $12 + (12 - 8) = 16$ afin de supporter l'indisponibilité de quatre nœuds durant l'opération d'écriture.

5.2.3 Des entrées/sorties tolérantes aux pannes

Cette section s'intéresse aux interactions entre les différents composants de RozoFS lors des opérations d'écriture, et de lecture. Nous verrons ensuite quel est l'impact les pannes sur le fonctionnement du système.

Les opérations d'écriture

Lorsqu'un client initialise une opération d'écriture, cela déclenche un processus d'encodage Mojette. Plus particulièrement, le fichier à écrire est découpé en blocs de $\mathcal{M} = 4$ Ko (valeur par défaut). Ces blocs de données remplissent un *buffer* adressé par k pointeurs, qui représentent les k lignes d'une grille Mojette. On utilise des *threads*, qui permettent de gérer plusieurs opérations d'encodage et de décodage en parallèle. Puisque l'on considère le code à effacement Mojette sous sa forme systématique, chaque *thread* du client calcule $(n - k)$ projections et transmet les n blocs encodés sur n supports de stockage. L'identification des supports concernés est fourni par *exportd*. Prenons l'exemple de l'écriture d'un gigaoctet d'information sur un volume RozoFS défini en *layout* 0, comme illustré dans la figure 5.4a. L'écriture distribue trois fichiers contenant les informations de projections. Plus précisément, chacun de ces fichiers fait environ 500 Mo tel que d_1 et d_2 contiennent les données en clair, et p_1 contient les données de la projection suivant la direction $(0, 1)$.

Afin d'étendre la structure des inodes à la structure de RozoFS, des métadonnées spécifiques sont allouées à chaque fichier sous la forme d'attributs étendus. Parmi ces métadonnées, on trouve notamment l'identifiant unique du fichier (l'équivalent du numéro

[3] http ://www.drbd.org/
[4] http ://clusterlabs.org/
[5] http ://www.fuse.sourceforge.net

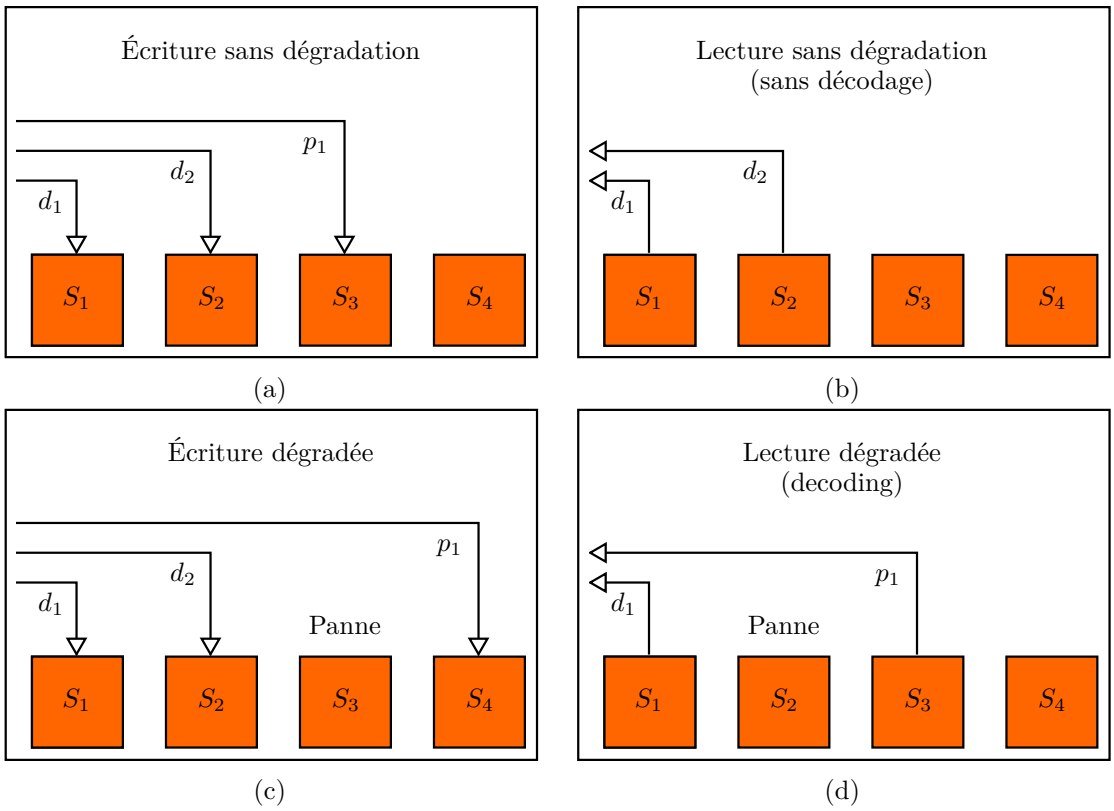


FIGURE 5.4 – Scénarios d'écriture et de lecture en *layout* 0 (i.e. utilisant le code Mojette systématique $(3, 2)$). Les figures 5.4a et 5.4b concernent les situations sans dégradation, alors que les figures 5.4c et 5.4d concernent des opérations dégradées. Les blocs de données, identifiés par d_i, et les projections Mojette, identifiées par p_j, sont distribués sur les supports de stockage S_k. Cette figure est inspirée de [Fan+09].

d'inode pour l'espace de noms de RozoFS) ainsi que les serveurs de stockage sur lesquels sont distribuées les données.

Les opérations de lecture

Lorsqu'une application demande la lecture d'une donnée présente sur le point de montage de RozoFS, celui-ci favorise la lecture des k blocs de données en clair (i.e. d_1 et d_2). Cette situation est représentée dans la figure 5.4b (cf. page 130). Le processus transfère alors environ 2×500 Mo en parallèle. Ensuite, la donnée est transmise à l'application.

Lors de l'appel de lecture, la correspondance entre le chemin du fichier et son identifiant unique est réalisée par le serveur de métadonnées, qui retourne au client les données relatives au placement des données désirées (i.e. l'ensemble des serveurs utilisés lors de la distribution pendant l'écriture)

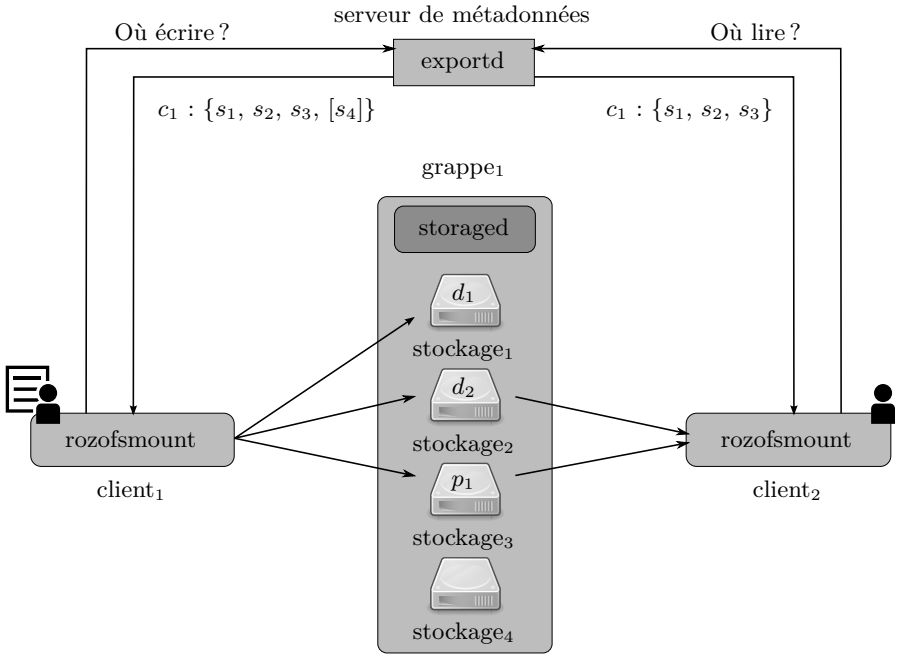


FIGURE 5.5 – Interactions entre les différents composants de RozoFS pendant les opérations d'écriture et de lecture en *layout* 0 (i.e. $(n = 3, k = 2)$). Lorsqu'une application émet une requête, le client contacte le serveur de métadonnées afin d'obtenir la liste des supports de stockage s_i, au sein d'une grappe de serveurs c_j, relatifs à un fichier. Les serveurs de secours sont illustrés entre crochets. Les blocs de données correspondent à d_k alors que les projections Mojette correspondent à p_l.

Impact en cas de pannes

La figure 5.4c illustre la situation d'une panne qui surviendrait lors d'une opération d'écriture. Dans ce cas, la donnée destinée au support de stockage défaillant est transférée au prochain nœud de secours disponible. De manière similaire en lecture, le processus essaie d'accéder à l'information depuis le prochain nœud de secours disponible. Les nœuds de secours sont renseignés dans la liste des supports de stockage émis par *exportd*. Une fois que le client accède à k blocs, une opération de décodage est déclenchée afin de reconstruire l'information initiale. Cette situation est illustrée dans la figure 5.4d.

La figure 5.5 représente la situation où des opérations d'écriture et de lecture sont respectivement déclenchées par client$_1$ et client$_2$. Le premier client envoie une requête d'écriture vers le serveur de métadonnées qui répond avec la liste d'identifiants de nœuds de stockage et d'un identifiant de grappe (*cluster id*) relative à un fichier. Ces identifiants sont en correspondance avec l'adresse IP des serveurs de stockage afin que les clients puissent les joindre directement, et transmettre les données en parallèle. En *layout* 0, le client calcule une projection p_1 depuis les données utilisateurs. La liste renvoyée par *exportd* correspond à l'ensemble des identifiants des nœuds de stockage $\{s_1, s_2, s_3\}$. Si l'un de ces nœuds n'est pas joignable, s_4 est défini comme nœud de secours. Après quoi

de son côté, client$_2$ reçoit la même liste de la part d'*exportd*, une fois qu'il a émis la requête de lire ce même fichier. Dans l'exemple de la figure 5.5, une panne survient sur le nœud s_1 lors de la lecture. En conséquence, le client reçoit d_2 et p_1. Une fois obtenues, ces informations sont passées en entrée du décodeur Mojette afin de reconstituer d_1.

5.3 Évaluation

Nous traitons ici l'expérimentation que nous avons réalisée. Sa mise en œuvre est présentée dans la section 5.3.1, tandis que les sections 5.3.2 et 5.3.3 présentent et analysent respectivement les résultats.

5.3.1 Mise en place de l'évaluation

Dans ce qui suit, on décrivons la configuration et les conditions de notre expérimentation.

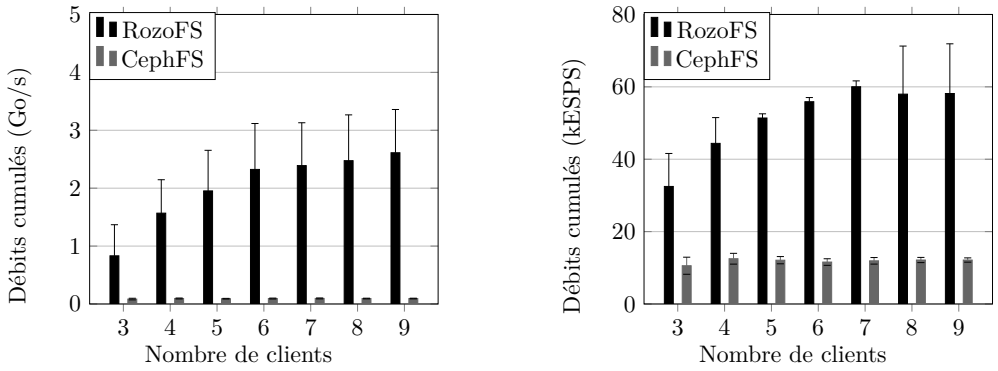
Configuration des compétiteurs

CephFS est un compétiteur intéressant dans une comparaison avec RozoFS puisqu'il s'agit d'un système de fichiers distribué POSIX basé sur des techniques de réplication. Par défaut, CephFS propose une réplication par trois, permettant de supporter deux pannes. En comparaison avec HDFS, CephFS n'est pas conçu pour une application en particulier. L'architecture de CephFS est représentée dans la figure 5.3 (cf. page 127). Elle est composée de services similaires à RozoFS. Toutefois, un service de monitorage supplémentaire est proposé afin de vérifier par exemple l'état d'une grappe de serveurs.

Des éléments aussi complexes que des systèmes de fichiers distribués peuvent être réglés avec de grandes quantités de paramètres. En conséquence, nous suivrons les recommandations énoncées dans les documentations respectives. La plupart du temps ces paramètres correspondent aux valeurs par défaut. Nous configurons RozoFS en *layout* 1 (i.e les écritures distribuent six fichiers de projection et les lectures nécessitent quatre fichiers parmi eux), afin de fournir la même capacité de correction que CephFS qui recommande par défaut de la réplication par trois. Dans CephFS, un paramètre important concerne le nombre de *placement group* (PG). Un PG agrège les objets dans un ensemble de supports de stockage. Dans notre expérimentation, on dispose de 64 PGs pour le stockage des objets, et 64 PGs pour les métadonnées.

Mise en place de l'expérimentation

Toutes les expériences ont été réalisées sur la plate-forme expérimentale Grid'5000 [Bal+13]. En particulier, elles ont été conduites sur la grappe « econome », composée de 22 serveurs. Chaque serveur contient deux CPU Intel Xeon E5-2660 cadencés à 2, 2 Ghz, 64 Go de RAM, de 1 To de disque dur SATA d'une vitesse de 7200 tours par minute, et d'une interface réseau 10 GbE. Huit serveurs sont utilisés pour stocker les données. Un nœud supplémentaire contient le serveur de métadonnées. Une machine supplémentaire est



(a) Débits d'écriture séquentielle (Go/s). (b) Débits d'écriture aléatoire (kESPS).

FIGURE 5.6 – Évaluation des performances d'écriture séquentielle et aléatoire. Les performances sont représentées comme les débits cumulés enregistrés par un nombre croissant de clients. La charge de travail correspond à l'écriture d'un fichier de 100 Mo par client simultanément. En particulier, les accès sont réalisés par blocs de 64 Ko et 8 Ko respectivement pour les accès séquentiels et aléatoires.

nécessaire pour le monitorage de CephFS. Enfin, côté clients, neuf machines sont réservées pour réaliser les opérations de lecture et d'écriture.

Configuration de l'expérimentation

Nous avons utilisé le logiciel IOzone pour cette expérimentation. Il s'agit d'un logiciel largement utilisé pour tester les systèmes de fichiers et les supports de stockage. Il est possible de définir différents tests en fonction de l'E/S désirée : lecture ou écriture ; ainsi qu'en fonction du schéma d'accès voulu : séquentiel ou aléatoire. La particularité d'IOzone est de fournir un mode « cluster » permettant de mesurer les bandes passantes dans le cas où plusieurs clients sont impliqués simultanément dans l'expérimentation. Cette particularité est adaptée aux systèmes de fichiers distribués. IOzone fournit ainsi le débit, ou le nombre d'E/S par seconde (ESPS), mesuré au niveau de chaque client. Dans notre expérimentation, un client implique une opération sur un fichier de 100 Mo. En pratique, les accès séquentiels sont généralement plus longs que les accès aléatoires. C'est pourquoi nous fixons la taille des E/S en accès séquentiel à 64 Ko, et 8 Ko pour les tests en aléatoire (valeurs classiques dans ce type d'expérimentation). Dans la suite, nous évaluons le débit enregistré à mesure que l'on augmente le nombre de clients (de 3 à 9) impliqués simultanément dans les opérations réalisées sur le point de montage ciblé (RozoFS puis CephFS). Les résultats illustrés correspondent à la moyenne de 30 itérations. Les caches des clients sont effacés entre chaque itération afin de garantir l'absence d'effet de cache du côté client.

5.3.2 Résultats de l'expérimentation

Nous verrons dans un premier temps les résultats de l'expérimentation en écriture, avant de nous intéresser aux lectures.

Évaluation en écriture

Dans cette section, nous étudions les performances de RozoFS et de CephFS en écriture séquentielle, puis aléatoire. Les débits mesurés correspondent au quotient de la taille du fichier écrit sur le temps nécessaire pour stocker de façon « sûre » la donnée. Cette « sûreté » correspond à la confirmation que la donnée est stockée de manière redondante au niveau des n supports de stockage. Notons cependant qu'en pratique, à un moment donné, la donnée peut être contenue dans le cache d'un nœud de stockage et non de manière sûre sur le support de stockage de masse sous-jacent.

La figure 5.6 (cf. page 133) illustre les débits obtenus à mesure que l'on augmente le nombre de clients qui écrivent simultanément dans les points de montage. Plus particulièrement, les figures 5.6a et 5.6b représentent respectivement les résultats pour des accès séquentiels et aléatoires. Dans les deux cas, on observe que les performances de RozoFS sont supérieures à celles fournies par CephFS. Alors qu'à mesure que l'on ajoute de nouveaux clients, les performances de RozoFS augmentent jusqu'à une limite correspondant à 2.7 Go/s en séquentiel, et 60000 ESPS en aléatoire, les mesures obtenues pour CephFS n'évoluent pas de manière distinctive.

Il semble alors que RozoFS ait atteint les limites imposées par le matériel (e.g. les disques ou le réseau), tandis que CephFS soit limité par des considérations logicielles. Notre principale intuition à cette proposition est la suivante. Durant l'écriture, grâce au code à effacement Mojette, RozoFS génère, transfère et stocke une quantité d'informations largement inférieure à CephFS (i.e. environ moitié moins comme nous l'avons vu dans la figure 3.2, cf. page 79). De plus, des spécificités liées aux deux logiciels expliquent ces résultats. La plus importante correspond au mode de transmission des données. Lors de l'écriture d'un fichier, RozoFS gère en parallèle plusieurs requêtes d'écriture et est capable de profiter de plusieurs liens réseaux lors de la distribution des blocs sur les différents supports de stockage. En revanche, CephFS souffre de son mode de distribution des répliquas. Lorsqu'une requête d'écriture est émise, CephFS copie un premier répliqua sur un OSD primaire. Cet OSD est par la suite responsable de la réplication et de la distribution des données aux seins des différents OSD du PG. Cette distribution séquentielle des données entraîne un coût significatif dans les débits mesurés.

