Optimisation des coûts de reconstruction dans un système de stockage distribué*

Elyas El Idrissi[1,2], Cheick Omar Diallo[1], Joris Carrier[2], and Gil Utard[1]

[1]MIS, Université de Picardie Jules Verne
[2]Ugloo SAS, Versailles

Résumé
Les systèmes de stockage distribué utilisent largement les codes de *Reed–Solomon* pour assurer la durabilité des données, mais cette fiabilité s'accompagne de coûts de reconstruction élevés. De nombreuses approches ont été proposées dans la littérature pour réduire ces coûts. Parmi celles-ci, le *piggybacking* permet de diminuer le volume de données lues et transférées lors des réparations sans augmenter la redondance, en ciblant principalement la reconstruction des blocs de données dans la plupart des schémas existants. Les codes à récupération locale (LRC) introduisent des mécanismes de réparation locale pour un sous-ensemble de fragments au prix d'un surcoût de stockage. Dans cet article, nous proposons *un schéma hybride* combinant piggybacking et récupération locale. En associant un piggybacking de type Hitchhiker à une organisation de parités locales inspirée des LRC et appliquée aux seules parités globales, cette approche réduit les coûts moyens de jusqu'à 60 % tout en maintenant un compromis maîtrisé entre efficacité de stockage et résilience.

Mots-clés : Stockage distribué, Codes d'effacement, Coût de reconstruction, Reed–Solomon, Piggybacking, LRC

1. Introduction

Les systèmes de stockage distribué à grande échelle constituent aujourd'hui un pilier essentiel des infrastructures numériques modernes et reposent largement sur les codes correcteurs d'effacement, en particulier les codes de Reed–Solomon [9], pour garantir la durabilité des données. Ces codes offrent une excellente efficacité de stockage et une forte résilience face aux défaillances, mais leur coût de reconstruction demeure élevé. La réparation d'un fragment perdu nécessite en effet de lire et de transférer un volume équivalent à l'ensemble des données utiles de la bande. Ce coût, bien identifié dans la littérature et observé en environnement industriel, engendre une charge importante sur le réseau et les disques, qui limite l'évolutivité et la viabilité des infrastructures de stockage distribué, notamment chez Google, Facebook et Ugloo [7, 5].
De nombreuses approches ont été proposées pour réduire ce coût [1, 6, 11, 2]. Nos travaux se concentrent sur deux d'entre elles : le piggybacking [8] et les codes à récupération locale (LRC) [3]. Le piggybacking réduit le volume de données lu lors des réparations sans surcoût de stockage, mais ses gains concernent les blocs de données dans la plupart des schémas existants.

*. Ce projet est soutenu par le fond FEDER Emile BlockStoreNet de la région Hauts-de-France.

Les LRC introduisent des parités locales permettant de réduire le coût de réparation de tout ou partie des fragments selon les constructions, mais augmentent le surcoût de stockage.
Nous proposons *Hitchhiker–LRC*, un schéma hybride qui combine le piggybacking de Hitchhiker pour la réparation des fragments de données et une organisation de parités locales inspirée des LRC, appliquée aux seules parités globales. Ce schéma réduit les coûts moyens et les pires cas de reconstruction. Nos simulations montrent qu'il permet de diminuer le coût moyen de reconstruction jusqu'à 60 % tout en préservant un compromis équilibré entre efficacité de stockage et résilience.
La suite de cet article est organisée comme suit. La section 2 présente les principes des codes Reed–Solomon et les limitations associées au coût de reconstruction dans les systèmes de stockage distribué. La section 3 décrit le système DeepTorrent et les contraintes industrielles qui encadrent notre étude. La section 4 présente les approches de l'état de l'art, en particulier le piggybacking et les codes à récupération locale. La section 5 introduit le schéma Hitchhiker–LRC et détaille sa construction ainsi que les mécanismes de reconstruction. La section 6 décrit la méthodologie de simulation et analyse les résultats obtenus. Enfin, la section 7 conclut l'article et discute les perspectives d'intégration et d'évaluation dans un système de stockage distribué réel.

2. Reed–Solomon et limitations

2.1. Principe

Un bloc de données de taille B est divisé en s fragments de taille $\frac{B}{s}$. On calcule ensuite r fragments de parité, obtenant au total $n = s + r$ fragments répartis sur les disques du cluster (figure 1). Les codes Reed–Solomon [9] sont des codes à distance séparable maximale (MDS) [12] : tout sous-ensemble de s fragments parmi n permet de reconstruire la donnée originale. Un exemple d'implémentation de Reed–Solomon est disponible avec la bibliothèque GFLIB. Une version optimisée est celle proposée par Intel, ISA-L.
Le facteur de redondance est défini par $\rho = \frac{n}{s}$ et l'efficacité de stockage par $\frac{s}{n}$. À efficacité de stockage égale, les codes d'effacement offrent une tolérance aux pannes bien supérieure à la réplication. Par exemple, un schéma $(4, 12)$ tolère jusqu'à 12 pertes tout en conservant la même efficacité qu'une réplication $(1, 3)$, qui ne tolère qu'au plus 3 pertes ($\rho = 3$).

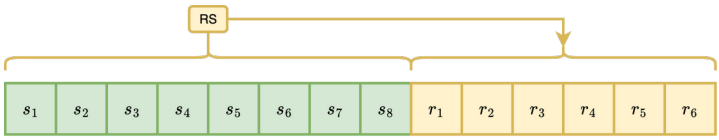


FIGURE 1 – Schéma de construction de Reed–Solomon pour $s = 8$, $r = 6$.

2.2. Coût de reconstruction

Dans un code Reed–Solomon, la réparation d'un fragment perdu nécessite la lecture de s fragments disponibles, soit un volume équivalent à l'ensemble des données utiles de la bande. Ce coût constitue une source importante de charge réseau et disque.
Dans la suite, le coût de reconstruction est mesuré comme le nombre de fragments lus pour réparer les fragments perdus et normalisé par le coût Reed–Solomon, exprimé en pourcentage, tel que $c_{norm} = \frac{c_{lu}}{s} \times 100$. Ainsi, Reed–Solomon à un coût normalisé de 100 %.

2.3. Limitations observées en production

À grande échelle, ce coût de reconstruction devient limitant. Le cluster Hadoop de Facebook compte plus de 3000 nœuds et la réparation des fragments génère un trafic réseau journalier médian supérieur à 180 To, soit environ 10 à 20 % du trafic interne total [7]. Une augmentation significative de la proportion de données encodées avec Reed–Solomon pourrait ainsi saturer le réseau interne. Par ailleurs, la majorité des incidents correspond à la perte d'un seul fragment : 98.08 % chez Facebook, et 99.2 % dans les clusters de stockage de Google qui exploitent plus de 1.7 million de disques pour plus de 10^{12} bandes d'encodage [7, 5].

3. DeepTorrent

DeepTorrent est une solution de stockage distribué développée par Ugloo, déployée en environnement industriel et servant de plateforme expérimentale aux travaux présentés. Le système expose une interface compatible S3 et s'appuie sur une extension du protocole BitTorrent pour orchestrer la distribution et la reconstruction des fragments. Les données sont fragmentées, encodées par des codes de Reed–Solomon puis réparties sur les disques du cluster suivant un modèle pair-à-pair.

La résilience repose sur une détection distribuée des défaillances et sur une reconstruction exploitant les propriétés MDS des codes utilisés. La stratégie est proactive : toute indisponibilité déclenche la régénération du fragment concerné afin de limiter le fonctionnement en mode dégradé et le risque de pertes corrélées.

Les optimisations étudiées sont contraintes par les caractéristiques de l'infrastructure cible. Celle-ci comprend trois serveurs comportant chacun entre 6 et 12 disques, ce qui fixe les paramètres d'encodage admissibles. Le système doit tolérer la perte simultanée d'un serveur complet et d'un disque supplémentaire, sans perte de données ni interruption de service. Enfin, le facteur de redondance est limité à $\rho = \frac{n}{s} \in [1.6, 1.8]$ pour des raisons de coût de stockage.

4. État de l'art

Plusieurs approches ont été proposées dans la littérature pour réduire les coûts de reconstruction [1, 6, 11, 2]. Nous nous intéressons ici à deux techniques déployées en production à grande échelle : le piggybacking et les codes à récupération locale (LRC).

4.1. Piggybacking

Le piggybacking [8] vise à réduire le volume de données à lire lors d'une réparation sans augmenter la redondance. Le principe consiste à ajouter des combinaisons linéaires supplémentaires, appelées *piggybacks*, sur certains fragments de parité issus d'un code MDS sous-jacent.

Le schéma Hitchhiker [7] applique cette idée aux codes Reed–Solomon en organisant les fragments en deux sous-bandes, notées a et b. Les parités de la sous-bande a conservent la structure Reed–Solomon, tandis que celles de la sous-bande b portent également les piggybacks de la sous-bande a (figure 2). Cette organisation réduit significativement le coût de réparation des fragments de données. La reconstruction des parités ou de pertes corrélées de données nécessite un décodage complet Reed–Solomon.

4.2. Codes à récupération locale (LRC)

Les codes à récupération locale (LRC) [3] adoptent une stratégie différente en introduisant des fragments de parité supplémentaires dits *locaux*. Les fragments sont partitionnés en groupes, chacun produisant une parité locale permettant de reconstruire un fragment manquant à partir

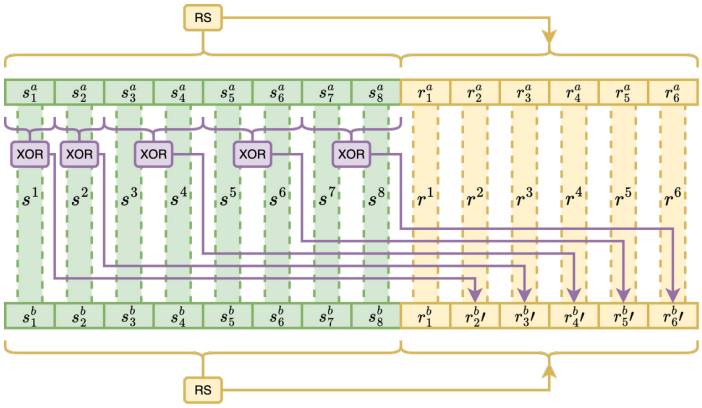


FIGURE 2 – Schéma Hitchhiker pour $s = 8$ et $r = 6$.

On note $r_6^b{}'$ la parité de la sous-bande b après application du piggybacking tel que : $r_6^b{}' = r_6^b \oplus s_7^a \oplus s_8^a$. En cas de perte du fragment s_8, la reconstruction avec Hitchhiker est telle que : $(s_8^b, r_6^b) = \mathrm{RS}^{-1}(s_1^b, \ldots, s_7^b, r_1^b)$; $s_8^a = r_6^b{}' \oplus r_6^b \oplus s_7^a$; $s_8 = \mathrm{CONCAT}(s_8^a, s_8^b)$. Le coût est $c_{norm} = \frac{5}{8} \times 100 = 62.5\,\%$, chaque sous-fragment ayant une taille égale à la moitié de celle d'un fragment.

des seuls fragments du groupe.
Dans le schéma Uniform Cauchy LRC [5], les fragments sont d'abord générés par un code Reed–Solomon, puis répartis en ℓ groupes. Chaque groupe produit une parité locale par XOR, permettant de réparer un fragment en ne lisant qu'environ n/ℓ fragments (figure 3). Des pertes multiples ou corrélées dans un même groupe empêchent la réparation locale et nécessitent un décodage complet Reed–Solomon.
Plusieurs variantes de LRC existent, notamment *Azure-LRC* [4] et *Xorbas-LRC* [10]. Ces constructions présentent toutefois des limitations de symétrie ou de flexibilité pour les configurations de grande taille [5].