Pour mettre en évidence cette considération, nous avons réalisé une évaluation des performances en écriture de CephFS. Les paramètres sont similaires à l'expérience précédente, cependant, le facteur de réplication r varie de 1 à 3. Les résultats de cette expérimentation sont fournis dans la figure 5.7 (page 135). Par exemple, lorsque dix clients écrivent simultanément, CephFS atteint un débit proche des 300 Mo/s quand aucune copie d'information n'est générée (i.e.
$r = 1$). En revanche, la valeur de ce débit chute à 120 Mo/s lorsque le système gère $r = 3$ copies d'information. Enfin un élément supplémentaire concerne la gestion de la

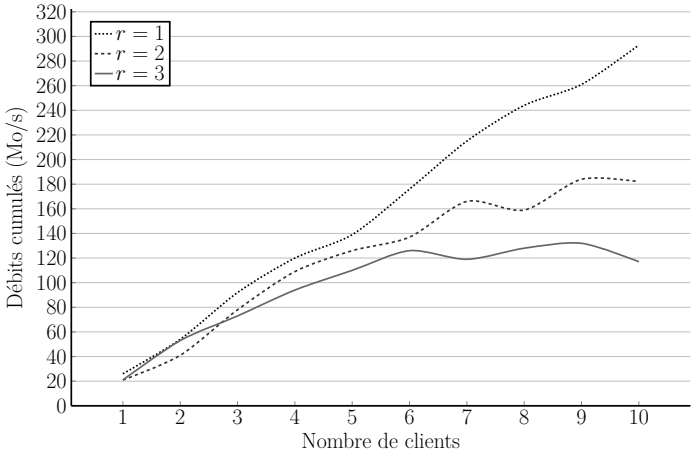


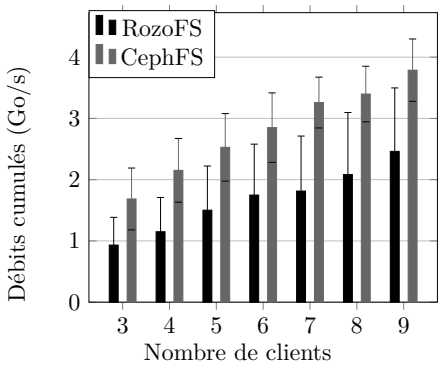
FIGURE 5.7 – Impact du facteur de réplication sur les performances en écriture séquentielle dans CephFS. Ces performances correspondent aux débits cumulés relevés à mesure que l'on augmente le nombre de clients. Chaque client écrit un fichier de 100 Mo par blocs de 64 Ko. La valeur du facteur de réplication r évolue dans l'ensemble $\{1, 2, 3\}$.

cohérence des données. Dans CephFS, un journal est tenu afin garantir des transactions fiables. En conséquence, lorsqu'une E/S modifie la donnée, une E/S supplémentaire est nécessaire pour enregistrer une entrée dans ce journal. Cela réduit significativement les performances, en particulier lorsque plusieurs répliquas sont en jeu (puisqu'autant d'entrées sont nécessaires dans le journal). Un moyen de contrer ce problème est de mettre en place le journal sur un disque séparé, ce qui aurait pu être mis en place sur une plate-forme plus grande qu'*econome*.

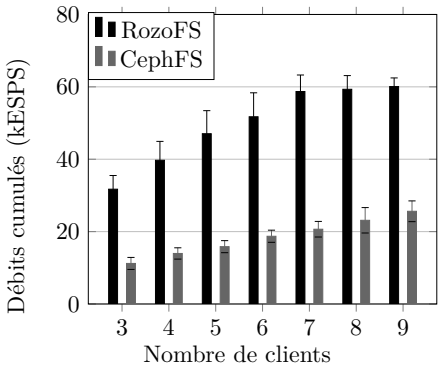
Évaluation en lecture

On considère à présent les performances des deux systèmes en lecture. Les résultats des tests sont illustrés dans la figure 5.8 (cf. page 136). En particulier, la figure 5.8a présente les résultats en accès séquentiel. Dans ce test, les performances de RozoFS sont 30% plus faibles que celles fournies par CephFS. À la différence des opérations en écriture, la lecture met en jeu la récupération de la même quantité d'informations pour les deux systèmes. La différence provient probablement de la taille des blocs utilisés dans chaque système. CephFS accède aux données par des blocs de 4 Mo, tandis que RozoFS utilise des blocs de 4 Ko. En conséquence, le nombre d'E/S nécessaire pour le premier est moins important et CephFS bénéficie des accès séquentiels du test.

La figure 5.8b présente les résultats obtenus en accès aléatoire. Dans ce cas, les performances de RozoFS sont trois plus élevées que celles obtenues par CephFS. Bien que les performances augmentent globalement pour les deux systèmes, il est intéressant de remarquer que celles de RozoFS plafonnent à proximité des 60000 ESPS. Cette limite correspond à la même limite rencontrée dans le test en séquentiel, et représentée dans la figure 5.6a (cf. page 133). Les disques rotatifs offrent généralement les mêmes



(a) Débits de lecture séquentiel (Go/s).



(b) Débits de lecture aléatoire (kESPS).

FIGURE 5.8 – Évaluation des performances de lecture séquentielle et aléatoire. Les performances sont représentées comme les débits cumulés enregistrés par un nombre croissant de clients. La charge de travail correspond à la lecture d'un fichier de 100 Mo par chaque client simultanément. En particulier, les accès sont réalisés par blocs de 64 Ko et 8 Ko respectivement pour les accès séquentiels et aléatoires.

performances en lecture ou écriture quand les accès se font en aléatoire. En conséquence, cette limite correspond probablement à la limite des performances des disques. Il serait intéressant de vérifier si cette limite est repoussée dans le cas où on utilise soit plus de disques durs (pour répartir la charge) soit des disques SSD.

5.3.3 Discussion

Avant de conclure le chapitre, nous proposons de discuter dans un premier temps des résultats obtenus dans notre expérimentation, puis du choix de la version du code à effacement Mojette.

Choix de CephFS CephFS a été choisi comme DFS de comparaison à RozoFS dans cette expérimentation parce qu'il s'agit d'un logiciel libre, proposant un système de fichiers compatible POSIX et de type *scale-out* (i.e. extension du système par ajout de serveurs). Toutefois, ces systèmes sont trop complexes pour pouvoir les comparer directement. Pour s'en convaincre, il suffit de considérer les différences de conception, d'architecture, et de paramètres disponibles. Sans toutes les énumérer, nous proposons de discuter de deux différences fondamentales : Ceph est conçu pour la mise à l'échelle et la cohérence des données. La mise à échelle repose principalement sur l'algorithme décentralisé de distribution de données CRUSH, proposé par WEIL [Wei07] dans ses travaux de thèse. Le second point, qui concerne la cohérence des données, joue un rôle essentiel dans les mesures des performances enregistrées. Ceph est un système de fichiers qui garde trace des opérations de lecture et d'écriture afin de garantir une cohérence forte des données et des transactions. En conséquence, chaque opération d'écriture sur un OSD entraîne une seconde écriture dans le journal. En particulier, si le facteur de réplication

vaut 2, une écriture de l'application revient à déclencher 4 opérations d'écritures (1 E/S sur deux OSDs, et 2 E/S dans le journal).

En conséquence, les résultats obtenus ne permettent pas clairement de se prononcer sur quel est le système de fichiers le plus efficace. Toutefois, ces résultats laissent apparaître le fait que RozoFS est capable de fournir de bonnes performances, tout en divisant le volume de données stockées par deux.

Choix entre version systématique et non-systématique Bien que l'on se soit focalisé sur des codes systématiques pour des raisons de performance, les versions non-systématiques peuvent convenir pour d'autres considérations. Nous proposons de discuter de deux avantages intéressants. Tout d'abord, la distribution des données sous forme de projections peut permettre de compliquer la lecture de la donnée à un tiers malveillant (les données ne sont en effet pas stockées en clair). C'est le principe de l'algorithme de dispersion d'information (ou IDA, pour *Information Dispersal Algorithm*) de Rabin [Rab89]. Pour reconstituer exactement la donnée initiale, il doit disposer de k projections, ce qui signifie corrompre k supports de stockage.

L'équité de l'ensemble des symboles du mot de code constitue le deuxième avantage de cette version. On s'intéresse ici au poids d'une projection dans la distribution d'un bloc de données. En particulier, chaque projection d'un code non-systématique possède le même poids. En revanche, dans le cas systématique, l'opération de lecture tend à favoriser la récupération des k premiers symboles (i.e. les lignes de la grille) dans le cas du code systématique. En conséquence, cela complique la prise de décision concernant par exemple, le délai d'attente en lecture avant de réclamer une projection, lorsque le dernier symbole systématique tarde à arriver.

Les chapitres 3 et 4 ont montré que malgré le gain observé des performances en systématique, la version non-systématique permet déjà de fournir de très hauts débits. Le goulot d'étranglement du système réside alors soit dans la partie matérielle (e.g. disques ou réseau), soit dans le surcout impliqué par la gestion logicielle du DFS (e.g. journalisation, réplication). Le critère des performances n'est donc pour l'instant pas déterminant dans le choix de la version du code. Par conséquent, l'utilisation du code non-systématique dans RozoFS peut se justifier par les avantages présentés précédemment.

Conclusion du chapitre

Ce chapitre a permis de détailler la mise en œuvre du code à effacement Mojette au sein du système de fichiers distribué RozoFS. La section 5.1 a présenté quelques notions sur les systèmes de stockage. En particulier, nous avons expliqué l'évolution de la distribution des données sur des supports de stockage depuis l'invention du RAID dans les années 80. On a par la suite expliqué comment fonctionne un système de fichiers distribué et comment gérer la redondance dans les systèmes de fichiers distribués actuels.

La section 5.2 a décrit le fonctionnement de RozoFS à travers ses trois composants : (i) le serveur de métadonnées, (ii) les serveurs de stockage, (iii) les clients. Nous avons également détaillé les interactions entre ces éléments, et en particulier, le cas des entrées/sorties

tolérantes aux pannes grâce au code à effacement Mojette.

Dans une dernière partie, nous avons évalué les performances de RozoFS dans des tests de charges intensives, menés sur la plate-forme de test Grid'5000. La section 5.3 donne une comparaison des performances de latence en encodage et décodage, de RozoFS avec CephFS. Les résultats obtenus ont montré que dans les conditions de nos tests, RozoFS est capable de fournir de meilleures performances qu'un système de fichiers distribué basé sur de la réplication, tout en divisant par deux le volume de stockage grâce au code à effacement Mojette.

6

Méthode distribuée de reprojection

Sommaire

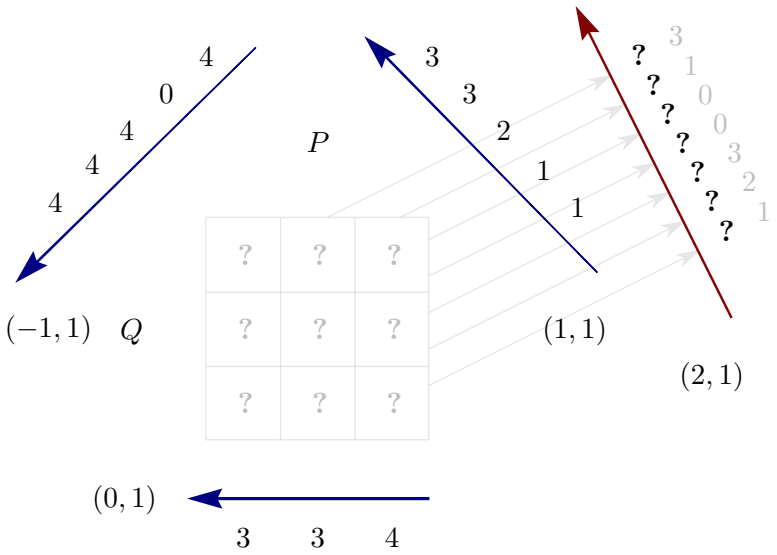


FIGURE 6.1 – Représentation du problème de reprojection. Étant donné un ensemble suffisant (i.e. selon KATZ) de projections, le problème consiste à calculer les valeurs d'une nouvelle projection sans reconstruire la grille. Dans cet exemple, on propose une reprojection suivant la direction $(2, 1)$.

Introduction du chapitre

Les chapitres précédents ont permis de présenter le fonctionnement du code à effacement Mojette dans un système de stockage distribué tel que RozoFS. Les travaux que nous allons à présent détailler utilisent le code Mojette sous sa forme non-systématique afin de déterminer de nouvelles projections. On rappelle que sous cette forme, l'écriture entraîne la distribution de n projections sur différents supports de stockage, qui représentent de manière redondante l'information contenue dans une grille de hauteur k, telle que $k < n$. Le critère de KATZ permet de déterminer l'unicité de la solution de construction. Plus précisément, il permet de garantir qu'un ensemble de k projections constitue un ensemble suffisant pour reconstruire la grille (au plus $n - k$ peuvent être perdues).

Le problème posé dans ce chapitre sera abordé du point de vue du système de stockage, avant d'être décrit dans le formalisme Mojette. À l'échelle du système de stockage, chaque panne qui engendre la perte d'un support de stockage, entraîne également la perte d'une projection. En conséquence, la quantité de redondance diminue avec le nombre de pannes. Avec le temps, les projections sont alors vouées à disparaître. Cette baisse de la redondance constitue le problème principal abordé dans ce chapitre. Afin de ne pas perdre définitivement de la donnée, il est nécessaire de rétablir le seuil de redondance du système. L'objectif ici sera donc de fournir une méthode afin de distribuer de nouvelles données de redondance sur différents supports de stockage. Du point de vue de la transformée Mojette, le problème est illustré dans la figure 6.1. Cet exemple met en jeu un ensemble de trois projections satisfaisant le critère de KATZ. Les directions (p, q) de ces projections

appartiennent à l'ensemble $\{(-1,1),(0,1),(1,1)\}$. Le problème que l'on cherche à résoudre consiste à déterminer les valeurs d'une nouvelle projection (dans cet exemple, suivant la direction $(2,1)$) à partir de l'ensemble des projections existantes. Nous appelons cette opération « reprojection ».

Une approche peu performante de résolution de ce problème consiste à reconstruire la grille à partir des projections, avant de projeter les valeurs reconstruites vers la direction voulue. Nous proposerons dans la suite, une nouvelle méthode de reprojection distribuée, et sans reconstruction de l'information initiale. Cette méthode sera détaillée dans la section 6.1. Une évaluation de cette technique sera ensuite proposée dans la section 6.2 afin de mettre en avant le gain de latence dû à la distribution des calculs de reprojection. Enfin, la section 6.3 exposera trois applications dans lesquelles cette méthode peut être avantageusement utilisée.

6.1 Reprojection sans reconstruction

Dans cette section, nous présentons la nouvelle méthode de reprojection. Nous définissons dans la section 6.1.1 la méthode de reprojection sans reconstruction par des opérations 2D. La section 6.1.2 montre ensuite comment réaliser cette méthode avec des opérations 1D (de convolution et de déconvolution). Enfin, la section 6.1.3 expose des simplifications dans les opérations de convolution.

6.1.1 Méthode de reprojection

Cette section présente la méthode de reprojection. Pour cela, nous définissons dans un premier temps la notion de reconstruction partielle, afin de décomposer le processus de reconstruction de la grille. Cette décomposition permet notamment la distribution des calculs de reprojection. Par linéarité de l'opérateur Mojette, nous montrons que la reprojection par des opérations 2D est possible.

Reconstruction partielle

La notion de reconstruction partielle a été introduite par Philippe [Phi98, chap. 3] dans ses travaux de thèse. Rappelons que le critère de Katz permet de déterminer l'unicité de la solution de reconstruction, relativement à un ensemble de projections. Considérons par ailleurs, un ensemble de projections insuffisant au regard du critère de Katz. Dans ce cas, l'algorithme de reconstruction ne permet pas de reconstruire entièrement la grille. En revanche, les informations contenues dans les projections permettent de reconstruire une partie de la grille. L'algorithme d'inversion reconstruit ainsi un nombre insuffisant de pixels avant d'être bloqué. Dans ce cas, Philippe [Phi98] propose d'affecter une valeur arbitraire à certaines parties de l'image (appelées « érodées » de l'image), afin de poursuivre la reconstruction exacte des pixels de la grille.

Nous proposons ici de débloquer le processus de reconstruction en étendant l'ensemble de projection disponible par des projections de valeur arbitraire (typiquement des projections nulles). L'ensemble de projection ainsi étendu permet de reconstruire exactement la

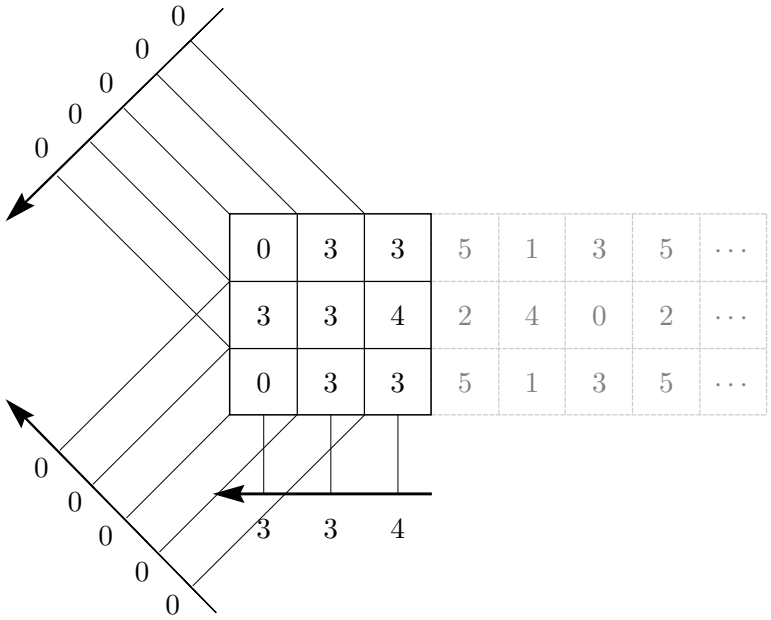


FIGURE 6.2 – Reconstruction partielle $f_S^{\{(0,1)\}}$ d'une grille 3×3 à partir de la projection suivant la direction $(0,1)$, avec $S = \{(0,1),(1,1),(-1,1)\}$. En pratique, l'algorithme de NORMAND et al. [NKÉ06], utilisé pour reconstruire la grille, s'arrête lorsque les pixels de la grille sont reconstruits. Dans le cas de cet exemple, nous l'avons itéré au delà de la grille (partie grisée). Les opérations sont arbitrairement réalisées modulo 6.

grille. Dû fait de l'absence de l'information des projections nulles, nous appelons l'image ainsi obtenue « reconstruction partielle ». La figure 6.2 illustre la reconstruction partielle d'une image 3×3 par la projection $(0,1)$ calculée depuis l'exemple de la figure précédente (cf. figure 6.1). Afin de satisfaire le critère de KATZ, l'ensemble des projections utilisées pour la reconstruction est étendu par deux nouvelles directions, suivant les directions $(-1,1)$ et $(1,1)$. Bien que les éléments de ces deux projections soient nuls, un algorithme de reconstruction peut être utilisé pour reconstruire de manière exacte une image. Pour cette reconstruction, on utilise l'algorithme de NORMAND et al. [NKÉ06]. Notons que dans l'exemple de la figure, l'algorithme ne s'est pas arrêté lorsque tous les pixels de la grille ont été reconstruits. L'information en dehors de la grille (partie grisée) n'est pas essentielle pour l'instant, mais sera utile pour la compréhension de la suite de cette étude.