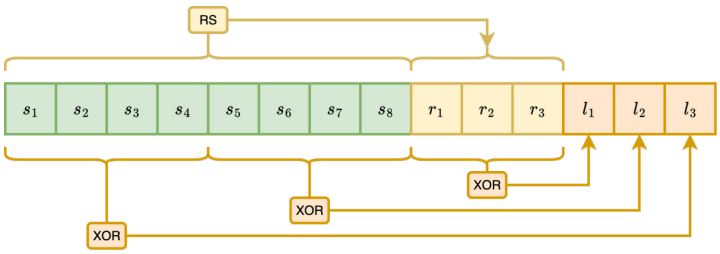


FIGURE 3 – Schéma Uniform Cauchy LRC pour $s = 8$, $r = 3$ et $\ell = 3$.

En cas de perte du fragment s_1, la reconstruction avec Wide Uniform LRC est telle que : $s_1 = \bigoplus_{i=2}^{4} s_i \oplus l_3$. Le coût est $c_{norm} = \frac{4}{8} \times 100 = 50\,\%$.

5. Hybridation Hitchhiker–LRC

Nous proposons Hitchhiker–LRC, un schéma hybride combinant piggybacking et récupération locale afin de tirer parti des avantages des deux approches tout en atténuant leurs limites.

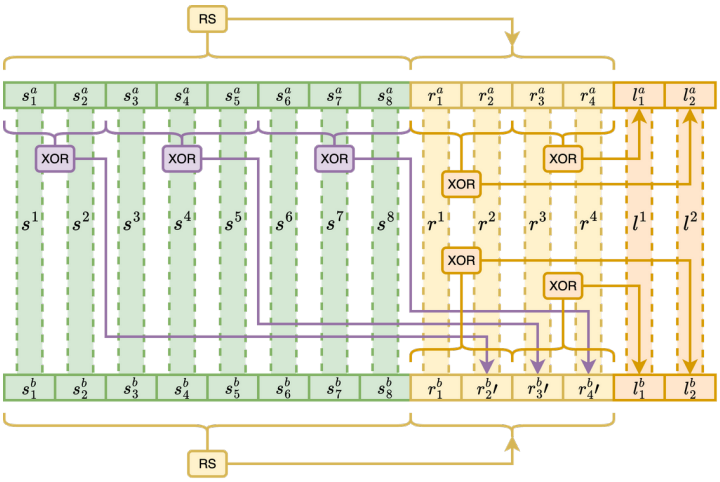
5.1. Motivation

Les codes Hitchhiker et les codes à récupération locale présentent des propriétés complémentaires. Hitchhiker optimise la réparation des fragments de données sans introduire de redondance supplémentaire, tandis que les LRC permettent de réduire le coût de réparation de fragments arbitraires en introduisant des parités locales. Par ailleurs, Hitchhiker repose sur un code

MDS sous-jacent dont il conserve les propriétés, tandis que la localité des LRC peut être appliquée sans altérer ni faire d'hypothèses sur le code de base.
Cette complémentarité permet d'hybrider les deux approches en appliquant la localité uniquement aux parités globales issues de Hitchhiker. Cette restriction limite l'augmentation de redondance par rapport à Uniform Wide LRC et réduit le coût des réparations de fragments de parité par rapport à Hitchhiker.

5.2. Construction et reconstruction

Le schéma est défini par les paramètres (s, r, ℓ), où s est le nombre de fragments de données, r le nombre de parités globales et ℓ le nombre de parités locales. La figure 4 illustre le principe.



En cas de perte du fragment s_8, la méthode de reconstruction est identique à celle de Hitchhiker avec un coût $c_{norm} = \frac{5.5}{8} \times 100 = 68.75\,\%$. En cas de perte du fragment r_4, la reconstruction avec la localité est telle que : $r_4^a = l_1^a \oplus r_3^a$; $r_4^{b\prime} = l_1^b \oplus r_3^{b\prime}$; $r_4 = \mathrm{CONCAT}(r_4^a, r_4^{b\prime})$. Le coût est $c_{norm} = \frac{2}{8} \times 100 = 25\,\%$, chaque sous-fragment ayant une taille égale à la moitié de celle d'un fragment.

FIGURE 4 – Schéma de construction de Hitchhiker–LRC pour $s = 8$, $r = 4$ et $\ell = 2$.

La construction comporte deux étapes : (1) les r parités globales sont générées selon Hitchhiker à partir des s fragments de données ; (2) ℓ parités locales sont construites en appliquant la localité aux seules parités globales, réparties en ℓ groupes produisant chacun une parité locale par XOR. Les fragments de données sont réparés par piggybacking, tandis que les parités sont réparées localement au sein de leur groupe. Lorsque ces optimisations ne sont pas applicables, la reconstruction est assurée par le code Reed–Solomon sous-jacent.

6. Évaluation par simulation

Nous avons développé un simulateur évaluant l'ensemble des combinaisons de pannes possibles pour un schéma, des paramètres et un nombre de pertes simultanées donnés. Les coûts de reconstruction sont calculés selon les mécanismes du schéma, puis agrégés sous forme de moyenne, extrêmes et distribution. Les schémas simulés sont Hitchhiker (HH), Uniform Cauchy LRC (LRC) et Hitchhiker–LRC (HHLRC). Le simulateur est implémenté en Python et disponible sur GITHUB [2].

2. https ://github.com/e1y4s/ec-cost-simulator

6.1. Contraintes et paramètres

Les paramètres d'encodage sont choisis pour satisfaire les contraintes matérielles, la tolérance aux pertes corrélées et les bornes sur ρ, tout en garantissant un surcoût de stockage identique entre les schémas. Nous fixons $n = 36$, correspondant au nombre maximal de disques dans l'architecture cible. Nous imposons $r \geq 13$, afin de tolérer la perte simultanée d'un serveur de 12 disques et d'un disque supplémentaire. Nous retenons $\rho \in \{1.64, 1.71, 1.80\}$, seules valeurs admissibles pour $n = 36$ dans l'intervalle considéré. Les coûts de reconstruction sont évalués pour $e \in \{1, 2, 3\}$ pertes simultanées.

6.2. Résultats

Les figures des résultats sont reportés en annexe. Pour $\rho = 1.80$, l'analyse exhaustive repose sur 36 cas pour une perte ($e = 1$), 630 pour deux pertes ($e = 2$) et 7140 pour trois pertes ($e = 3$).
Pour une perte unique, Hitchhiker présente un coût moyen normalisé de 74.3 %, Wide Uniform LRC de 55 %, et Hitchhiker–LRC de 40 %. Aucun scénario dans l'hybridation n'atteint le coût maximal car les pertes de parités globales sont réparées localement.
Pour deux pertes simultanées, Hitchhiker atteint 88.2 % en moyenne et Wide Uniform LRC 100 %, tandis que Hitchhiker–LRC descend à 68.4 %. La proportion de scénarios au coût maximal chute alors de 70.6 % avec Hitchhiker à 6.8 % avec l'hybridation.
Pour trois pertes simultanées, les coûts moyens sont respectivement de 95 % pour Hitchhiker, 100 % pour LRC et 83.9 % pour HHLRC. Même lorsque les opportunités d'optimisation diminuent, la localité appliquée aux seules parités globales continue donc de limiter la concentration des pires cas.
Ces résultats montrent que Hitchhiker–LRC surpasse Hitchhiker et Uniform Cauchy LRC dans l'ensemble des scénarios étudiés, avec des réductions du coût de reconstruction pouvant atteindre 60 %.

7. Conclusion

Nous avons proposé Hitchhiker–LRC, un schéma hybride combinant piggybacking et récupération locale pour réduire les coûts de reconstruction dans les systèmes de stockage distribué. Cette approche exploite la complémentarité entre Hitchhiker, qui optimise la réparation des fragments de données sans surcoût de stockage, et la localité appliquée aux parités globales pour réduire le coût de réparation des parités.
L'évaluation exhaustive par simulation, couvrant l'ensemble des combinaisons de pannes sous les contraintes industrielles d'Ugloo, montre que cette hybridation réduit significativement le coût moyen de reconstruction et limite fortement la fréquence des reconstructions nécessitant un décodage Reed–Solomon. À surcoût de stockage égal, Hitchhiker–LRC surpasse Hitchhiker et Uniform Cauchy LRC dans l'ensemble des configurations étudiées, avec des réductions du coût de reconstruction pouvant atteindre 60 %.
Une implémentation expérimentale du schéma est actuellement en cours d'intégration dans la solution de stockage distribué développée par Ugloo. Cette mise en œuvre permettra d'évaluer ultérieurement le comportement du schéma en conditions réelles, notamment son impact sur la charge réseau, les performances du système, les temps de réparation et la robustesse face aux pannes corrélées.

Bibliographie

1. Byers (J. W.), Luby (M. G.), Mitzenmacher (M.) et Ashutosh (M.). – A digital fountain approach to reliable distribution of bulk data. – In *Proceedings of the ACM SIGCOMM '98 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, *SIGCOMM '98*, SIGCOMM '98, p. 56–67, New York, NY, USA, 1998. Association for Computing Machinery.
2. Estrada-Galiñanes (V.), Miller (E.), Felber (P.) et Pâris (J.-F.). – Alpha entanglement codes : Practical erasure codes to archive data in unreliable environments. – In *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pp. 183–194, 2018.
3. Gopalan (P.), Huang (C.), Simitci (H.) et Yekhanin (S.). – On the locality of codeword symbols. *IEEE Transactions on Information Theory*, vol. 58, n11, 2012, pp. 6925–6934.
4. Huang (C.), Simitci (H.), Xu (Y.), Ogus (A.), Calder (B.), Gopalan (P.), Li (J.) et Yekhanin (S.). – Erasure coding in windows azure storage. – In *Proceedings of the 2012 USENIX Conference on Annual Technical Conference*, *USENIX ATC'12*, USENIX ATC'12, p. 2, USA, 2012. USENIX Association.
5. Kadekodi (S.), Silas (S.), Clausen (D.) et Merchant (A.). – Practical design considerations for wide locally recoverable codes (lrcs). *ACM Trans. Storage*, vol. 19, n4, novembre 2023.
6. Luby (M. G.). – Lt codes. – In *The 43rd Annual IEEE Symposium on Foundations of Computer Science, 2002. Proceedings.*, pp. 271–280, 2002.
7. Rashmi (K. V.), Shah (N. B.), Gu (D.), Kuang (H.), Borthakur (D.) et Ramchandran (K.). – A "hitchhiker's" guide to fast and efficient data reconstruction in erasure-coded data centers. – In *Proceedings of the 2014 ACM Conference on SIGCOMM*, *SIGCOMM '14*, SIGCOMM '14, p. 331–342, New York, NY, USA, 2014. Association for Computing Machinery.
8. Rashmi (K. V.), Shah (N. B.) et Ramchandran (K.). – A piggybacking design framework for read-and download-efficient distributed storage codes. *IEEE Transactions on Information Theory*, vol. 63, n9, 2017, pp. 5802–5820.
9. Reed (I. S.) et Solomon (G.). – Polynomial codes over certain finite fields. *Journal of the Society for Industrial and Applied Mathematics*, vol. 8, n2, 1960, pp. 300–304.
10. Sathiamoorthy (M.), Asteris (M.), Papailiopoulos (D.), Dimakis (A. G.), Vadali (R.), Chen (S.) et Borthakur (D.). – Xoring elephants : novel erasure codes for big data. *Proc. VLDB Endow.*, vol. 6, n5, mars 2013, p. 325–336.
11. Shokrollahi (A.) et Luby (M.). – Raptor codes. *Found. Trends Commun. Inf. Theory*, vol. 6, n 3–4, mars 2009, p. 213–322.
12. Singleton (R.). – Maximum distanceq-nary codes. *IEEE Transactions on Information Theory*, vol. 10, n2, 1964, pp. 116–118.