Nous détaillons à présent la théorie mathématique de cette opération, dans le formalisme de la transformée Mojette. Soit S un ensemble de Q directions de projection de la forme $(p_i, q_i = 1)$. En conséquence $\sum q_i = Q$ et selon le critère de KATZ, toute image $P \times Q$ peut être reconstruite de manière unique par l'ensemble des projections de directions dans S. Si R est un sous-ensemble non vide de S, alors une reconstruction partielle est le processus qui reconstruit une image f_S^R depuis un ensemble de projections de directions dans R (i.e. invalidant le critère de KATZ si $R \subsetneq S$). Parce que les projections ; dont les directions qui sont dans l'ensemble $S \setminus R$, sont nulles, la reconstruction

partielle f_S^R est un fantôme pour ces même directions. Un fantôme, tel qu'introduit dans le second chapitre, est une image $g : M_{\{\{(p,q)\}\}}[g] = 0$. Ses propriétés seront utilisées dans la section 6.1.2.

Reprojection 2D

On considère à présent un ensemble de projections suffisant, dont l'ensemble des directions est S. Soit $M_S[f]$ l'ensemble des projections (i.e. transformée) de f, engendré par l'ensemble des directions S. Soit R un sous-ensemble non vide de S. On considère alors $M_R[f]$, qui correspond à un sous-ensemble de projections de la transformée engendrée par S. En conséquence, si l'ensemble des directions de projection utilisées forme une partition $\mathcal{P}(S)$ (i.e. $\cup_{R_i \in \mathcal{P}(S)} R_i = S$) alors l'ensemble des projections engendré par R_i est égale à la transformée de f par S :

$$\bigcup_{R_i \in \mathcal{P}(S)} M_{R_i}[f] = M_S[f] \ . \tag{6.1}$$

En particulier, il est possible de déterminer les reconstructions partielles engendrées par l'ensemble des directions formant une partition de S. En conséquence, par linéarité de l'opérateur Mojette inverse, la somme des valeurs de ces reconstructions partielles donne l'image f :

$$\sum_i f_S^{R_i} = f \ . \tag{6.2}$$

Il est ainsi possible de reconstruire une image de hauteur Q à partir de ces reconstructions partielles. En particulier, chaque reconstruction partielle f_S^R peut être utilisée pour calculer une projection $M_{\{(p_k,q_k)\}}\left[f_S^R\right]$ suivant une direction (p_k, q_k). Par linéarité de l'opérateur Mojette, la somme des projections obtenues correspond à la projection $M_{\{(p_k,q_k)\}}[f]$. L'équation (6.2) peut alors s'écrire :

$$M_{\{(p_k,q_k)\}}[f] = \sum_{R_i \in \mathcal{P}(S)} M_{\{(p_k,q_k)\}}\left[f_S^{R_i}\right] \ . \tag{6.3}$$

La figure 6.3 (de la page 144) représente un exemple appliqué sur une grille de taille 3×3. Trois remarques peuvent être faites à partir de l'analyse de cette figure. Premièrement, chaque image f_S^R correspond à la reconstruction partielle obtenue à partir d'une projection de direction dans $S = \{(0,1),(1,1),(-1,1)\}$. Cette reconstruction est réalisée à la manière de la figure 6.2. Plus précisément, les images $f_S^{\{(0,1)\}}$, $f_S^{\{(1,1)\}}$, et $f_S^{\{(-1,1)\}}$, sont respectivement reconstruites depuis les projections $M_{\{(0,1)\}}[f]$, $M_{\{(1,1)\}}[f]$ et $M_{\{(-1,1)\}}[f]$.

Deuxièmement, puisque les trois projections utilisées forment une partition de S, la somme des images obtenues par reconstructions partielles reconstruit l'image f (cf. équation (6.2)). Ce traitement est symbolisé sur la figure par le fléchage en pointillés bleu. Notons qu'en pratique, la reconstruction peut se limiter à la taille de l'image. Par ailleurs,

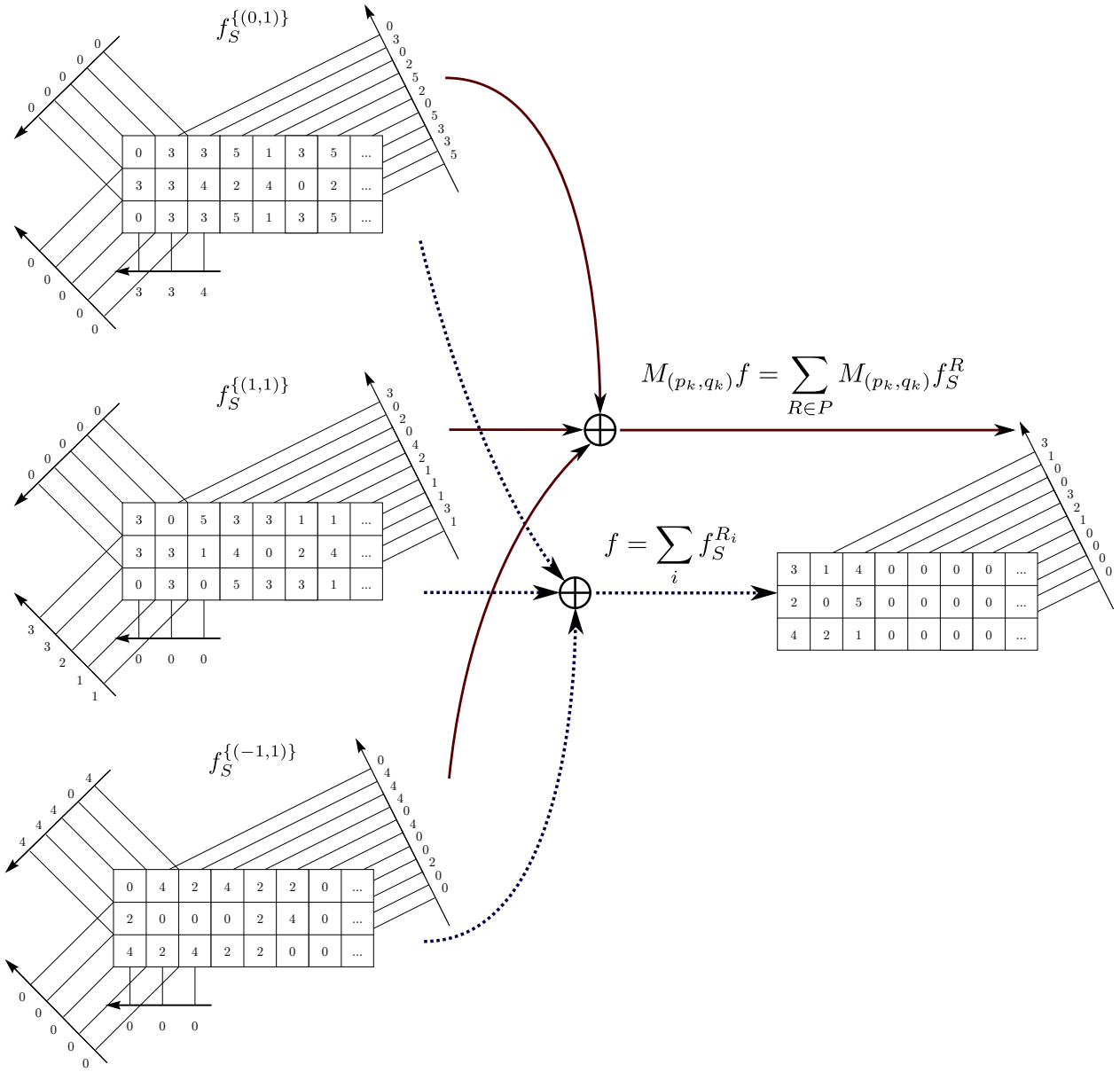


FIGURE 6.3 – Reconstructions partielles à partir des projections $(0,1)$, $(1,1)$ et $(-1,1)$. La reprojection se fait suivant la direction $(2,1)$. La somme des reconstructions partielles est égale à l'image d'origine (pointillés bleus) et la somme des reprojections vaut la projection de l'image (lignes rouges). Les opérations sont réalisées modulo 6.

$g_{\{(0,1)\}}$ $g_{\{(1,1)\}}$ $g_{\{(-1,1)\}}$ $g_{\{(2,1)\}}$

FIGURE 6.4 – Représentation de quatre fantômes élémentaires. Chaque fantôme est défini suivant une direction (p, q) appartenant à l'ensemble $\{(0, 1), (1, 1), (-1, 1), (2, 1)\}$. Par définition, la somme des valeurs de l'image suivant la direction du fantôme est nulle.

on observe que l'ensemble des valeurs obtenues externes à la grille 3×3, est nul (et peut donc être tronqué).

Troisièmement, on représente les projections $M_{\{(2,1)\}}\left[f_S^{\{(0,1)\}}\right]$, $M_{\{(2,1)\}}\left[f_S^{\{(1,1)\}}\right]$ et $M_{\{(2,1)\}}\left[f_S^{\{(-1,1)\}}\right]$ de chaque reconstruction partielle, suivant la nouvelle direction $(2, 1)$. Nous appellerons ces projections des « reprojections partielles ». La somme de ces projections permet de déterminer la projection $M_{\{(2,1)\}}[f]$ que l'on recherche (cf. équation (6.1)). En pratique, dans le cas où les projections sont distribuées (comme dans le cas du stockage distribué, présenté en introduction) le processus de reconstruction peut être distribué puisque la valeur des projections distantes n'a pas besoin d'être connue. En particulier, l'image n'a pas besoin d'être reconstruite pour déterminer la valeur de la nouvelle projection.

Cette première approche a permis de valider la méthode de reprojection par des opérations 2D (i.e. reconstruction et projection de l'image). La notion de reconstruction partielle, permet de distribuer la tâche de reprojection sans avoir à reconstruire l'image.

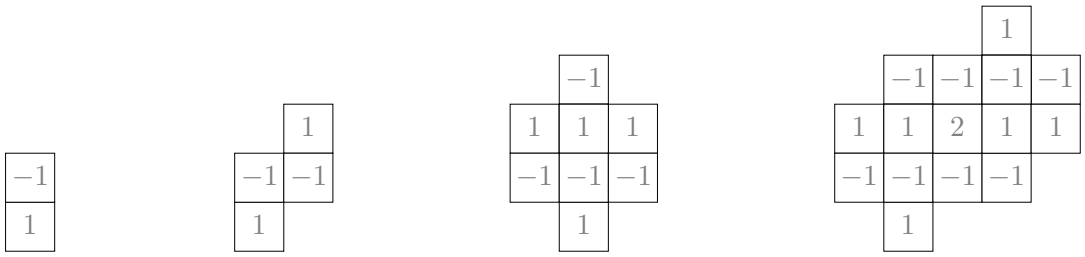
6.1.2 Reconstruction par convolutions 1D

Nous avons vu précédemment une méthode de reprojection 2D. Dans cette section, nous expliciterons le rôle des fantômes dans la méthode précédente, afin de définir l'opération de reprojection à travers des opérations de convolution 1D.

Rappel sur les fantômes

Les fantômes sont des images pour lesquelles les projections suivant un ensemble de directions sont nulles. En conséquence, ils sont invisibles dans le domaine projeté suivant les directions pour lesquelles ils sont définis. Un fantôme élémentaire est défini par une unique direction (p, q) tel que :

$$g_{\{(p,q)\}}(x, y) = \begin{cases} 1 & \text{si } (x, y) = (0, 0) \\ -1 & \text{si } (x, y) = (p, q) \\ 0 & \text{sinon} \end{cases} . \tag{6.4}$$



$g_{\{(0,1)\}}$ $g_{\{(0,1),(1,1)\}}$ $g_{\{(-1,1),(0,1),(1,1)\}}$ $g_{\{(2,1),(-1,1),(0,1),(1,1)\}}$

FIGURE 6.5 – Différentes itérations de la construction du fantôme composé $g_{\{(2,1),(-1,1),(0,1),(1,1)\}}$. Par définition, la somme des valeurs de l'image suivant chaque direction du fantôme est nulle.

La figure 6.4 représente quatre fantômes élémentaires suivant chaque direction de l'ensemble $\{(0,1),(1,1),(-1,1),(2,1)\}$. Il est possible de construire un fantôme composé $g_{\{(p,q)\}}$ à partir de fantômes élémentaires. Pour cela, l'opérateur de convolution 2D $*$ est utilisé ainsi :

$$g_{\{(p,q)\}} = \mathop{\ast}_{i} g_{(p_i,q_i)} \ . \tag{6.5}$$

Le fantôme composé est donc obtenu par la convolution 2D d'un ensemble de fantômes élémentaires. La figure 6.5 illustre cette opération à travers plusieurs itérations de convolution :

$$\begin{aligned} g_{\{(0,1),(1,1)\}} &= g_{(0,1)} * g_{(1,1)} \ , \\ g_{\{(-1,1),(0,1),(1,1)\}} &= g_{\{(0,1),(1,1)\}} * g_{\{(-1,1)\}} \ , \\ g_{\{(2,1),(-1,1),(0,1),(1,1)\}} &= g_{\{(-1,1),(0,1),(1,1)\}} * g_{\{(2,1)\}} \ . \end{aligned}$$

Considérons le fantôme composé construit à partir des directions d'un ensemble de projections. NORMAND et al. [Nor+96] ont montré que si ce fantôme ne pouvait être contenu dans l'image, alors l'ensemble de projections est suffisant pour reconstruire l'image de manière unique.

Puisque les projections d'un fantôme sont nulles, la convolution d'une image f avec un fantôme $g_{\{(p,q)\}}$ donne une image dont les valeurs des projections suivant les directions de $\{(p,q)\}$ sont nulles. La figure 6.6 (cf. page 147) illustre un exemple dans lequel une image f de hauteur $Q = 1$ est convoluée avec le fantôme composé $g_{\{(-1,1),(1,1)\}}$. L'image qui correspond à cette opération possède des projections nulles suivant les directions du fantôme. Dans cet exemple, les opérations sont réalisées modulo 6.

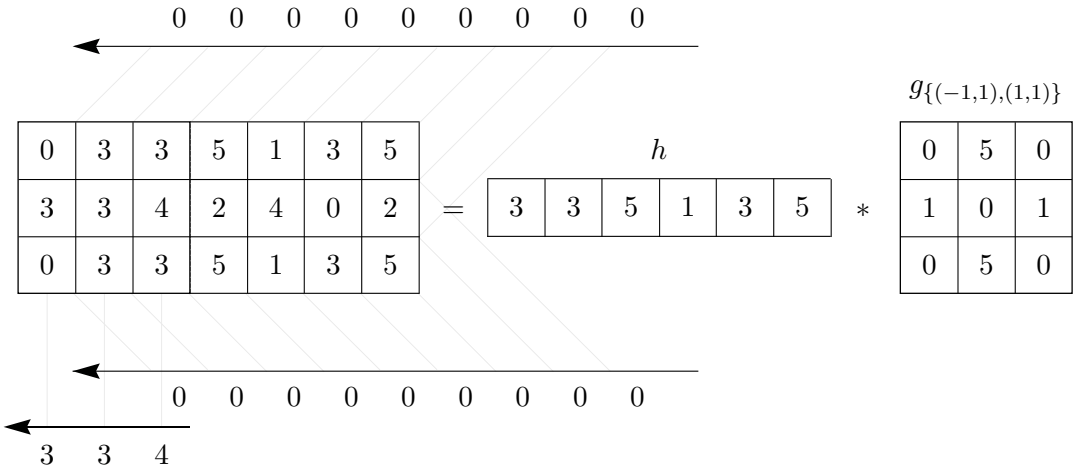


FIGURE 6.6 – Convolution d'une image f avec le fantôme composé $g_{\{(-1,1),(1,1)\}}$. Le résultat est une image dont les projections suivant les directions du fantôme sont nulles. Les opérations sont réalisées modulo 6.

Reprojection par convolutions 1D

Dans cette section, nous explicitons le rôle des fantômes dans la méthode de reprojection définie dans la première section. La reconstruction partielle est par définition une image issue d'un ensemble insuffisant de directions dans R, tel que les projections de directions appartenant à $S \setminus R$ soient nulles. Nous remarquions très justement dans la première section que l'image obtenue est alors un fantôme pour les directions de $S \setminus R$. En conséquence, cette image $f_S^{\{(p_i,q_i)\}}$ est constituée du fantôme composé $g_{S\setminus\{(p_i,q_i)\}}$.

Cette conséquence a été étudiée par PHILIPPE dans ses travaux de thèse. En particulier, dans son théorème de la « Décomposition Unique en Fantômes », PHILIPPE [Phi98, p. 76] démontre qu'il existe une décomposition unique de l'image f par la somme d'une image sous contrainte f_{SC} avec un ensemble de fantômes g_i de directions dans S, telle que :

$$f = f_{SC} + \sum_i a_i g_i \ . \tag{6.6}$$

Comme énoncé dans l'introduction de la reconstruction partielle, PHILIPPE fait le choix de débloquer l'algorithme de reconstruction en initialisant les valeurs d'une partie de l'image (cette partie correspond à l'image érodée). L'image f_{SC} correspond au résultat de la reconstruction partielle quand l'image érodée est nulle. L'ensemble g_i correspond aux fantômes pour chaque élément de l'érodée.

Notre approche diffère de celle de PHILIPPE en cela que pour débloquer l'algorithme, nous initialisons les valeurs des projections de direction dans $S \setminus R$ de manière à ce que ces projections soient nulles. Si l'on considère à présent la reconstruction partielle à partir d'une seule et une seule projection de direction $R = \{(p_i, q_i = 1)\}$, alors :

1. f_{SC} correspond à la reconstruction à partir des projections de directions dans $S \setminus R$,

initialisées à zéro. En conséquence f_{SC} est nulle ;

2. l'érodée correspond alors à une image 1D (i.e. de hauteur 1), et l'image issue de la reconstruction partielle résulte d'une convolution de l'image érodée avec le fantôme composé $g_{S\setminus\{(p_i,q_i)\}}$.

En conséquence, l'équation (6.6) peut s'écrire :

$$f_S^{\{(p_i,q_i=1)\}} = h * g_{S\setminus\{(p_i,q_i)\}} \ , \tag{6.7}$$

où h est une image de hauteur 1 (parce que $q_i = 1$). La figure 6.6 (cf. page 147) illustre cette équation. L'image $f_S^{(0,1)}$ correspond à la convolution d'une image de hauteur $Q = 1$, avec le fantôme $g_{\{(-1,1),(1,1)\}}$ de directions $S \setminus R$. Par linéarité de l'opérateur Mojette, quelle que soit la direction (p_k, q_k), l'équation (6.7) devient :

$$M_{\{(p_k,q_k)\}}\left[f_S^{\{(p_i,q_i)\}}\right] = \underbrace{M_{\{(p_k,q_k)\}}[h]}_{h} * M_{\{(p_k,q_k)\}}\left[g_{S\setminus\{(p_i,q_i)\}}\right] \ , \tag{6.8}$$

puisque la transformée d'une image 1D pour n'importe quelle direction correspond à l'image. En particulier, la reprojection de la reconstruction partielle suivant la direction de $R = \{(p_i, q_i)\}$ correspond à la projection de l'image suivant cette direction :

$$M_{\{(p_i,q_i)\}}\left[f_S^{\{(p_i,q_i)\}}\right] = M_{\{(p_i,q_i)\}}[f] \ . \tag{6.9}$$

Puisque nous connaissons la valeur de la projection $M_{\{(p_i,q_i)\}}[f]$, et que l'on peut déterminer la valeur des projections du fantôme composé $M_{\{(p_i,q_i)\}}\left[g_{S\setminus\{(p_i,q_i)\}}\right]$, ainsi que $M_{\{(p,q)\}}\left[g_{S\setminus\{(p_i,q_i)\}}\right]$ (suivant la nouvelle direction). Par conséquent, on peut déterminer la reprojection $M_{\{(p,q)\}}\left[f_S^{\{(p_i,q_i)\}}\right]$, telle que :

$$\begin{aligned} M_{\{(p_k,q_k)\}}\left[f_S^{\{(p_i,q_i)\}}\right] = {} & M_{\{(p_i,q_i)\}}[f] \\ & *^{-1} M_{\{(p_i,q_i)\}}\left[g_{\{S\setminus\{p_i,q_i\}\}}\right] \\ & * M_{\{(p_k,q_k)\}}\left[g_{\{S\setminus\{p_i,q_i\}\}}\right] \ . \end{aligned} \tag{6.10}$$

Cette équation permet donc de déterminer la reprojection par des opérations de convolution et de déconvolution 1D.

Les algorithmes 2 et 3 décrivent les étapes pour obtenir une projection suivant une direction (p, q) à partir d'un ensemble de projections suffisant. Pour cela, on considère indépendamment chaque projection de l'image $M_{\{(p_i,q_i)\}}[f]$ et une direction de reprojection (p, q). Reprenons l'exemple de la figure 6.2 (cf. page 142) dans laquelle on détermine la reprojection suivant $(2, 1)$ à partir de la projection suivant la direction $(0, 1)$. Dans un premier temps, on détermine le fantôme composé qui correspond à $g_{S\setminus\{0,1\}} = g_{\{(1,1),(-1,1)\}}$. Il est ensuite nécessaire de déterminer la valeur de ses projections suivant la direction (p_i, q_i) et (p, q) :

Algoritme 2 Algorithme de reprojection partielle

Entrée : (P, Q) : les dimensions de la grille
Entrée : $M_{\{p_i,q_i=1\}}[f]$: une projection de la grille
Entrée : S : un ensemble de directions satisfaisant le critère de Katz
Entrée : $(p_k, q_k = 1)$: une direction de reprojection
Sortie : $M_{\{p_k,q_k=1\}}\left[f_S^{\{(p_i,q_i)\}}\right]$: la reprojection partielle suivant la nouvelle direction

1: Calculer $M_{\{(p_i,q_i)\}}\left[g_{S\setminus\{(p_i,q_i)\}}\right]$
2: Calculer $M_{\{(p,q)\}}\left[g_{S\setminus\{(p_i,q_i)\}}\right]$
3: Calculer la reprojection par convolution, puis par déconvolution 1D ▷ équation (6.10)

Algoritme 3 Algorithme de fusion des reprojections partielles

Entrée : un ensemble suffisant de reprojections partielles $M_{\{(p_k,q_k=1)\}}\left[g_{S\setminus\{(p_i,q_i)\}}\right]$
Sortie : $M_{\{(p_k,q_k=1)\}}[f]$

1: **pour tout** (p_i, q_i) d'une partition de S **faire**
2: $\quad M_{\{(p_k,q_k)\}}[f] \leftarrow \sum_{R_i\in\mathcal{P}(S)} M_{\{(p_k,q_k)\}}\left[f_S^{R_i}\right]$ ▷ équation (6.3)
3: **fin pour**

$$M_{\{(0,1)\}}\left[g_{S\setminus\{(0,1)\}}\right] = \begin{pmatrix}5 & 2 & 5\end{pmatrix}, \tag{6.11}$$

$$M_{\{(2,1)\}}\left[g_{S\setminus\{(0,1)\}}\right] = \begin{pmatrix}15 & 0 & 5 & 1\end{pmatrix}. \tag{6.12}$$

si l'on considère des opérations modulo 6. En utilisant l' équation (6.10), on peut alors déterminer la valeur de la reprojection :

$$\begin{aligned} M_{\{(2,1)\}}\left[f_S^{\{(0,1)\}}\right] &= \begin{pmatrix}3 & 3 & 4\end{pmatrix} \\ &\quad *^{-1}\begin{pmatrix}5 & 2 & 5\end{pmatrix} \\ &\quad *\begin{pmatrix}15 & 0 & 5 & 1\end{pmatrix} \\ M_{\{(2,1)\}}\left[f_S^{\{(0,1)\}}\right] &= \begin{pmatrix}0 & 3 & 0 & 2 & 5 & 2 & 0 & 5 & 3 & 3 & 5\end{pmatrix}. \end{aligned} \tag{6.13}$$

Ce qui correspond bien au résultat attendu. Une fois les reprojections partielles calculées, on les additionne pour obtenir la projection $M_{\{(p_k,q_k)\}}[f]$ comme indiqué dans l'équation (6.3).

6.1.3 Simplification des opérations

L'algorithme 3 utilise les projections du fantôme composé $g_{\{(p_k,q_k)\}}$ suivant les directions $(p_i, q_i) \in \mathcal{P}(S)$ et (p_k, q_k). La détermination de ces projections nécessite des opérations

2D de transformation Mojette. Or, par définition, le fantôme composé est issu de la convolution de fantômes élémentaires (voir équation (6.5), page 146). Il est donc possible de les déterminer par des opérations de convolution 1D. L'équation (6.10) peut alors s'écrire sous la forme :

$$\begin{aligned} M_{\{(p_k,q_k)\}}\left[f_S^{\{(p_i,q_i)\}}\right] = (M_{\{(p_i,q_i)\}}[f]) \\ \overset{*^{-1}}{\underset{(p_j,q_j)\in S\setminus\{(p_i,q_i)\}}{}}(M_{\{(p_i,q_i)\}}\left[g_{\{(p_j,q_j)\}}\right]) \\ \overset{*}{\underset{(p_j,q_j)\in S\setminus\{(p_i,q_i)\}}{}}(M_{\{(p_k,q_k)\}}\left[g_{\{(p_j,q_j)\}}\right]) \ . \end{aligned} \tag{6.14}$$

En particulier, chaque $M_{\{(p_i,q_i)\}}\left[g_{(p_j,q_j)}\right]$ de l'équation (6.14) correspond à la projection d'un fantôme élémentaire suivant la direction (p_i, q_i). La détermination de cette projection est triviale et correspond à la séquence :

$$t \mapsto \begin{cases} 1 & \text{si } t = 0 \\ -1 & \text{si } t = j - i \\ 0 & \text{sinon} \end{cases} . \tag{6.15}$$

L'intérêt de cette décomposition est de pouvoir exprimer le calcul de reprojection sans avoir à réaliser d'opération de projection. D'autre part, cette décomposition permet de faciliter les opérations de déconvolution. En effet, les séquences ainsi obtenues sont de la forme $\left(1, \ldots, -1\right)$. La déconvolution correspond alors à un filtre récursif. Cette représentation permet également de révéler parfois des simplifications dans les opérations de convolution. En particulier, convoluer et déconvoluer par la même séquence revient à ne rien faire. Par exemple :

$$\begin{aligned} M_{\{(0,1)\}}\left[g_{S\setminus\{(0,1)\}}\right] &= \begin{pmatrix} -1 & 2 & -1 \end{pmatrix} \\ M_{\{(0,1)\}}\left[g_{S\setminus\{(0,1)\}}\right] &= \left(M_{\{(0,1)\}}\left[g_{\{(-1,1)\}}\right]\right) * \left(M_{\{(0,1)\}}\left[g_{\{(-1,1)\}}\right]\right) \\ &= \begin{pmatrix} -1 & 1 \end{pmatrix} * \begin{pmatrix} 1 & -1 \end{pmatrix} \end{aligned} \tag{6.16}$$

$$\begin{aligned} M_{\{(2,1)\}}\left[g_{S\setminus\{(0,1)\}}\right] &= \begin{pmatrix} 1 & -1 & 0 & -1 & 1 \end{pmatrix} \\ M_{\{(2,1)\}}\left[g_{S\setminus\{(0,1)\}}\right] &= \left(M_{\{(2,1)\}}\left[g_{\{(-1,1)\}}\right]\right) * \left(M_{\{(2,1)\}}\left[g_{\{(-1,1)\}}\right]\right) \\ &= \begin{pmatrix} -1 & 0 & 0 & 1 \end{pmatrix} * \begin{pmatrix} -1 & 1 \end{pmatrix} \end{aligned} \tag{6.17}$$

L'équation (6.14) devient alors :

$$
\begin{aligned}
M_{\{(2,1)\}}\left[f_S^{\{(0,1)\}}\right] &= M_{\{(0,1)\}}[f] \\
&\quad *^{-1}\left(\begin{pmatrix}-1 & 1\end{pmatrix} * \begin{pmatrix}1 & -1\end{pmatrix}\right) \\
&\quad *\left(\begin{pmatrix}-1 & 0 & 0 & 1\end{pmatrix} * \begin{pmatrix}-1 & 1\end{pmatrix}\right), \\
M_{\{(2,1)\}}\left[f_S^{\{(0,1)\}}\right] &= M_{\{(0,1)\}}[f] \\
&\quad *\begin{pmatrix}-1 & 0 & 0 & 1\end{pmatrix} *^{-1} \begin{pmatrix}1 & -1\end{pmatrix} \\
&\quad \underbrace{*\begin{pmatrix}-1 & 1\end{pmatrix} *^{-1} \begin{pmatrix}-1 & 1\end{pmatrix}}_{0}, \\
M_{\{(2,1)\}}\left[f_S^{\{(0,1)\}}\right] &= M_{\{(0,1)\}}[f] * \begin{pmatrix}-1 & 0 & 0 & 1\end{pmatrix} *^{-1} \begin{pmatrix}1 & -1\end{pmatrix}.
\end{aligned}
\tag{6.18}
$$

La décomposition des projections de fantôme permet ici de révéler les opérations de convolution et de déconvolution par la séquence $\begin{pmatrix}-1 & 1\end{pmatrix}$. En conséquence, il est possible de simplifier l'opération de reprojection en supprimant les éléments en rouge. La reprojection d'une reconstruction partielle peut ainsi être calculée à partir de la connaissance d'une projection et d'un ensemble de directions qui vérifie le critère de Katz. L'équation (6.18) montre comment obtenir ce résultat en utilisant uniquement des opérations de convolution et de déconvolution 1D, dont les opérations peuvent parfois se simplifier quand on décompose les projections du fantôme composé.

6.2 Évaluation des performances

Dans cette section, nous nous intéresserons à l'évaluation de la technique décrite dans la section précédente. En particulier, nous verrons dans un premier temps comment nous avons mis en place cette évaluation qui distribue le calcul de reprojection sur différents cœurs CPU. Par la suite, nous analyserons les résultats et verrons que cette technique induit un gain significatif lorsque la taille des blocs utilisés est importante.

6.2.1 Implémentation distribuée par *OpenMP*

L'évaluation met en avant le bénéfice de la distribution des calculs. Pour cela, nous avons réalisé une implémentation des algorithmes 2 et 3 en langage de programmation C. Afin d'exploiter l'ensemble des cœurs du CPU à notre disposition, nous avons utilisé la bibliothèque *Open Multi-Processing* (OpenMP) qui offre un ensemble de directives pour le compilateur, ainsi qu'une interface de programmation pour le calcul parallèle [DE98]. En particulier, notre code repose sur deux fonctions principales montées en série, inspirées du modèle de programmation *Map-Reduce*.

La fonction *map* Cette fonction consiste à distribuer la tâche de reprojection à l'ensemble des nœuds de calcul. Puisque chaque reprojection peut être réalisée de manière

```
1 /* assigne le processus de reprojection pour chaque
     projection, retourne les reconstructions partielles */
2 void map(int w, int k, projection_t *projections,
     projection_t *p_reprojections, int nb_proj, int
     extra_dir)
3 {
4     #pragma omp parallel for schedule(static)
5     for (int i = 0; i < nb_proj; i++)
6         reprojection(projections, i, w, k, nb_proj,
              p_reprojections + i, extra_dir);
7 }
```

Listing 6.1 – Fonction *map*. La directive *pragma omp parallel* permet de rendre la boucle parallèle.

indépendante (chaque reprojection traite des éléments de projections distincts), le processus peut être parallélisé. La liste 6.1 donne un extrait du code qui correspond à cette fonction *map()*. La boucle *for* permettant d'itérer sur chaque projection correspond à la boucle de l'algorithme 2, ligne 1 (cf. page 149). La ligne 4 contient les directives utilisées par *OpenMP* pour distribuer le calcul sur les cœurs du CPU. La fonction *reprojection()* correspond à l'équation (6.10). Elle permet de calculer la valeur des reprojections, et de remplir le buffer *p_reprojections* avec ces valeurs.

La fonction *reduce* Cette seconde fonction prend en entrée les reprojections partielles calculées par la fonction *map*, et consiste à les fusionner pour obtenir la reprojection de l'image. Pour cela, elle retourne un buffer *reproj* correspondant à la projection voulue. Chaque élément de cette projection correspond à la somme des éléments correspondants dans l'ensemble des reprojections. Cette fonction correspond à la ligne 2 de l'algorithme 3 (cf. page 149). La liste 6.2 donne un extrait du code qui correspond à cette fonction *reduce()*. Bien que la ligne 4 concerne les directives d'*OpenMP*, l'addition est une opération trop efficace pour gagner en parallélisme.

Dans notre expérience, nous enregistrons la latence nécessaire pour réaliser ces opérations en fonction de la technique utilisée. En particulier, nos tests mettent en jeu la technique classique qui consiste à reconstruire l'image a priori, et la nouvelle technique décrite précédemment, sans reconstruction de l'image. Puisque ces résultats dépendent de la difficulté de l'opération de reprojection, nous faisons varier la taille $\mathcal{M}$ des blocs étudiés. Les valeurs affichées résultent de la moyenne de 10 itérations. Cette expérience a été conduite sur un serveur de la plate-forme *FEC4Cloud*, présentée précédemment (cf. page 111).

```
1  /* fusionne les produits de map() en une reprojection */
2  void reduce(projection_t *reproj, projection_t *
       p_reprojections, int nb_proj)
3  {
4      #pragma omp parallel for
5      for (int i = 0; i < reproj->size; i++)
6      {
7          pxl_t sum = 0;
8          for (int j = 0; j < nb_proj; j++)
9              sum ^= p_reprojections[j].bins[i];
10         reproj->bins[i] = sum;
11     }
12 }
```

Listing 6.2 – Fonction *reduction*

6.2.2 Résultats et comparaison avec l'approche classique

La figure 6.7 (cf. page 154) représente les résultats obtenus lors de notre expérimentation. Il est tout d'abord intéressant de remarquer que les courbes se croisent lorsque la taille de bloc vaut $\mathcal{M} = 8$ Ko. Avant ce point, la méthode classique est plus efficace que la nouvelle méthode. Ce désavantage de la nouvelle technique provient de la création et de la gestion des *threads* qui coûte un temps significatif dans le cas où l'opération de réduction est simple à réaliser. Après ce point, l'écart des performances enregistrées pour chaque technique devient de plus en plus significatif. En particulier, lorsque $\mathcal{M} = 128$ Ko, la nouvelle technique calcule la reprojection en 1.217×10^{-3} s, ce qui correspond à la moitié du temps nécessaire pour l'ancienne méthode qui nécessite 2.591×10^{-3} s. À mesure que la taille du bloc augmente, le coût de l'opération de reprojection devient de plus en plus important, et en comparaison, le coût de la gestion des *threads* devient négligeable. De plus, l'utilisation de plusieurs cœurs CPU en parallèle offre un gain de temps linéaire.

6.3 Applications de la reprojection

La méthode présentée précédemment peut être utilisée dans plusieurs applications du stockage distribué. Nous proposons dans cette section trois applications. La section 6.3.1 présente tout d'abord une méthode de distribution de données "incompréhensibles" sur des supports de stockages non fiables (favorisant la protection de la vie privée par exemple). La section 6.3.2 concerne quant à elle la génération de redondance supplémentaire dans l'objectif : (i) de rétablir un seuil de redondance ; (ii) d'allouer dynamiquement un seuil de redondance. Enfin, la section 6.3.3 traitera de la réduction de la bande passante utilisée pour la réparation.