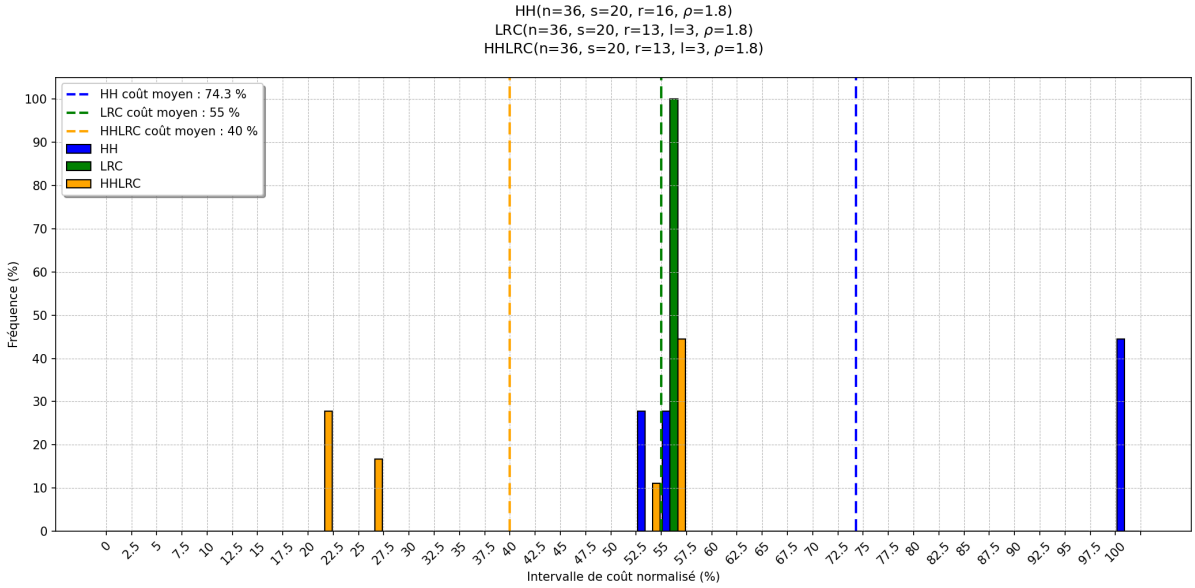


FIGURE 5 – Coûts de reconstruction pour $\rho = 1.80$ et $e = 1$.

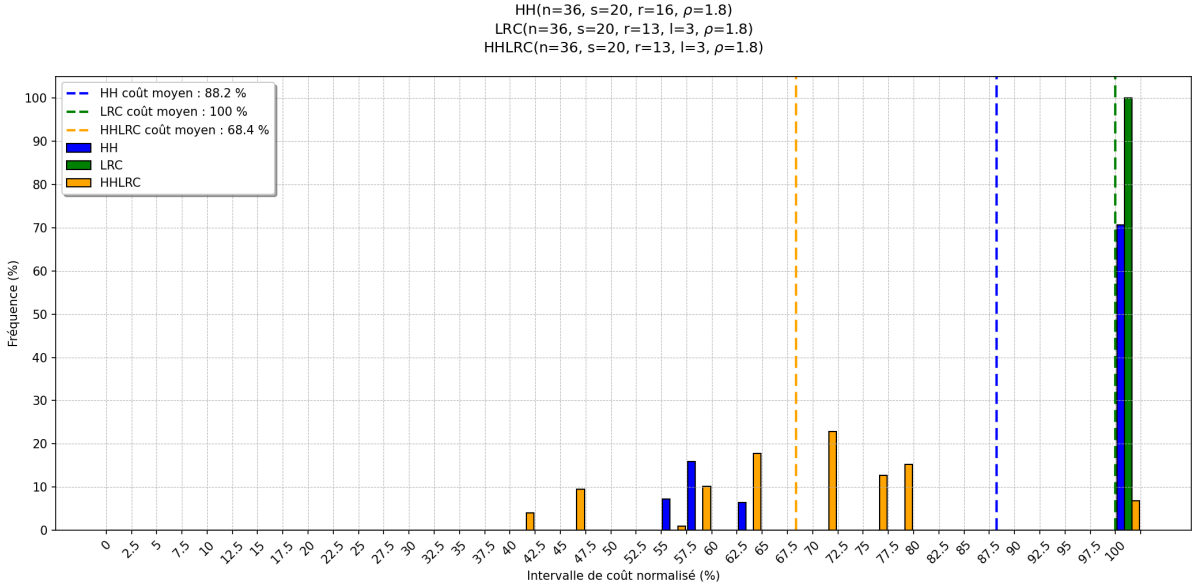


FIGURE 6 – Coûts de reconstruction pour $\rho = 1.80$ et $e = 2$.