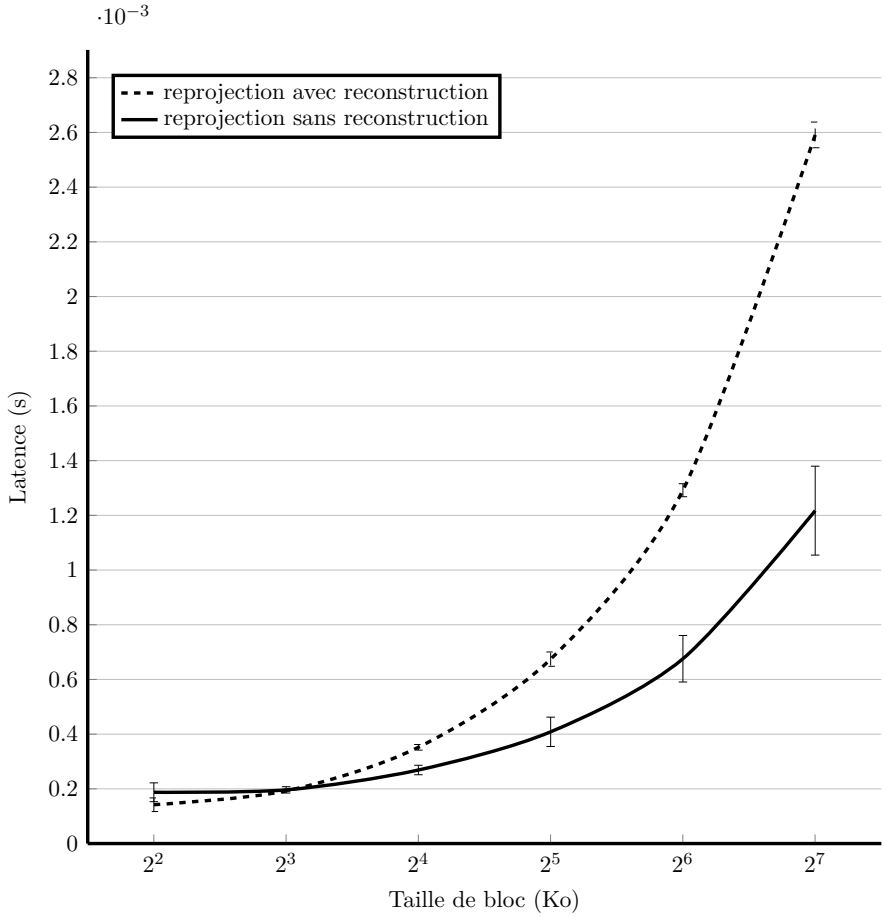


FIGURE 6.7 – Évaluation des performances de latence de la nouvelle technique de reprojection sans reconstruction (représentée en continu) par rapport à la reprojection classique (en pointillés). La taille $\mathcal{M}$ des blocs étudiés évolue de 4 Ko à 128 Ko.

6.3.1 Dispersion d'information incompréhensible

L'utilisation d'une méthode de reprojection sans reconstruction peut être nécessaire dans certaines applications de stockage. Par exemple, la Mojette non-systématique peut être utilisée dans un système de stockage distribué (NDSS) afin de distribuer des données incompréhensibles par un tiers malveillant. Cette technique est basée sur l'« algorithme de dispersion de l'information » (ou IDA pour *Information Dispersal Algorithm*) [Rab89]. Dans cette application, on ne désire pas que les nœuds de stockage puissent accéder à l'information de l'utilisateur. Guédon et al. [Gué+12] ont par exemple utilisé RozoFS pour distribuer des données sensibles (i.e. des documents médicaux) sur des plates-formes publiques de stockage en nuage : Amazon *S3*, *Google Cloud Storage* et *Rackspace Cloud Files*. Dans leur étude, Guédon et al. partent du principe qu'il parait compliqué de reconstruire les données utilisateurs à partir d'une seule projection. Pour reconstruire efficacement la donnée, il serait nécessaire que k plates-formes de stockage partagent leurs informations. C'est le cas de la reprojection avec reconstruction, qui consiste à fournir à un moment donné (par l'opération de reconstruction), l'information de l'utilisateur.

6.3.2 Rétablir un seuil de redondance

Afin de protéger les systèmes de stockage distribués face aux pannes inévitables, il est nécessaire de distribuer des informations de redondance. Cette redondance permet de compenser la perte d'une partie de l'information lors de la lecture d'une donnée. Dans de tels systèmes, le seuil de redondance peut évoluer avec le temps pour deux raisons. La première correspond à la perte naturelle de la redondance. En effet, les pannes entraînent la perte d'information, et cette perte contribue à faire diminuer la redondance au sein du système d'information. La deuxième raison concerne une volonté de modifier le seuil de tolérance par l'administrateur du système d'information. On parle d'allocation dynamique de la tolérance du NDSS. Par exemple, il peut être nécessaire d'augmenter le seuil de redondance de données cruciales pour une entreprise. À l'inverse, il peut être nécessaire de réduire le seuil de redondance pour des raisons économiques.

Dans les deux cas, le principe consiste à distribuer de nouvelles projections au sein de supports de stockage ne contenant pas déjà de projections sur la donnée à protéger. Dans le cas d'une architecture comme celle utilisée dans RozoFS, notre méthode de reprojection pourrait agir de la manière suivante :

1. l'administrateur décide de rétablir un seuil de redondance pour un ensemble de fichiers et en fait la demande au serveur de métadonnées ;
2. le serveur de métadonnées liste les projections relatives à ces fichiers ;
3. pour chaque fichier, un ensemble de k supports de stockage contenant une projection participe au calcul de reprojection et transfère le résultat à l'ensemble des supports de reconstruction ;
4. les supports de reconstruction reconstituent les reprojections en sommant les valeurs qui leur sont transférées.

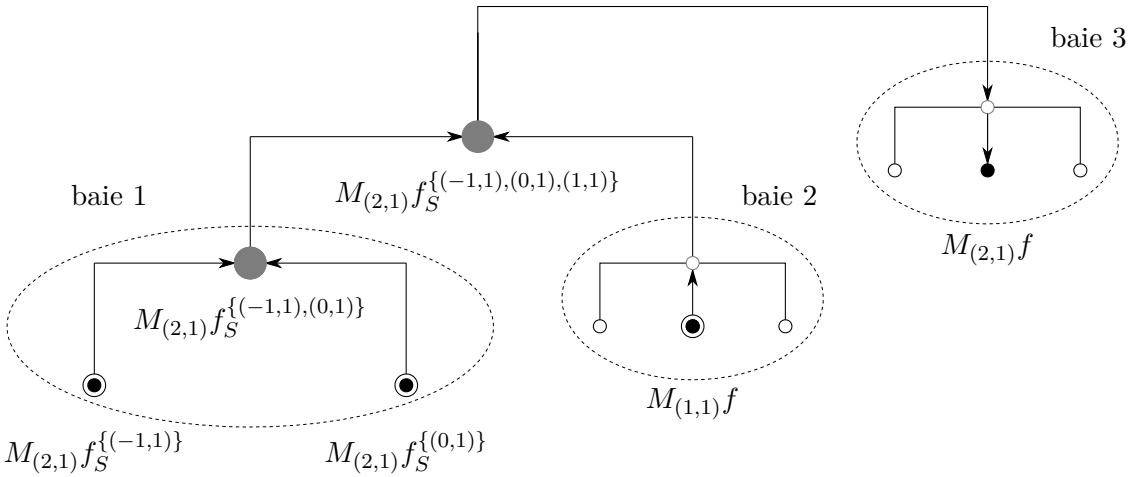


FIGURE 6.8 – Mise en situation permettant d'exploiter notre méthode de reprojection pour faire du codage réseau. L'objectif est de reconstruire une nouvelle projection dans la baie 3. Les reprojections provenant des nœuds cerclés sont successivement fusionnées par les nœuds intermédiaires (en gris) afin de transmettre l'équivalent d'une projection (au lieu de trois) vers la baie 3.

6.3.3 Réduction de la bande passante

La bande passante de la réparation correspond à la quantité d'information transférée lors de l'opération de réencodage. Dans le cas où la quantité de données contenues dans les disques est très importante (i.e. plusieurs téraoctets), le processus de réparation peut à la fois nécessiter plusieurs heures de transfert, et affecter le trafic réseau nécessaire à l'application qui utilise le NDSS. C'est le cas chez FACEBOOK qui analyse une grappe de 3000 nœuds de stockage [Sat+13]. Chaque nœud comptabilise 15 To de données, et l'étude présente une moyenne de 20 pannes par jour. Pour donner une référence au lecteur, le transfert de 15 To d'information nécessite un peu moins de 5 heures sur un réseau délivrant un débit de 1 Gops.

Pour répondre à ce problème, de nouveaux codes ont été proposés. Une première proposition consiste à distinguer des blocs de parité locale (i.e. relation linéaire entre des blocs distribués au sein d'une baie de stockage) et des blocs de parité globale (i.e. relation linéaire entre des blocs distribués sur différentes baies). Désignés sous le nom *Locally Repairable Code* (LRC), ces codes permettent de limiter le trafic réseau généré par la réparation entre les baies en favorisant les transferts intra-baies [Sat+13]. Une deuxième méthode consiste à utiliser la technique de codage réseau (ou NC pour *Network Coding*). Cette technique consiste à combiner l'information de blocs de données au sein d'un nœud de stockage afin de réduire la quantité de données transmises. Ces opérations de combinaisons linéaires ont été par exemple appliquées sur les *Array codes* [Dim+11] ou sur les codes aléatoires [And+14].

Dans cette optique, nos travaux permettent également de combiner l'information par l'opération de réduction. Notre solution se distingue des travaux cités précédemment

puisque la combinaison n'est pas réalisée par les nœuds de stockage participant au réencodage. La réduction ne peut être faite que par des nœuds intermédiaires. La figure 6.8 intègre l'exemple présenté au cours de ce chapitre, dans cette application. Dans cet exemple, trois projections situées dans les baies 1 et 2 (numérotation sur la figure) sont utilisées pour construire la nouvelle projection à placer dans la baie 3. Les nœuds contenant ces projections calculent respectivement leur reprojection $M_{2,1}f_S^{R_i}$. Dans cet exemple, deux nœuds intermédiaires sont utilisés pour combiner les résultats et réduire la quantité d'information transportée par le reste du réseau.

Conclusion du chapitre

Ce chapitre a présenté une nouvelle méthode pour déterminer de nouvelles projections, à partir d'un ensemble déjà existant. Plus spécifiquement, cette technique permet de répondre au problème de la flexibilité et de la tolérance aux pannes du système au cours du temps.

La section 6.1 a permis de présenter la nouvelle méthode de reprojection. Nous y avons défini la notion de reconstruction partielle, afin de décomposer le processus de reconstruction de la grille. Cette décomposition permet notamment la distribution des calculs de reprojection. Par linéarité de l'opérateur Mojette, nous avons montré que la reprojection sans reconstruction de la grille, est possible en utilisant des opérations 2D. La section 6.1.2 montre ensuite, comment traduire ces opérations par des convolutions et des déconvolutions 1D. Cette méthode permet en conséquence une reprojection sans reconstruire d'image. Enfin, la section 6.1.3 expose des simplifications dans les opérations de convolution.

La section 6.2 s'est intéressée à la mise en œuvre d'une expérimentation visant à déterminer le gain de cette nouvelle technique. En particulier, notre expérimentation repose sur l'utilisation d'*OpenMP* pour distribuer les calculs sur l'ensemble des cœurs CPU disponibles. La distribution des calculs de reprojection améliore la latence lorsque de grandes quantités d'informations sont traitées. Dans notre expérimentation, nous avons notamment observé que la latence était divisée par 2 en utilisant cette méthode distribuée.

Trois applications du stockage distribué ont été ensuite présentées dans la section 6.3. La première propose d'exploiter l'absence de reconstruction de la donnée initiale, pour distribuer des projections non exploitables par un tiers, sur des supports de stockage potentiellement malveillants (e.g. fournisseurs de stockage publics). La seconde application permet de maintenir un seuil de redondance voulu dans le système de stockage en générant à la volée, de nouvelles projections. La dernière application permet d'exploiter l'opération de réduction des reprojections pour faire du codage réseau, et ainsi réduire le trafic transféré lors des opérations de réparation.

Conclusion de la partie

L'intégration du code à effacement Mojette dans l'application du stockage distribué a été le sujet de cette seconde et dernière partie. En particulier, nous avons vu dans le chapitre 4 une comparaison théorique puis expérimentale du code à effacement Mojette dans ce contexte. Les résultats théoriques ont montré que ce code nécessite moins d'opérations en encodage, en décodage, et lors de la mise à jour de blocs de données, que d'autres codes MDS de référence (Reed-Solomon et *Array*). Les résultats expérimentaux ont par ailleurs confirmé des vitesses d'encodage et de décodage jusqu'à trois fois supérieures pour le code Mojette. Cette expérience a donc permis de prouver que le code Mojette est le code le plus performant sur des blocs de données adaptés au contexte du stockage (i.e. 4 à 8 Ko).

Le chapitre 5 a détaillé l'intégration de ce code au sein d'un réel système de stockage, en l'occurrence, le système de fichier distribué RozoFS. Celui-ci est la seule solution de stockage qui repose sur le code à effacement Mojette pour distribuer les données de manière redondante. Cette conception a donné suite à une comparaison avec CephFS, qui a permis de montrer que RozoFS est capable de fournir de meilleures performances qu'un système basé sur la réplication, tout en divisant par deux le volume de données stockées.

Enfin, une nouvelle technique de distribution du calcul de projections Mojette, développée dans le chapitre 6, a également été présentée. L'évaluation de cette méthode met en avant la réduction par deux de la latence de l'opération de reprojection grâce à la distribution des calculs. Cette méthode permet ainsi de rétablir un niveau de redondance dans un système de stockage, et peut en conséquence s'inscrire dans un mécanisme de réparation dans le cas de supports de stockage défaillants.

7

Conclusion générale et perspectives

Ce chapitre résume l'ensemble des contributions abordées dans cette thèse, à savoir : l'évaluation des codes à effacement, la conception d'une version systématique du code Mojette, son intégration dans RozoFS, ainsi que l'élaboration d'une méthode de reprojection sans reconstruction. Pour chacune des contributions détaillées dans la suite, des perspectives sont proposées.

7.1 Comparaison et évaluation des codes

Au cours de ce manuscrit, nous avons comparé les codes à effacement étudiés avec notre code à effacement Mojette proposé. Les évaluations théoriques menées dans le chapitre 3 ont permis de montrer que l'algorithme itératif utilisé par ce dernier, nécessite moins d'opérations en encodage et en décodage, par rapport aux *Array* codes dans le cas du RAID-6, et aux codes de Reed-Solomon dans le cas général. Les résultats obtenus lors de l'évaluation expérimentale appuient ce constat dans le chapitre 4. En effet, nous avons montré que notre implémentation du code Mojette encode l'information deux fois plus rapidement que l'encodeur fourni dans ISA-L. Par ailleurs, le gain en décodage peut atteindre un facteur 3 dans les conditions de notre expérimentation. Ce gain de performance nécessite toutefois un rendement sous-optimal, dû à la géométrie de la transformation Mojette, dont les projections ont des tailles variables. Malgré cela, nous avons montré dans le chapitre 3, que le rendement de notre version du code Mojette tend vers le rendement optimal des codes MDS quand la largeur de la grille augmente. En conséquence, on observe un très faible surcout de 3% avec blocs de données de 4 Ko

(taille utilisée en pratique), par rapport aux codes MDS, ce qui est négligeable pour un NDSS.

Plus particulièrement, nous avons proposé une liste de critères (cf. page 33), permettant de distinguer les codes à effacement linéaires entre eux (e.g. critères sur la complexité théorique, l'indépendance des paramètres, ou les débits de l'implémentation). Après avoir observé qu'aucun code de l'état de l'art ne parvenait à répondre à l'ensemble des critères, nous avons montré que le code Mojette est le plus apte à les satisfaire.

Perspective de travail : explorer les liens entre les codes. De fortes connexions existent entre les codes FRT et les codes de Reed-Solomon. Nous avons notamment vu que ces deux codes sont obtenus par une matrice de Vandermonde [Nor+10]. Bien que dans le cas des codes de Reed-Solomon, les coefficients de cette matrice correspondent à des éléments du corps fini, il s'agit dans le cas des codes FRT de monômes dont le degré caractérise une permutation cyclique. Une étude plus poussée des relations entre ces deux codes permettrait d'adapter des algorithmes aux deux formalismes.

De même, le code Mojette partage des liens avec les codes LDPC. En particulier, les deux codes utilisent un algorithme itératif pour l'opération de décodage. Bien que dans le cas des codes LDPC, cet algorithme puisse être bloqué avant d'avoir fini, dans le cas Mojette, le critère de Katz est un moyen efficace pour garantir la capacité de l'algorithme à reconstruire exactement l'information initiale. En conséquence, il est possible que le code Mojette corresponde à une structure particulière des codes LDPC.

7.2 Code à effacement Mojette systématique

Partant du principe que la transformation Mojette n'est pas MDS, nous nous sommes intéressés à réduire la quantité de redondance qu'elle génère. Pour cela, nous avons conçu une version du code Mojette sous sa forme systématique, et avons proposé un algorithme de décodage approprié. Par ailleurs, nous avons montré que la construction de cette version systématique est simple et immédiate, ce qui n'est pas le cas par exemple des codes de Reed-Solomon.

Par sa conception hybride (projections et message), le code Mojette systématique s'approche davantage de l'optimal MDS que sa version non-systématique. Cette amélioration, évaluée dans le chapitre 3, montre que le code est ainsi quasi-MDS.

Un second aspect de cette conception est la réduction du nombre d'opérations nécessaires lors de l'encodage. Cette réduction provient de l'intégration des symboles sources dans le mot de code (c'est de la donnée en moins à calculer). Cette construction permet par exemple de réduire par trois, le nombre de projections à calculer pendant l'encodage d'un code de rendement $\frac{2}{3}$. Par ailleurs, une analyse du nombre d'opérations a permis de montrer que cette construction permet au code de réaliser moins d'opérations en décodage, accélérant ainsi la vitesse de reconstruction.

Ces deux propriétés ont fait l'objet d'une publication [Per+15b]. Depuis, une implémentation du code Mojette systématique a été réalisée, et les améliorations introduites précédemment en théorie, ont été vérifiées par une évaluation dans le chapitre 4. En

conséquence la forme systématique permet au code Mojette de fournir de meilleures performances que les autres codes, tout en améliorant son rendement. Ces éléments permettent de confirmer le choix de ce code performant pour améliorer la vitesse de transmission dans un système de communication protégé par du code à effacement.

Perspective de travail : code Mojette non-systématique et confidentialité. Bien que dans le cas de la gestion des données chaudes, nous nous intéressons principalement aux performances du code (améliorées par la version systématique), d'autres considérations peuvent être prises en compte. En particulier, les problématiques de confidentialité peuvent motiver le choix de la version non-systématique.

Une première approche concerne la distribution des données exclusivement sous la forme de projections. Dans ce cas, l'ensemble de la donnée n'est pas disponible en clair par un tiers malveillant. Toutefois, dans le cas du code Mojette, certaines parties de l'information sont disponibles en clair (typiquement les coins de la grille), et peuvent participer à déterminer des connaissances a priori, facilitant l'attaque, telles que la nature du document (e.g. texte, image), ou la langue d'un fichier texte.

Une seconde approche consiste à tirer profit de la propagation d'information de l'algorithme de reconstruction, pour chiffrer efficacement les blocs d'information. Plus précisément, il suffit de chiffrer les premiers bins des projections pour protéger l'information contenue dans la grille. Cet aspect, est étudié de manière préalable dans [Gué09]. En comparaison, la version systématique nécessite de chiffrer l'ensemble des symboles de la partie systématique (ce qui nécessite beaucoup plus d'opérations). Cette considération présente un exemple où la version non-systématique est plus performante.

7.3 Rôle et impact du code Mojette dans RozoFS

Les bonnes performances des implémentations du code Mojette nous ont motivé à l'intégrer dans le système de fichiers distribué RozoFS. Nous avons ainsi montré dans le chapitre 5, qu'en plaçant le code au niveau des clients, il est possible de distribuer le calcul des projections, afin de ne pas surcharger le serveur de stockage. Ce choix permet également de profiter de l'envoi en parallèle des projections sur plusieurs liens réseau. Nous avons alors fourni une évaluation des performances de RozoFS par rapport à CephFS (réglé en mode réplication des données), à partir de la plate-forme Grid'5000. Cette évaluation montre que RozoFS est capable de fournir de meilleures performances qu'un système basé sur de la réplication, tout en divisant par deux le coût du volume de données. RozoFS est ainsi le seul système de fichiers distribué reposant sur un code à effacement, capable de gérer à la fois des données froides et des données chaudes. Ces travaux ont fait l'objet d'une publication à *CLOSER 2014* [Per+14], et ont depuis été soumis à la revue *Transactions on Storage* [Per+].

Perspective de travail : décentralisation des métadonnées. Un aspect qui n'a pas été abordé dans nos travaux concerne la mise à l'échelle de RozoFS. Premièrement, le serveur de métadonnées de RozoFS est un point de défaillance unique (bien qu'en

pratique, il est répliqué sur plusieurs serveurs). Une première piste pour répondre à ce problème serait de distribuer les métadonnées sous la forme de projections Mojette comme on le fait actuellement pour les données. Cela nécessite en revanche un travail important dans la conception de la gestion des métadonnées puisqu'il est nécessaire de déterminer, à la manière de l'algorithme CRUSH [Wei07] de Ceph, l'ensemble des serveurs en charge des métadonnées, à partir d'une information du fichier (e.g. identifiant unique, chemin de fichier).

Second problème, la centralisation des métadonnées implique un problème dans la mise à l'échelle du système. En effet, le serveur de métadonnées reçoit actuellement l'ensemble des requêtes. À mesure que le nombre de clients augmente, la sollicitation du serveur peut engendrer un problème de ressources. La distribution des métadonnées permettrait de décentraliser ce service et d'harmoniser la charge sur un ensemble de serveurs. La thèse de Bastien CONFAIS (débutée en octobre 2015) traite ce sujet dans le contexte de l'Internet des objets.

7.4 Reprojection sans reconstruction

Dans le dernier chapitre (cf. chapitre 6), nous avons proposé un algorithme distribué permettant de calculer de nouvelles projections sans reconstruire la grille d'origine. Nous avons donné une évaluation de cette méthode dans la section 6.2, qui montre que la distribution des calculs permet une réduction de la latence par un facteur 2. Cette contribution peut s'appliquer à la fois aux systèmes de stockage distribués comme à un contexte réseau (*network coding*). Ces travaux ont fait l'objet d'une publication à Reims Image [PN14a].

Perspective de travail : réparation partielle des projections. Jusque là, nous avons étudié les relations mathématiques entre les projections et la grille. Cependant, il existe également des relations entre les bins des projections. On peut notamment définir des groupes de bins dont la somme des valeurs est nulle. Il est ainsi possible de déterminer la valeur de certains bins, en utilisant cette définition, et par conséquent, d'envisager de reconstruire des bins plutôt que des projections. Cette solution a l'avantage de ne pas avoir à reconstruire la grille, ni la projection entièrement. La première difficulté consiste à déterminer l'ensemble des relations entre les bins pour un ensemble de projections et une grille donnée. Un deuxième problème consiste à déterminer s'il existe des groupes qui engendrent moins d'opérations de reconstruction que d'autres, et comment les déterminer efficacement dans un processus de réparation. Cet aspect correspond à des travaux en cours, en collaboration avec Suayb ARSLAN, Professeur associé à la MEF Université d'Istamboul.

Communications

Revues internationales avec comité de lecture

Dimitri Pertin, Alexandre Van Kempen, Bastien Confais, Didier Féron, Nicolas Normand, Benoît Parrein. « RozoFS : an erasure-coded distributed file system for I/O intensive workloads ». Article soumis à la revue *ACM Transactions on Storage*, décembre 2015.

Conférences internationales avec comité de lecture

Dimitri Pertin, Giulio D'Ippolito, Nicolas Normand, Benoît Parrein. « Spatial Implementation for Erasure Coding by Finite Radon Transform ». In : *International Symposium on signal, Image, Video and Communication*. Valenciennes, France, juil. 2012. URL : https://hal.archives-ouvertes.fr/hal-00716061.

Dimitri Pertin, Sylvain David, Pierre Évenou, Benoît Parrein, Nicolas Normand. « Distributed File System Based on Erasure Coding for I/O-Intensive Applications ». In : *Proceedings of the 4th International Conference on Cloud Computing and Services Science*. CLOSER. Barcelona, Spain, avr. 2014, p. 451–456. DOI : 10.5220/0004960604510456.

Dimitri Pertin, Alexandre Van Kempen, Benoît Parrein, Nicolas Normand. « Comparison of RAID-6 Erasure Codes ». In : *The third Sino-French Workshop on Information and Communication Technologies*. SIFWICT. Nantes, France, juin 2015. URL : https://hal.archives-ouvertes.fr/hal-01162047.

Dimitri Pertin, Nicolas Normand. « Re-projection without Reconstruction ». In : *9ème Journées du Groupe de travail de Géométrie Discrète, Reims Image 2014*. Reims, France, nov. 2014, p. 43. URL : https://hal.archives-ouvertes.fr/hal-01164872.

Dimitri Pertin, Benoît Parrein, Normand Nicolas. *The Mojette erasure code for distributed file systems*. Session poster. Amsterdam, The Netherlands : EuroSys'14 (Session poster), avr. 2014.

Communications internationales sans acte

Benoît Parrein, Nicolas Normand, **Dimitri Pertin**, Jérôme Lacan, Jonathan Detchart. *FEC4Cloud : a research project for promoting erasure codes in Cloud storage architectures*. Bordeaux, France : Algebra, Codes et Networks (ACN), juin 2014. URL : https://hal.archives-ouvertes.fr/hal-01150750.

Dimitri Pertin. *RozoFS : A Distributed File System based on Erasure Coding for I/O Intensive Workloads*. Workshop on Storage and Processing of Big Data, WOS 4. Technicolor, Cesson-Sévigné, France, 4 déc. 2014. URL : http://www.bretagne-networking.org/wos4.

Dimitri Pertin. *RozoFS : A High-Performance Encoded-based Distributed Filesystem*. Future Cloud Symposium. Session poster. Inria Conference Centre, Rennes, France : EIT Digital, 10 19 2015. URL : http://www.eitdigital.eu/cloudsymposium2015/.

Communications nationales sans acte

Benoît Parrein, **Dimitri Pertin**, Nicolas Normand, Féron Didier. *RozoFS : a fault tolerant I/O intensive distributed file system based on Mojette erasure code*. Toulouse, France : Workshop Autonomic, oct. 2014. URL : https://hal.archives-ouvertes.fr/hal-01150741.

Benoît Parrein, Jérôme Lacan, Nicolas Normand, **Dimitri Pertin**, Jonathan Detchart, Alexandre Van Kempen. « FEC4Cloud : a research project promoting erasure coding for Cloud storage architectures ». In : *Rendez-vous de la Recherche et de l'Enseignement de la Sécurité des Systèmes d'Information (RESSI)*. Troyes, France, mai 2015. URL : https://hal.archives-ouvertes.fr/hal-01156462.

Dimitri Pertin, Nicolas Normand. *Re-projection without reconstruction*. Mojette Day 2015. Polytech Nantes, Nantes, France, fév. 2015. URL : http://www.mojette.org/event/mojette-day/.

Écoles doctorales

Dimitri Pertin. « Réduire considérablement le coût et la complexité du stockage en nuage grâce au codage Mojette ». In : *14ème Journée des Doctorants de l'ED STIM*. JDOC. Nantes, France, mar. 2014.

Dimitri Pertin. *The Mojette erasure code as a geometric erasure code for reliable distributed hot data systems*. Session des doctorants. Flaine, France : Hot Topic in Distributed Computing, HTDC 2015, mar. 2015. URL : htdc2015.inria.fr/fr/.

Pré-publication/document de travail

Dimitri Pertin, Didier Féron, Alexandre Van Kempen, Benoît Parrein. « Performance evaluation of the Mojette erasure code for fault-tolerant distributed hot data storage ». In : *CoRR* abs/1504.07038 (avr. 2015). URL : http://arxiv.org/abs/1504.07038.

Démonstrations

Benoît Parrein, Sylvain David, **Dimitri Pertin**. *Video streaming over RozoFS with fault tolerance*. NEM SUMMIT. Cité de Congrès, Nantes, France, 28 oct. 2013.

Dimitri Pertin, Sylvain David, Didier Féron, Pierre Évenou, Normand Nicolas, Benoît Parrein. *RozoFS : a fault tolerant distributed file system based on the Mojette transform*. Third Workshop On Storage and Cloud Computing, WOS 3. Technicolor, Cesson-Sevigné, France, nov. 2013. URL : http://www.bretagne-networking.org/wos3.

Vulgarisations

Dimitri Pertin. *Reduce the storage consumption of your storage clusters with RozoFS*. Free and Open Source Software Developers' European Meeting FOSDEM'14. Université Libre de Bruxelles, Belgium, fév. 2014. URL : https://archive.fosdem.org/2014/schedule/event/hpc_devroom_rozofs/.

Dimitri Pertin. *Stockage distribuée à la Mojette : Gastronomie, Tomographie, Transmission et Stockage*. Vidéo de ma présentation de vulgarisation de mes travaux de thèse à Pas Sage En Seine. Paris, France, avr. 2014. URL : http://numaparis.ubicast.tv/videos/rozofs/.

Dimitri Pertin, Nicolas Normand. *RozoFS : Le Système de Fichiers Distribués basé sur un code à effacement*. Forum de Médias. Montaigu, France : Lycée Léonard de Vinci, fév. 2014. URL : http://issuu.com/ln23/docs/forum_28_01_14/1?e=2608851/6519442.

Bibliographie

[And+14] Fabien André, Anne-Marie Kermarrec, Erwan Le Merrer, Nicolas Le Scouarnec, Gilles Straub, Alexandre Van Kempen. « Archiving Cold Data in Warehouses with Clustered Network Coding ». In : *Proceedings of the Ninth European Conference on Computer Systems*. EuroSys'14. Amsterdam, The Netherlands : ACM, 2014. ISBN : 978-1-4503-2704-6. DOI : 10.1145/2592798.2592816 (cf. p. 156).

[Anv04] Hans Peter Anvin. *The mathematics of RAID-6*. Jan. 2004. URL : https://www.kernel.org/pub/linux/kernel/people/hpa/raid6.pdf (cf. p. 98, 109).

[APV96] Hari Adiseshu, Guru Parulkar, George Varghese. « A Reliable and Scalable Striping Protocol ». In : *Conference Proceedings on Applications, Technologies, Architectures, and Protocols for Computer Communications*. SIGCOMM '96. Palo Alto, California, USA : ACM, 1996, p. 131–141. ISBN : 0-89791-790-1. DOI : 10.1145/248156.248169 (cf. p. 82).

[Ave+06] Amir Averbuch, Ronald R. Coifman, David L. Donoho, Michael Elad, Moshe Israeli. « Fast and accurate polar Fourier transform ». In : *Applied and Computational Harmonic Analysis* 21.2 (2006), p. 145–167. DOI : 10.1016/j.acha.2005.11.003 (cf. p. 47).

[Bal+13] Daniel Balouek, Alexandra Carpen Amarie, Ghislain Charrier, Frédéric Desprez, Emmanuel Jeannot, Emmanuel Jeanvoine, Adrien Lèbre, David Margery, Nicolas Niclausse, Lucas Nussbaum, Olivier Richard, Christian Pérez, Flavien Quesnel, Cyril Rohr, Luc Sarzyniec. « Adding Virtualization Capabilities to the Grid'5000 Testbed ». In : *Cloud Computing and Services Science*. Sous la dir. d'IvanI. IVANOV, Marten SINDEREN, Frank LEYMANN et Tony SHAN. T. 367. Communications in Computer and Information Science. Springer International Publishing, 2013, p. 3–20. ISBN : 978-3-319-04518-4. DOI : 10.1007/978-3-319-04519-1 (cf. p. 132).

[BBV96] Mario Blaum, Jehoshua Bruck, Alexander Vardy. « MDS array codes with independent parity symbols ». In : *Transactions on Information Theory* 42.2 (mar. 1996), p. 529–542. ISSN : 0018-9448. DOI : 10.1109/18.485722 (cf. p. 95).

[Bel84] Chester G. Bell. « The Mini and Micro Industries ». In : *Computer - IEEE Centennial : the state of computing* 17.10 (oct. 1984), p. 14–30. ISSN : 0018-9162. DOI : 10.1109/MC.1984.1658955 (cf. p. 94).

[Bla+95] Mario Blaum, Jim Brady, Jehoshua Bruck, Jai Menon. « EVENODD : an efficient scheme for tolerating double disk failures in RAID architectures ». In : *Transactions on Computers* 44.2 (fév. 1995), p. 192–202. ISSN : 0018-9340. DOI : 10.1109/12.364531 (cf. p. 95, 98, 120).

[Blö+95] Johannes Blömer, Malik Kalfane, Marek Karpinski, Richard Karp, Michael Luby, David Zuckerman. *An XOR-based erasure-resilient coding scheme*. Rapp. tech. TR-95-048. International Computer Science Institute, août 1995. URL : http://www.icsi.berkeley.edu/cgi-bin/pubs/publication.pl?ID=000988 (cf. p. 36, 104–107).

[Boh+01] Vasken Bohossian, Chenggong C. Fan, Paul S. LeMahieu, Marc D. Riedel, Jehoshua Bruck, Lihao Xu. « Computing in the RAIN : A Reliable Array of Independent Nodes ». In : *IEEE Transactions on Parallel and Distributed Systems* 12.2 (fév. 2001), p. 99–114. ISSN : 1045-9219. DOI : 10.1109/71.910866 (cf. p. 120).

[BR54] Ronald N. Bracewell, James A. Roberts. « Aerial Smoothing in Radio Astronomy ». In : *Australian Journal of Physics* 7.4 (jan. 1954), p. 615–640. DOI : 10.1071/PH540615 (cf. p. 56).

[BR93] Mario Blaum, Ron M. Roth. « New array codes for multiple phased burst correction ». In : *Transactions on Information Theory* 39.1 (jan. 1993), p. 66–77. ISSN : 0018-9448. DOI : 10.1109/18.179343 (cf. p. 104, 106).

[BR99] Mario Blaum, Ron M. Roth. « On lowest density MDS codes ». In : *Transactions on Information Theory* 45.1 (jan. 1999), p. 46–59. ISSN : 0018-9448. DOI : 10.1109/18.746771 (cf. p. 120).

[Bra56] Ronald N. Bracewell. « Strip integration in radio astronomy ». In : *Australian Journal of Physics* 9.2 (1956), p. 198–217. DOI : 10.1071/PH560198 (cf. p. 42, 43, 56).

[Cha+12] Shekhar S. Chandra, Imants D. Svalbe, Jeanpierre Guédon, Andrew M. Kingston, Nicolas Normand. « Recovering Missing Slices of the Discrete Fourier Transform Using Ghosts ». In : *Transactions on Image Processing* 21.10 (oct. 2012), p. 4431–4441. ISSN : 1057-7149. DOI : 10.1109/TIP.2012.2206033 (cf. p. 52, 57).

[Che+94] Peter M. Chen, Edward K. Lee, Garth A. Gibson, Randy H. Katz, David A. Patterson. « RAID : High-performance, Reliable Secondary Storage ». In : *ACM Computing Surveys* 26.2 (juin 1994), p. 145–185. ISSN : 0360-0300. DOI : 10.1145/176979.176981 (cf. p. 120).

[CLW69] James W. Cooley, Peter A.W. Lewis, Peter D. Welch. « The finite Fourier transform ». In : *Transactions on audio and electroacoustics* 17.2 (juin 1969), p. 77–85. ISSN : 0018-9278. DOI : 10.1109/TAU.1969.1162036 (cf. p. 42, 63).

[CMC07a] David Coeurjolly, Annick Montanvert, Jean-Marc Chassery. « Éléments de base ». In : *Géométrie discrète et images numériques*. Sous la dir. de David Coeurjolly, Annick Montanvert et Jean-Marc Chassery. Traité IC2, série signal et image. ISBN 13 : 978-2-7462-1643-3. Hermès, sept. 2007. Chap. 1. URL : https://hal.archives-ouvertes.fr/hal-00175341 (cf. p. 47, 48).

[Cor+04] Peter Corbett, Bob English, Atul Goel, Tomislav Grcanac, Steven Kleiman, James Leong, Sunitha Sankar. « Row-diagonal parity for double disk failure correction ». In : *Proceedings of the 3rd USENIX Conference on File and Storage Technologies*. FAST '04. San Francisco, CA : USENIX Association, 2004. URL : http://static.usenix.org/event/fast04/tech/corbett/corbett_html/ (cf. p. 95, 98, 100, 120).

[Cor63] Allan M. Cormack. « Representation of a Function by its Line Integrals, with Some Radiological Applications ». In : *Journal of Applied Physics* 34.9 (1963), p. 2722–2727. DOI : 10.1063/1.1729798 (cf. p. 43).

[Cor82] Tim Cornwell. « Image restoration and the CLEAN technique ». In : *Synthesis Mapping*. T. 1. 1982, p. 9 (cf. p. 56).

[CPK14] John D. Cook, Robert Primmer, Ab Kwant. « Compare cost and performance of replication and erasure coding ». In : *Hitachi Review* 63 (2014), p. 304. URL : http://www.hitachi.com/rev/pdf/2014/r2014_july.pdf (cf. p. 12, 125).

[CR08] Mathieu Cunche, Vincent Roca. *Improving the Decoding of LDPC Codes for the Packet Erasure Channel with a Hybrid Zyablov Iterative Decoding/Gaussian Elimination Scheme*. Research Report RR-6473. INRIA, 2008, p. 19. URL : https://hal.inria.fr/inria-00263682 (cf. p. 37).

[CS08] Shekhar S. Chandra, Imants Svalbe. « A method for removing cyclic artefacts in discrete tomography using latin squares ». In : *19th International Conference on Pattern Recognition*. Déc. 2008. DOI : 10.1109/ICPR.2008.4761615 (cf. p. 52).