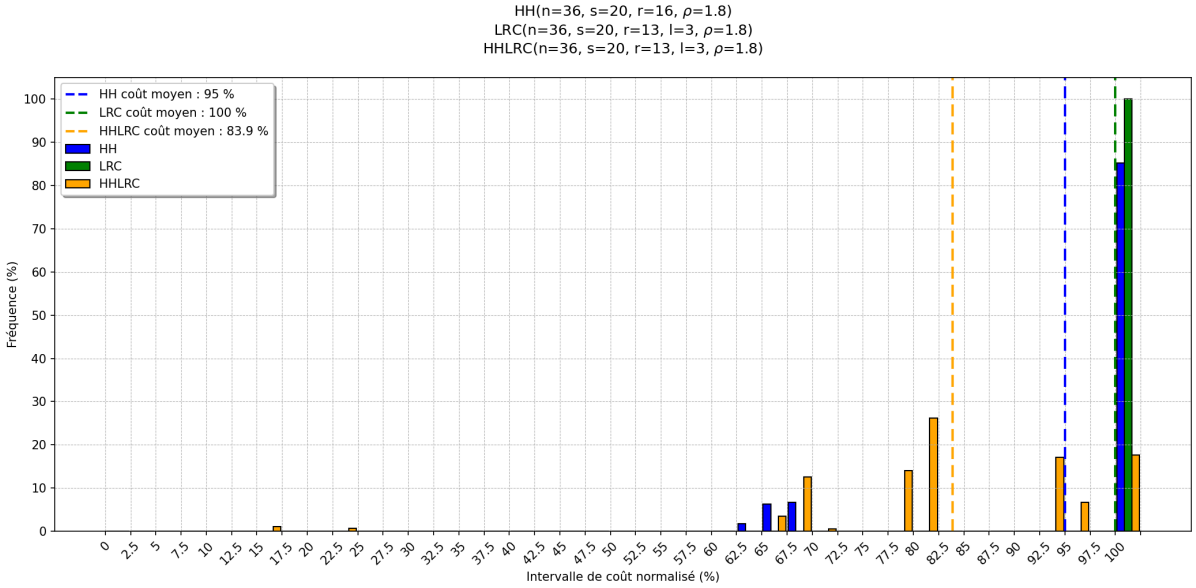


FIGURE 7 – Coûts de reconstruction pour $\rho = 1.80$ et $e = 3$.

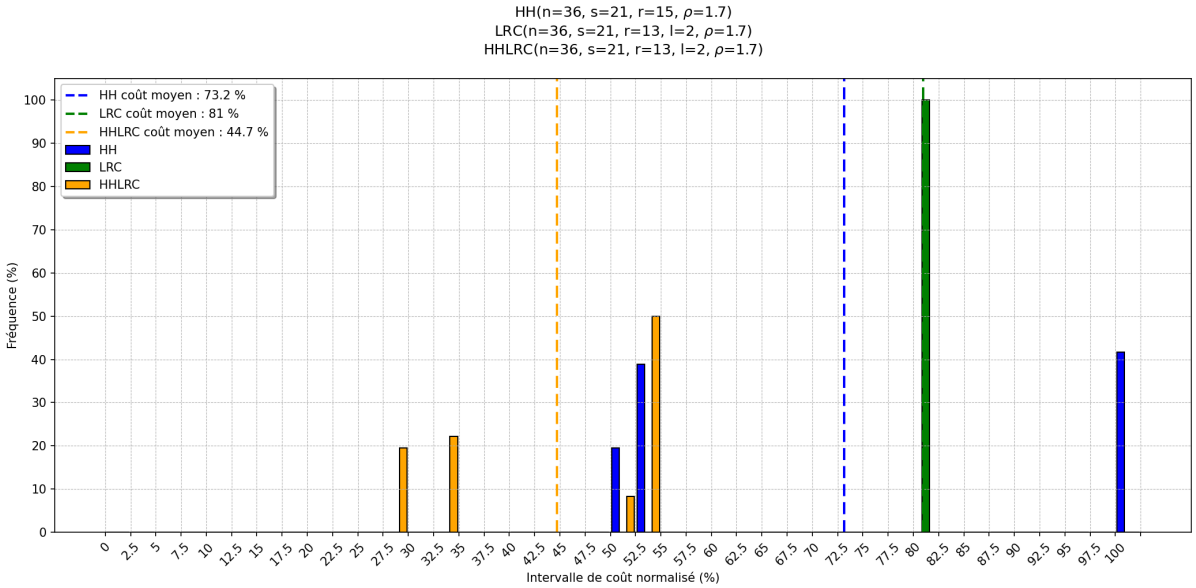


FIGURE 8 – Coûts de reconstruction pour $\rho = 1.71$ et $e = 1$.