[CS14] Shekhar S. Chandra, Imants Svalbe. « Exact Image Representation via a Number-Theoretic Radon Transform ». In : *IET Computer Vision* 8.4 (août 2014), p. 338–346. ISSN : 1751-9632. DOI : 10.1049/iet-cvi.2013.0101 (cf. p. 55).

[CSG08] Shekhar Chandra, Imants Svalbe, Jean-Pierre Guédon. « An Exact, Non-iterative Mojette Inversion Technique Utilising Ghosts ». English. In : *Discrete Geometry for Computer Imagery*. Sous la dir. de David Coeurjolly, Isabelle Sivignon, Laure Tougne et Florent Dupont. T. 4992. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2008, p. 401–412. isbn : 978-3-540-79125-6. doi : 10.1007/978-3-540-79126-3_36 (cf. p. 53, 56–58, 60).

[Cun10] Mathieu Cunche. « Codes AL-FEC hautes performances pour les canaux à effacements : variations autour des codes LDPC ». Version du 30 juin 2010. Thèse de doct. Université Joseph-Fourier - Grenoble I ; Institut National Polytechnique de Grenoble - INPG, mar. 2010. url : https://tel.archives-ouvertes.fr/tel-00451336 (cf. p. 32).

[Dav+15] Sylvain David, Pierre Evenou, Jean-Pierre Guedon, Nicolas Normand, Benoît Parrein. « Procédé et appareil permettant de reconstruire un bloc de données ». Centre National de la Recherche Scientifique (CNRS) Université de Nantes. WO Patent App. PCT/EP2014/071,310. 23 avr. 2015. url : http://www.google.com/patents/WO2015055450A1?cl=fr (cf. p. 73).

[DE98] Leonardo Dagum, Rameshm Enon. « OpenMP : an industry standard API for shared-memory programming ». In : *Computational Science & Engineering* 5.1 (jan. 1998), p. 46–55. issn : 1070-9924. doi : 10.1109/99.660313 (cf. p. 151).

[Der15] Henri Der Sarkissian. « Tomographie et géométrie discrètes avec la transformée Mojette ». Thèse de doct. Université de Nantes, juin 2015. url : https://hal.archives-ouvertes.fr/tel-01217874 (cf. p. 43).

[DG08] Jeffrey Dean, Sanjay Ghemawat. « MapReduce : Simplified data processing on large clusters ». In : *Communication of the ACM* 51.1 (jan. 2008), p. 107–113. issn : 0001-0782. doi : 10.1145/1327452.1327492 (cf. p. 10, 125).

[Dim+11] Alexandros G. Dimakis, Ramchandran Kanan, Yunnan Wu, Changho Suh. « A Survey on Network Codes for Distributed Storage ». In : *Proceedings of the IEEE* 99.3 (mar. 2011), p. 476–489. issn : 0018-9219. doi : 10.1109/JPROC.2010.2096170 (cf. p. 156).

[DR97] Pierre Duhamel, Olivier Rioul. « Codage Conjoint Source/Canal : Enjeux et Approches ». In : *16° Colloque sur le Traitement du Signal et des Images*. T. 7. Groupe d'Etudes du Traitement du Signal et des Images, GRETSI. Grenoble, sept. 1997, p. 699–704 (cf. p. 22).

[Dum+07] Jean-Guillaume Dumas, Jean-Louis Roch, Eric Tannier, Sébastien Varrette. *Théorie des Codes : compression, cryptage, correction*. Sciences Sup. Dunod, jan. 2007, p. 352. isbn : 2100506927. url : https://hal.archives-ouvertes.fr/hal-00318546 (cf. p. 27).

[Eli55] Peter Elias. « Coding for Two Noisy Channels ». In : *Information Theory, Third London Symposium*. T. 67. London, England. 1955 (cf. p. 27).

[Eva11] Dave Evans. *The Internet of Things : How the Next Evolution of the Internet Is Changing Everything*. White paper. CISCO Internet Business Solutions Group, avr. 2011. 11 p. URL : http://www.cisco.com/c/dam/en_us/about/ac79/docs/innov/IoT_IBSG_0411FINAL.pdf (cf. p. 9).

[Fan+09] Bin Fan, Wittawat Tantisiriroj, Lin Xiao, Garth Gibson. « DiskReduce : RAID for data-intensive scalable computing ». In : *Proc. PDSW 2009*. Portland, Oregon : ACM, 2009, p. 6–10. ISBN : 978-1-60558-883-4. DOI : 10.1145/1713072.1713075 (cf. p. 12, 121, 126, 130).

[Far+06] Gerard Faria, Jukka A. Henriksson, Erik Stare, Pekka Talmola. « DVB-H : Digital Broadcast Services to Handheld Devices ». In : *Proceedings of the IEEE* 94.1 (jan. 2006), p. 194–209. ISSN : 0018-9219. DOI : 10.1109/JPROC.2005.861011 (cf. p. 32).

[Fly72] Michael Flynn. « Some Computer Organizations and Their Effectiveness ». In : *Transactions on Computers* C-21.9 (sept. 1972), p. 948–960. ISSN : 0018-9340. DOI : 10.1109/TC.1972.5009071 (cf. p. 109).

[For+10] Daniel Ford, François Labelle, Florentina I. Popovici, Murray Stokely, Van-Anh Truong, Luiz Barroso, Carrie Grimes, Sean Quinlan. « Availability in Globally Distributed Storage Systems ». In : *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation*. OSDI'10. Vancouver, BC, Canada : USENIX Association, 2010. URL : http://dl.acm.org/citation.cfm?id=1924943.1924948 (cf. p. 11).

[Fra24] Jérôme Franel. « Les suites de Farey et le problème des nombres premiers ». In : *Nachrichten von der Gesellschaft der Wissenschaften zu Göttingen, Mathematisch-Physikalische Klasse* (1924), p. 198–201. URL : http://eudml.org/doc/59156 (cf. p. 49).

[Gal62] Robert G. Gallager. « Low-density parity-check codes ». In : *IRE Transactions on Information Theory* 8.1 (jan. 1962), p. 21–28. ISSN : 0096-1000. DOI : 10.1109/TIT.1962.1057683 (cf. p. 22, 33, 36).

[GBB95] Jean-Pierre Guédon, Dominique Barba, Nicole Burger. « Psychovisual image coding via an exact discrete Radon transform ». In : *Proceedings of SPIE*. T. 2501. Visual Communications et Image Processing '95. Avr. 1995, p. 562–572. DOI : 10.1117/12.206765 (cf. p. 13, 62, 67).

[GGL03] Sanjay Ghemawat, Howard Gobioff, Shun-Tak Leung. « The Google File System ». In : *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*. SOSP '03. Bolton Landing, NY, USA : ACM, 2003, p. 29–43. ISBN : 1-58113-757-5. DOI : 10.1145/945445.945450 (cf. p. 10, 12, 122, 125, 126).

[Gia98] Dominic Giampaolo. *Practical File System Design with the Be File System*. 1st. San Francisco, CA, USA : Morgan Kaufmann Publishers Inc., 1998. ISBN : 1558604979 (cf. p. 126).

[Gib+89] G. A. Gibson, L. Hellerstein, R. M. Karp, D. A. Patterson. « Failure Correction Techniques for Large Disk Arrays ». In : *SIGARCH Computer Architecture News - Special issue : Proceedings of ASPLOS-III : the third international conference on architecture support for programming languages and operating systems* 17.2 (avr. 1989), p. 123–132. ISSN : 0163-5964. DOI : 10.1145/68182.68194 (cf. p. 120).

[GR12] John Gantz, David Reinsel. « The Digital Universe in 2020 : Big data, Bigger Digital Shadows, and Biggest Growth in the Far East ». In : *IDC iView : IDC Analyze the future* 2007 (déc. 2012). Sponsored by EMC Corporation. URL : https://www.emc-technology.com/collateral/analyst-reports/idc-the-digital-universe-in-2020.pdf (cf. p. 9).

[Gre13] Brendan Gregg. *Systems Performance : Enterprise and the Cloud*. 1st. Upper Saddle River, NJ, USA : Prentice Hall Press, 2013. ISBN : 978-0133390094 (cf. p. 81).

[Gué+12] Jeanpierre Guédon, Pierre Evenou, Pierre Tervé, Sylvain David, Jérôme Béranger. « CEDIMS : Cloud Ethical DICOM Image Mojette Storage ». In : *Advanced PACS-based Imaging Informatics and Therapeutic Applications*. T. 8319. SPIE proceedings. San Diego, United States : Society of Photo-Optical Instrumentation Engineers, fév. 2012. DOI : 10.1117/12.911396 (cf. p. 155).

[Gué09] JeanPierre Guédon. *The Mojette transform. Theory and applications*. Anglais. ISTE-WILEY, jan. 2009. 274 p. ISBN : 9781848210806. URL : http://hal.archives-ouvertes.fr/hal-00367681 (cf. p. 163).

[Had02] Jacques Hadamard. « Sur les problèmes aux dérivées partielles et leur signification physique ». In : *Princeton University Bulletin* 13.49–52 (1902) (cf. p. 47).

[Ham50] Richard W. Hamming. « Error Detecting and Error Correcting Codes ». In : *Bell System Technical Journal* 29.2 (avr. 1950), p. 147–160. ISSN : 1538-7305. DOI : 10.1002/j.1538-7305.1950.tb00463.x (cf. p. 22–24, 28).

[HL11] Martin Hilbert, Priscila López. « The World's Technological Capacity to Store, Communicate, and Compute Information ». In : *Science* 332.6025 (2011), p. 60–65. ISSN : 0036-8075. DOI : 10.1126/science.1200970. eprint : http://science.sciencemag.org/content/332/6025/60.full.pdf (cf. p. 9).

[HN15] Tom Haynes, Dave Noveck. *Network File System (NFS) Version 4 Protocol*. RFC 7530 (Proposed Standard). Internet Engineering Task Force, mar. 2015. URL : http://www.ietf.org/rfc/rfc7530.txt (cf. p. 121).

[Hou73] Godfrey N. Hounsfield. « Computerized transverse axial scanning (tomography) : Part 1. Description of system ». In : *The British Journal of Radiology* 46.552 (1973), p. 1016–1022. DOI : 10.1259/0007-1285-46-552-1016 (cf. p. 43, 44).

[Hua+12] Cheng Huang, Huseyin Simitci, Yikang Xu, Aaron Ogus, Brad Calder, Parikshit Gopalan, Jin Li, Sergey Yekhanin. « Erasure Coding in Windows Azure Storage ». In : *Presented as part of the 2012 USENIX Annual Technical Conference (USENIX ATC 12)*. Boston, MA : USENIX, 2012, p. 15–26. URL : https://www.usenix.org/conference/atc12/technical-sessions/presentation/huang (cf. p. 12, 32).

[Hun11] Robert Hundt. « Loop Recognition in C++/Java/Go/Scala ». In : *Proceedings of Scala Days 2011*. 2011, p. 38. URL : https://days2011.scala-lang.org/sites/days2011/files/ws3-1-Hundt.pdf (cf. p. 109).

[Hun14] Neil Hunt. *Cloud Migration, DevOps, and Distribtued Systems*. AWS re :Invent. Las Vegas, Nevada, USA, 12 nov. 2014. URL : http://fr.slideshare.net/AmazonWebServices/ent209-netflix-cloud-migration-devops-and-distributed-systems-aws-reinvent-2014 (cf. p. 10).

[Int15] Intel® Corporation. *Intel®Intelligent Storage Acceleration Library (Intel®ISA-L) Open Source Version API Reference Manual - Version 2.14*. Juil. 2015. URL : https://01.org/sites/default/files/documentation/isa-l_open_src_2.14.pdf (cf. p. 12, 95, 110, 116).

[Int98] Intel® Corporation. *Using the RDTSC Instruction for Performance Monitoring*. Rapp. tech. Juil. 1998. URL : https://www.ccsl.carleton.ca/~jamuir/rdtscpm1.pdf (cf. p. 111).

[Kat78] Myron B. Katz. *Questions of uniqueness and resolution in reconstruction from projections*. T. 26. Lecture Notes in Biomathematics. Springer-Verlag Berlin Heidelberg, 1978, p. 175. ISBN : 978-3-642-45507-0. DOI : 10.1007/978-3-642-45507-0 (cf. p. 42, 52, 62–64, 66, 67, 140–142, 149, 151, 162).

[Kle86] Steve R. Kleiman. « Vnodes : An Architecture for Multiple File System Types in Sun UNIX ». In : *USENIX Summer*. T. 86. 1986, p. 238–247. URL : http://www.solarisinternals.com/si/reading/vnode.pdf (cf. p. 122).

[Kra85] Rich Krajewski. « Multiprocessing : an Overview ». In : *Byte - The Small Systems Journal*. T. 10. 5. Mc Graw-Hill Publication, mai 1985, p. 171–172 (cf. p. 94).

[KS06] Andrew Kingston, Imants D. Svalbe. « Projective transforms on periodic discrete image arrays ». In : *Advances in Imaging and Electron Physics* 139 (déc. 2006), p. 75–177. DOI : 10.1016/S1076-5670(05)39002-1 (cf. p. 55).

[Lac+09] Jerome Lacan, Vincent Roca, Jani Peltotalo, Sami Peltotalo. *Reed-Solomon Forward Error Correction (FEC) Schemes*. IETF RFC 5510. Avr. 2009. DOI : 10.17487/rfc5510 (cf. p. 32, 35).

[LF04] Jérôme Lacan, Jérôme Fimes. « Systematic MDS erasure codes based on Vandermonde matrices ». In : *Communications Letters* 8.9 (juil. 2004), p. 570–572. ISSN : 1089-7798. DOI : 10.1109/LCOMM.2004.833807 (cf. p. 106).

[LG14] Paul Luse, Kevin Greenan. *Swift Object Storage : Adding Erasure Code*. Tutoriel présenté sous forme de diapositives. Sept. 2014. URL : http://www.snia.org/sites/default/files/Luse_Kevin_SNIATutorialSwift_Object_Storage2014_final.pdf (cf. p. 12).

[LMC94] Darrell D. E. Long, Bruce R. Montague, Luis-Felipe Cabrera. « Swift/RAID : A Distributed RAID System ». In : *Computing Systems* 7.3 (juin 1994), p. 333–359. ISSN : 0895-6340. URL : http://dl.acm.org/citation.cfm?id=198008.198011 (cf. p. 120).

[Lub+97] Michael G. Luby, Michael Mitzenmacher, M. Amin Shokrollahi, Daniel A. Spielman, Volker Stemann. « Practical Loss-resilient Codes ». In : *Proceedings of the Twenty-ninth Annual ACM Symposium on Theory of Computing*. STOC '97. El Paso, Texas, USA : ACM, 1997, p. 150–159. ISBN : 0-89791-888-6. DOI : 10.1145/258533.258573 (cf. p. 36).

[Lub02] Michael Luby. « LT codes ». In : *Proceedings of the 43rd Annual IEEE Symposium on Foundations of Computer Science*. 2002, p. 271–280. DOI : 10.1109/SFCS.2002.1181950 (cf. p. 33).

[Mat+07] Avantika Mathur, Mingming Cao, Suparna Bhattacharya, Andreas Dilger, Alex Tomas, Laurent Vivier. « The new ext4 filesystem : current status and future plans ». In : *Proceedings of the Linux symposium*. T. 2. 2007, p. 21–33 (cf. p. 126).

[MF93] František Matúš, Jan Flusser. « Image representation via a finite Radon transform ». In : *Transactions on Pattern Analysis and Machine Intelligence* 15.10 (oct. 1993), p. 996–1006. ISSN : 0162-8828. DOI : 10.1109/34.254058 (cf. p. 42, 52–56).

[Min68] Hermann Minkowski. *Geometrie der zahlen*. T. 40. Leipzig Teubner, 1968. 282 p. (cf. p. 49).

[MS06] Florence J. MacWilliams, Neil J.A. Sloane. *The Theory of Error-Correcting Codes*. 12th. T. 16. North-Holland Mathematical Library, 2006. ISBN : 0-444-85193-3 (cf. p. 31).

[Mur+14] Subramanian Muralidhar, Wyatt Lloyd, Sabyasachi Roy, Cory Hill, Ernest Lin, Weiwen Liu, Satadru Pan, Shiva Shankar, Viswanath Sivakumar, Linpeng Tang, Sanjeev Kumar. « f4 : Facebook's Warm BLOB Storage System ». In : *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*. Broomfield, CO, USA : USENIX Association, oct. 2014, p. 383–398. ISBN : 978-1-931971-16-4. URL : https://www.usenix.org/conference/osdi14/technical-sessions/presentation/muralidhar (cf. p. 126).

[Mut+02] Athicha Muthitacharoen, Robert Morris, Thomer M. Gil, Benjie Chen. « Ivy : A Read/Write Peer-to-peer File System ». In : *SIGOPS Operating Systems Review* 36.SI (déc. 2002), p. 31–44. ISSN : 0163-5980. DOI : 10.1145/844128.844132 (cf. p. 123).

[NKÉ06] Nicolas Normand, Andrew Kingston, Pierre Évenou. « A geometry driven reconstruction algorithm for the Mojette transform ». In : *DGCI*. T. 4245. LNCS. Springer Berlin Heidelberg, 2006, p. 122–133. ISBN : 978-3-540-47651-1. DOI : 10.1007/11907350_11 (cf. p. 13, 42, 62–64, 72–74, 142).

[Nor+09] Nicolas Normand, Imants Svalbe, Pierre Evenou, Andrew Kingston. « Inverse Mojette transform algorithms ». In : *The Mojette transform. Theory and applications*. ISTE-WILEY, jan. 2009. Chap. 5. ISBN : 9781848210806. URL : http://hal.archives-ouvertes.fr/hal-00367681 (cf. p. 64).

[Nor+10] Nicolas Normand, Imants Svalbe, Benoît Parrein, Andrew Kingston. « Erasure Coding with the Finite Radon Transform ». In : *Proceedings of the Wireless Communications and Networking Conference (WCNC)*. IEEE, avr. 2010. DOI : 10.1109/WCNC.2010.5506385 (cf. p. 13, 42, 52, 58–61, 162).

[Nor+96] Nicolas Normand, JeanPierre Guédon, Olivier Philippé, Dominique Barba. « Controlled redundancy for image coding and high-speed transmission ». In : *Proceedings of SPIE*. T. 2727. Visual Communications et Image Processing'96. Orlando, FL, mar. 1996, p. 1070–1081. DOI : 10.1117/12.233180 (cf. p. 63, 64, 73, 146).