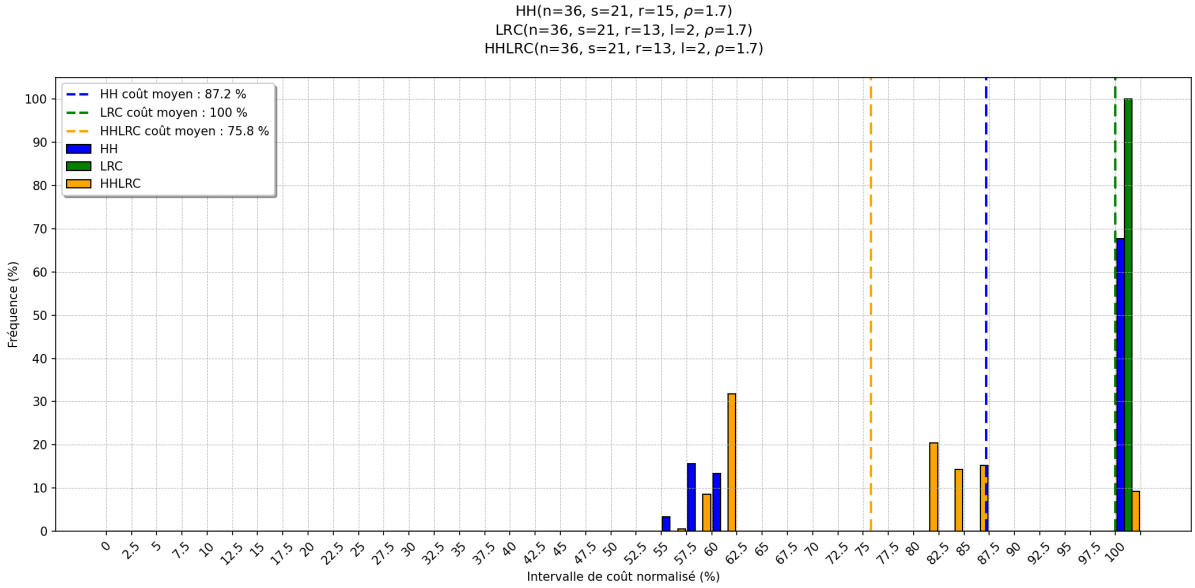


FIGURE 9 – Coûts de reconstruction pour $\rho = 1.71$ et $e = 2$.

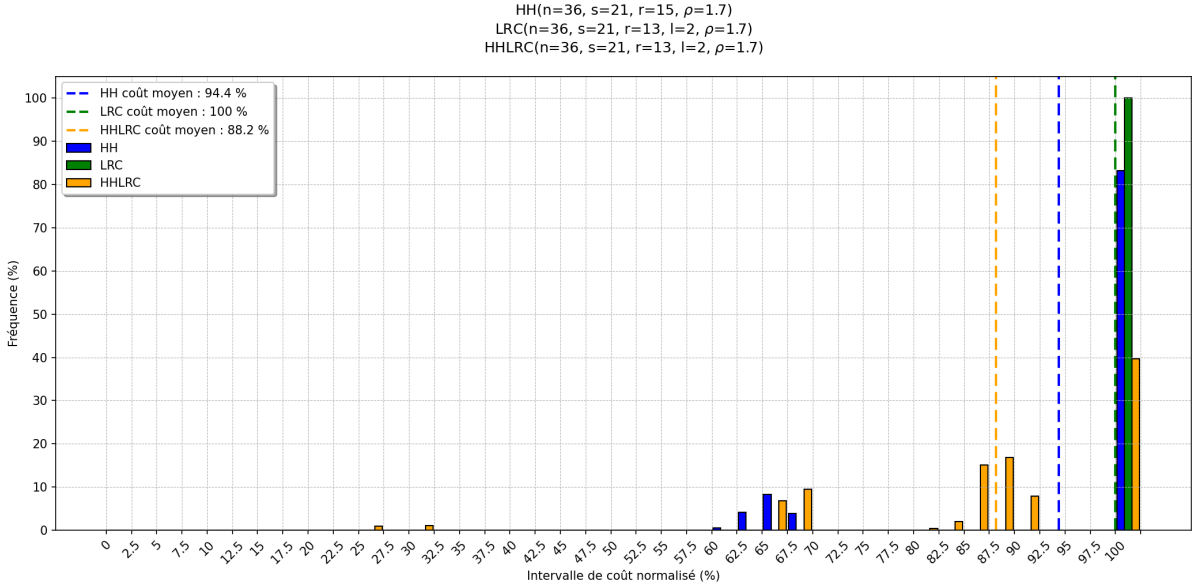


FIGURE 10 – Coûts de reconstruction pour $\rho = 1.71$ et $e = 3$.

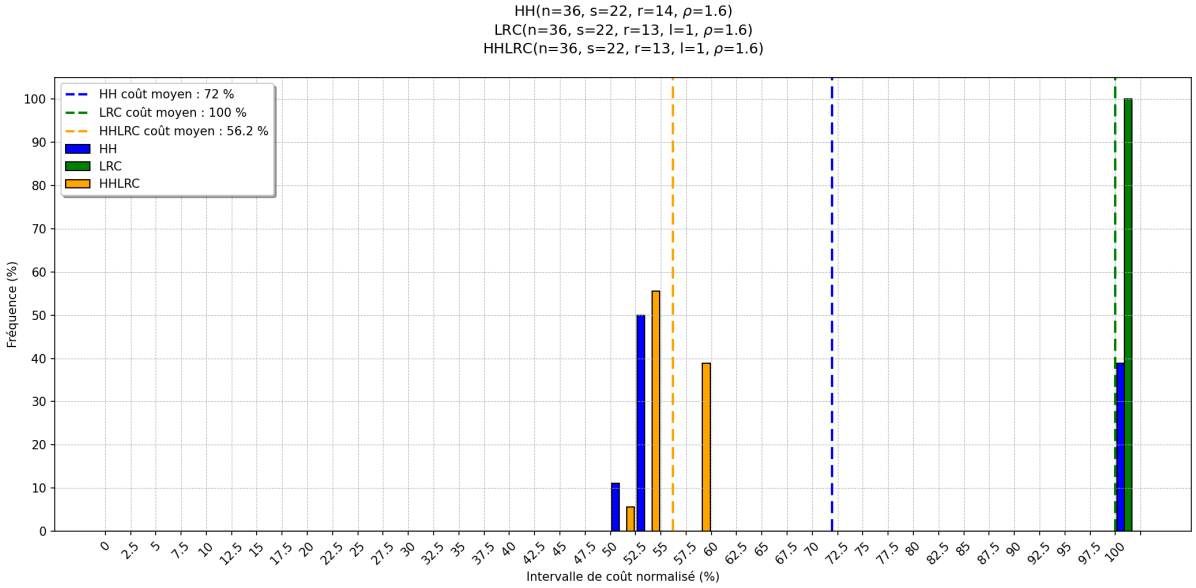


FIGURE 11 – Coûts de reconstruction pour $\rho = 1.64$ et $e = 1$.

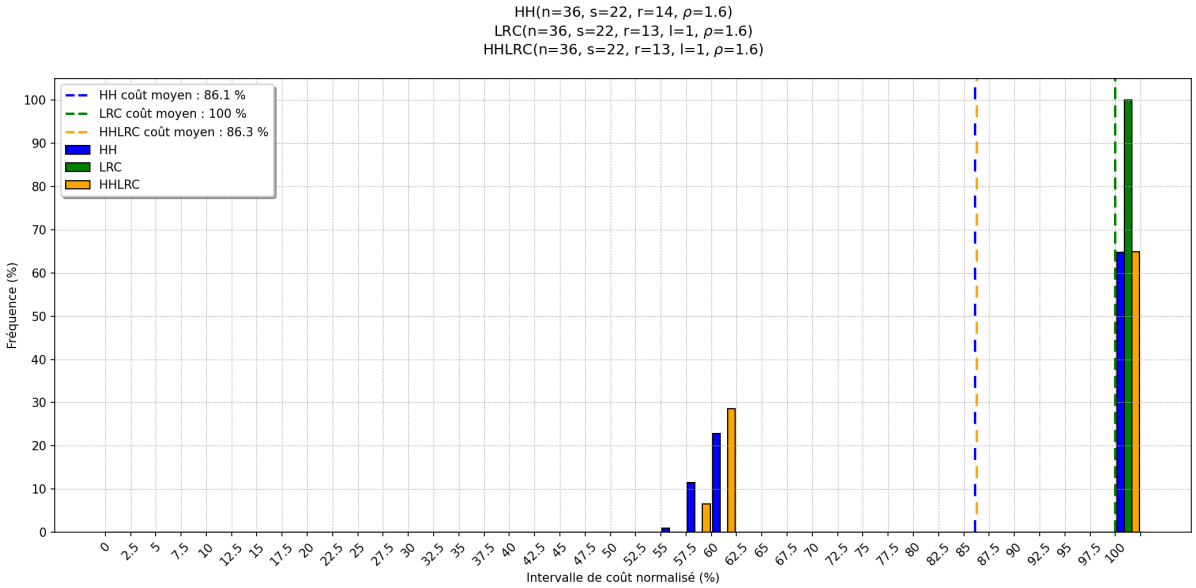


FIGURE 12 – Coûts de reconstruction pour $\rho = 1.64$ et $e = 2$.

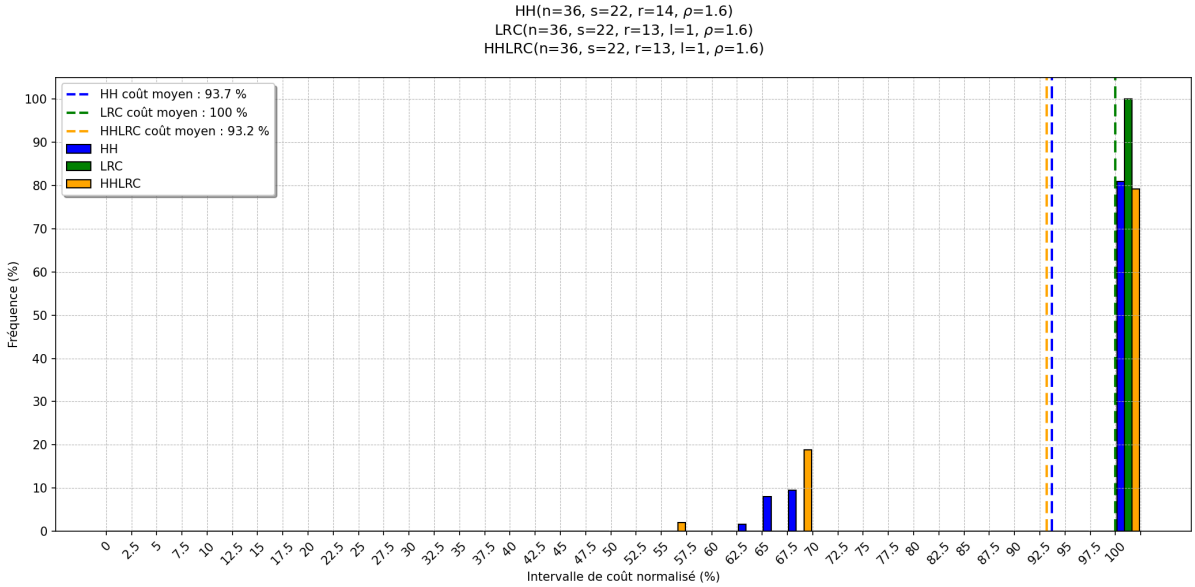


FIGURE 13 – Coûts de reconstruction pour $\rho = 1.64$ et $e = 3$.