[OD12] Frederique Oggier, Anwitaman Datta. *Coding Techniques for Repairability in Networked Distributed Storage Systems*. Notes de cours. International Conference on Distributed Computing et Networking, ICDCN 2012, 18 sept. 2012. URL : http://pdcc.ntu.edu.sg/sands/CodingForNetworkedStorage/pdf/longsurvey.pdf (cf. p. 9, 12, 120).

[OM09] Owen O'Malley, Arun C. Murthy. *Hadoop Sorts a Petabyte in 16.25 Hours and a Terabyte in 62 Seconds*. Yahoo ! Developer Network Blog. 11 mai 2009. URL : https://developer.yahoo.com/blogs/hadoop/hadoop-sorts-petabyte-16-25-hours-terabyte-62-422.html (cf. p. 125).

[Ope10] OpenFEC. *OpenFEC.org*. ISAE, INRIA. 30 sept. 2010. URL : http://openfec.org/ (cf. p. 110).

[Par+12] Benoît Parrein, Nicolas Normand, Majd Ghareeb, Giulio D'Ippolito, Federica Battisti. « Finite Radon coding for content delivery over hybrid client-server and P2P architecture ». In : *5th International Symposium on Communications Control and Signal Processing (ISCCSP)*. Mai 2012. DOI : 10.1109/ISCCSP.2012.6217794 (cf. p. 58).

[Par+15] Benoît Parrein, Jérôme Lacan, Nicolas Normand, **Dimitri Pertin**, Jonathan Detchart, Alexandre Van Kempen. « FEC4Cloud : a research project promoting erasure coding for Cloud storage architectures ». In : *Rendez-vous de la Recherche et de l'Enseignement de la Sécurité des Systèmes d'Information (RESSI)*. Troyes, France, mai 2015. URL : https://hal.archives-ouvertes.fr/hal-01156462 (cf. p. 112).

[Par01] Benoît Parrein. « Multiple description coding by Mojette tranform ». Français. Thèse de doct. Université de Nantes, nov. 2001. URL : https://tel.archives-ouvertes.fr/tel-00300613 (cf. p. 67, 68, 76).

[PD03] James S. Plank, Ying Ding. *Note : Correction to the 1997 Tutorial on Reed-Solomon Coding*. Rapp. tech. CS-03-504. University of Tennessee, avr. 2003 (cf. p. 106).

[Per+] **Dimitri Pertin**, Alexandre Van Kempen, Bastien Confais, Didier Féron, Nicolas Normand, Benoît Parrein. « RozoFS : an erasure-coded distributed file system for I/O intensive workloads ». Article soumis à la revue *ACM Transactions on Storage*, décembre 2015 (cf. p. 163).

[Per+12] **Dimitri Pertin**, Giulio D'Ippolito, Nicolas Normand, Benoît Parrein. « Spatial Implementation for Erasure Coding by Finite Radon Transform ». In : *International Symposium on signal, Image, Video and Communication*. Valenciennes, France, juil. 2012. URL : https://hal.archives-ouvertes.fr/hal-00716061 (cf. p. 58).

[Per+14] **Dimitri Pertin**, Sylvain David, Pierre Évenou, Benoît Parrein, Nicolas Normand. « Distributed File System Based on Erasure Coding for I/O-Intensive Applications ». In : *Proceedings of the 4th International Conference on Cloud Computing and Services Science*. CLOSER. Barcelona, Spain, avr. 2014, p. 451–456. DOI : 10.5220/0004960604510456 (cf. p. 163).

[Per+15a] **Dimitri Pertin**, Didier Féron, Alexandre Van Kempen, Benoît Parrein. « Performance evaluation of the Mojette erasure code for fault-tolerant distributed hot data storage ». In : *CoRR* abs/1504.07038 (avr. 2015). URL : http://arxiv.org/abs/1504.07038 (cf. p. 112).

[Per+15b] **Dimitri Pertin**, Alexandre Van Kempen, Benoît Parrein, Nicolas Normand. « Comparison of RAID-6 Erasure Codes ». In : *The third Sino-French Workshop on Information and Communication Technologies*. SIFWICT. Nantes, France, juin 2015. URL : https://hal.archives-ouvertes.fr/hal-01162047 (cf. p. 94, 162).

[PG14] James S. Plank, Kevin M. Greenan. *Jerasure : A library in C facilitating erasure coding for storage applications – version 2.0*. Rapp. tech. UT-EECS-14-721. University of Tennessee, 2014 (cf. p. 110).

[PGK88] David A. Patterson, Garth Gibson, Randy H. Katz. « A Case for Redundant Arrays of Inexpensive Disks (RAID) ». In : *SIGMOD Record* 17.3 (juin 1988), p. 109–116. ISSN : 0163-5808. DOI : 10.1145/971701.50214 (cf. p. 82, 94).

[Phi98] Olivier Philippe. « Représentation d'images pour le codage conjoint source-canal sur réseaux à qualité de service ». Thèse de doct. IRESTE, Université de Nantes, nov. 1998. URL : https://tel.archives-ouvertes.fr/tel-00300613 (cf. p. 66, 141, 147).

[Pla+09] James S. Plank, Jianqiang Luo, Catherine D. Schuman, Lihao Xu, Zooko Wilcox-O'Hearn. « A Performance Evaluation and Examination of Open-source Erasure Coding Libraries for Storage ». In : *Proccedings of the 7th Conference on File and Storage Technologies*. FAST '09. San Francisco, California : USENIX Association, 2009, p. 253–265. URL : http://dl.acm.org/citation.cfm?id%20=%201525908.1525927 (cf. p. 82, 96, 99, 120).

[PN14a] **Dimitri Pertin**, Nicolas Normand. « Re-projection without Reconstruction ». In : *9ème Journées du Groupe de travail de Géométrie Discrète, Reims Image 2014*. Reims, France, nov. 2014, p. 43. URL : https://hal.archives-ouvertes.fr/hal-01164872 (cf. p. 164).

[PNB01] Benoît Parrein, Nicolas Normand, Dominique Barba. « Description multiple par transformation de Radon discrète exacte ». In : *18ème Colloque sur le traitement du signal et des images*. Groupe d'Etudes du Traitement du Signal et des Images (GRETSI). 2001 (cf. p. 67).

[PPN14] **Dimitri Pertin**, Benoît Parrein, Normand Nicolas. *The Mojette erasure code for distributed file systems*. Session poster. Amsterdam, The Netherlands : EuroSys'14 (Session poster), avr. 2014 (cf. p. 112).

[PX06] James S. Plank, Lihao Xu. « Optimizing Cauchy Reed-Solomon Codes for Fault-Tolerant Network Storage Applications ». In : *Fifth International Symposium on Network Computing Applications. NCA'06 :* Cambridge, MA : IEEE, juil. 2006, p. 173–180. DOI : 10.1109/NCA.2006.43 (cf. p. 107).

[Rab89] Michael O. Rabin. « Efficient Dispersal of Information for Security, Load Balancing, and Fault Tolerance ». In : *Journal of the ACM* 36.2 (avr. 1989), p. 335–348. ISSN : 0004-5411. DOI : 10.1145/62044.62050 (cf. p. 137, 155).

[Rad17] Johann Radon. « Über die Bestimmung von Funktionen durch ihre Integralwerte längs gewisser Mannigfaltigkeiten ». In : *Verhandlungen der Königlich-Sächsischen Akademie der Wissenschaften zu Leipzig, Mathematisch-Physische Klasse* 69 (1917), p. 262–277 (cf. p. 13, 14, 19, 42, 43, 45–47, 49–62, 69).

[RD01] Antony Rowstron, Peter Druschel. « Storage Management and Caching in PAST, a Large-scale, Persistent Peer-to-peer Storage Utility ». In : *Proceedings of the Eighteenth ACM Symposium on Operating Systems Principles*. SOSP '01. Banff, Alberta, Canada : ACM, 2001, p. 188–201. ISBN : 1-58113-389-8. DOI : 10.1145/502034.502053 (cf. p. 120).

[Riz97a] Luigi Rizzo. « Effective Erasure Codes for Reliable Computer Communication Protocols ». In : *SIGCOMM Computer Communication Review* 27.2 (avr. 1997), p. 24–36. ISSN : 0146-4833. DOI : 10.1145/263876.263881 (cf. p. 104, 106).

[Ros70] Azriel Rosenfeld. « Connectivity in Digital Pictures ». In : *Journal of the ACM* 17.1 (jan. 1970), p. 146–160. ISSN : 0004-5411. DOI : 10.1145/321556.321570 (cf. p. 49).

[Ros79] Azriel Rosenfeld. « Digital Topology ». In : *The American Mathematical Monthly* 86.8 (1979), p. 621–630. ISSN : 00029890, 19300972. URL : http://www.jstor.org/stable/2321288 (cf. p. 49).

[RS60] Irvin S. Reed, Gustave Solomon. « Polynomial Codes Over Certain Finite Fields ». In : *Journal of the Society for Industrial and Applied Mathematics* 8.2 (1960), p. 300–304. DOI : 10.1137/0108018 (cf. p. 12, 23, 33, 35, 42, 60, 94, 105).

[Sat+13] Maheswaran Sathiamoorthy, Megasthenis Asteris, Dimitris Papailiopoulos, Alexandros G. Dimakis, Ramkumar Vadali, Scott Chen, Dhruba Borthakur. « XORing Elephants : Novel Erasure Codes for Big Data ». In : *Proceedings of the 39th international conference on Very Large Data Bases*. T. 6. 5. VLDB Endowment, mar. 2013, p. 325–336. URL : http://dl.acm.org/citation.cfm?id=2535573.2488339 (cf. p. 110, 116, 156).

[Ser+05a] Myriam Servieres, Jérôme Idier, Nicolas Normand, JeanPierre Guedon. « Conjugate gradient Mojette reconstruction ». In : *Proceedings of SPIE*. T. 5747. Medical Imaging 2005 : Image Processing, 2067. Mai 2005, p. 2067–2074. DOI : 10.1117/12.593399 (cf. p. 64).

[Sha48] Claude E. Shannon. « A Mathematical Theory of Communication ». In : *Bell System Technical Journal* 27.3 (1948), p. 379–423. ISSN : 1538-7305. DOI : 10.1002/j.1538-7305.1948.tb01338.x (cf. p. 19, 22–26).

[Sho06] Amin Shokrollahi. « Raptor codes ». In : *IEEE Transactions on Information Theory* 52.6 (juin 2006), p. 2551–2567. ISSN : 0018-9448. DOI : 10.1109/TIT.2006.874390 (cf. p. 33).

[Shv+10] Konstantin V. Shvachko, Hairong Kuang, Sanjay Radia, Robert Chansler. « The Hadoop Distributed File System ». In : *Proceedings of the 26th Symposium on Mass Storage Systems and Technologies (MSST)*. Incline Villiage, NV, USA : IEEE, mai 2010. DOI : 10.1109/MSST.2010.5496972 (cf. p. 111, 125).

[Sin64] Richard C. Singleton. « Maximum Distance q-nary Codes ». In : *IEEE Transactions on Information Theory* 10.2 (avr. 1964), p. 116–118. ISSN : 0018-9448. DOI : 10.1109/TIT.1964.1053661 (cf. p. 31).

[SL10] Alexandre Soro, Jérôme Lacan. « FNT-Based Reed-Solomon Erasure Codes ». In : *Proceedings of the 7th Consumer Communications and Networking Conference (CCNC)*. IEEE, jan. 2010. DOI : 10.1109/CCNC.2010.5421749 (cf. p. 36, 38).

[Slo+96] Neil J. A. Sloane, Simon Plouffe, Jonathan M. Borwein, Robert M. Corless. « The encyclopedia of integer sequences ». In : *SIAM Review* 38.2 (1996), p. 333–336 (cf. p. 68).

[SM08] Konstantin V. Shvachko, Arun C. Murthy. *Scaling Hadoop to 4000 nodes at Yahoo !* Yahoo ! Developer Network Blog. 30 sept. 2008. URL : https://developer.yahoo.com/blogs/hadoop/scaling-hadoop-4000-nodes-yahoo-410.html (cf. p. 125).

[Sto+01] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, Hari Balakrishnan. « Chord : A Scalable Peer-to-peer Lookup Service for Internet Applications ». In : *SIGCOMM Computer Communication Review - Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications* 31.4 (juin 2001), p. 149–160. ISSN : 0146-4833. DOI : 10.1145/964723.383071 (cf. p. 122).

[TB14] Andrew S. Tanenbaum, Herbert Bos. *Modern Operating Systems*. 4th. Upper Saddle River, NJ, USA : Prentice Hall Press, 2014. ISBN : 013359162X, 9780133591620 (cf. p. 80).

[Tou+11] Pierre Ugo Tournoux, Emmanuel Lochin, Jérôme Lacan, Amine Bouabdallah, Vincent Roca. « On-the-Fly Erasure Coding for Real-Time Video Applications ». In : *IEEE Transactions on Multimedia* 13.4 (août 2011), p. 797–812. ISSN : 1520-9210. DOI : 10.1109/TMM.2011.2126564 (cf. p. 32).

[TS06] Andrew S. Tanenbaum, Maarten Van Steen. *Distributed Systems : Principles and Paradigms*. 2nd. Upper Saddle River, NJ, USA : Prentice Hall PTR, 2006. ISBN : 0130888931 (cf. p. 9).

[Tur66] Richard L. Turner. *Inverse of the Vandermonde matrix with applications*. Rapp. tech. TN D-3547. Cleveland, Ohio, United States of America : NASA Glenn Research Center at Lewis Field, août 1966 (cf. p. 61).

[VRG04] Pierre Verbert, Vincent Ricordel, Jeanpierre Guédon. « Analysis of Mojette transform projections for an efficient coding ». In : *Workshop on Image Analysis for Multimedia Interactive Services (WIAMIS)*. Lisboa, Portugal, avr. 2004. URL : https://hal.archives-ouvertes.fr/hal-00451338 (cf. p. 68).

[Wei+06] Sage A. Weil, Scott A. Brandt, Ethan L. Miller, Darrell D. E. Long, Carlos Maltzahn. « Ceph : A scalable, high-performance distributed file system ». In : *Proc. OSDI 2006*. Seattle, Washington : USENIX Association, 2006, p. 307–320. ISBN : 1-931971-47-1. URL : http://dl.acm.org/citation.cfm?id%20=%201298455.1298485 (cf. p. 12, 120, 121, 123, 126).

[Wei07] Sage A. Weil. « Ceph : reliable, scalable, and high-performance distributed storage ». Thèse de doct. University of California Santa Cruz, déc. 2007 (cf. p. 136, 164).

[WK02] Hakim Weatherspoon, John Kubiatowicz. « Erasure Coding Vs. Replication : A Quantitative Comparison ». In : *Revised Papers from the First International Workshop on Peer-to-Peer Systems*. IPTPS 01. London, UK : Springer-Verlag, 2002, p. 328–338. ISBN : 3-540-44179-4. URL : http://dl.acm.org/citation.cfm?id=646334.687814 (cf. p. 12, 120).

[WW08] Zooko Wilcox-O'Hearn, Brian Warner. « Tahoe : The Least-authority Filesystem ». In : *Proceedings of the 4th ACM International Workshop on Storage Security and Survivability*. StorageSS '08. Alexandria, Virginia, USA : ACM, 2008, p. 21–26. ISBN : 978-1-60558-299-3. DOI : 10.1145/1456469.1456474 (cf. p. 121).

[ZHX12] Fenghao Zhang, Jianzhong Huang, Changsheng Xie. « Two Efficient Partial-Updating Schemes for Erasure-Coded Storage Clusters ». In : *7th International Conference on Networking, Architecture and Storage (NAS)*. IEEE, juin 2012, p. 21–30. DOI : 10.1109/NAS.2012.7 (cf. p. 96).

[Zwa15] Willy Zwaenepoel. *Analytics on Graphs with a Trillion Edges*. Workshop on Storage and Data Analytics, WOS 5. Technicolor, Cesson-Sévigné, France, 19 nov. 2015. URL : https://project.inria.fr/epfl-Inria/files/2015/07/inria.pdf (cf. p. 10).

UNIVERSITE BRETAGNE LOIRE

UNIVERSITÉ DE NANTES

Thèse de Doctorat

Dimitri PERTIN

Code à Effacement Mojette pour le Stockage Distribué

Mojette Erasure Code for Distributed Storage

Résumé

Les codes à effacement permettent de générer de la redondance de données numériques dans un système de stockage distribué. Cette redondance permet de restaurer une partie manquante des données en cas de panne.
L'avantage des codes est de réduire considérablement la quantité de redondance générée par rapport aux techniques classiques de réplication. Toutefois, cette réduction s'accompagne d'une complexité calculatoire significative, pénalisant les performances d'encodage et de décodage, ce qui limite leur utilisation aux données froides.
Dans cette thèse, nous nous intéressons à l'utilisation de la transformation Mojette afin de fournir un code à effacement performant, adapté aux données chaudes. Le code qui en résulte nécessite cependant plus de redondance par rapport aux codes classiques.
La première contribution de ces travaux de thèse traite de la conception d'une version systématique du code à effacement Mojette. Cette version a l'avantage d'augmenter significativement les performances du code, tout en réduisant la quantité de redondance nécessaire.
La seconde contribution s'intéresse à l'intégration de cette solution au sein du système de fichiers distribué RozoFS. Cette contribution permet au système d'assurer un service continu en cas de panne, tout en étant capable de gérer les données chaudes avec deux fois moins de données par rapport aux systèmes basés sur la réplication.
Un troisième axe de recherche se focalise sur la conception d'une méthode distribuée pour générer de nouveaux symboles de mots de code Mojette. Cette technique participe à la restauration d'un seuil de redondance du système de stockage.

Mots clés

Code à effacement, transformée Mojette, stockage distribué, tolérance aux pannes.

Abstract

Erasure codes can generate data redundancy in distributed storage systems. This redundancy can be used to recover missing data in case of a failure. Codes have the benefit of reducing the generated amount of redundancy drastically, compared to plain data replication. However, this reduction is combined with a significant computational complexity, which penalizes encoding and decoding performances, and limits the use of coding to cold data.
In this thesis, we focus on the use of the Mojette transform as an effective erasure code, adapted to hot data. The resulting code requires more redundancy than classical codes though.
The first contribution of this research work deals with the design of a systematic version of the Mojette erasure code. This version provides better performances while reducing the required amount of redundancy.
The second contribution covers the integration of this solution in the distributed file system RozoFS. This integration enables the system to provide a continuous service despite failures, while being able to manage hot data with half the volume of data compared to replication-based systems.
A third research focus addresses the design of a distributed method to compute extra Mojette codeword symbols. This method contributes to restore a redundancy threshold in the storage system.

Key Words

Erasure code, Mojette transform, distributed storage, fault tolerance.