Efficient Updates in LDPC Coded Distributed Storage Systems

Junming Ke
University of Tartu, Estonia.
junming.ke@ut.ee

***Abstract*—The update performance is becoming a common concern in the modern distributed storage systems. In this work, we analyze the application of graph-based codes, especially Low-Density Parity-Check (LDPC) codes for distributed storage systems in terms of frequent local updates. We first present the LDPC repetition codes and find that LDPC repetition codes achieve good update performance but work poorly in terms of repair performance, then we propose LDPC ideal codes which is update efficient by using discrete uniform distributions and the ideal distributions. We also evaluate the update performance of LDPC ideal codes by simulations. Finally, we propose the initial step of the projective plane theory to find the update efficient LDPC codes for the future work.**

I. Introduction

With the increasing size of datasets nowadays, modern distributed storage systems prefer to keep data with redundancy to prevent data loss. A widely used approach is *erasure coding* [3], where the distributed system takes the data as input, generates additional redundant symbols. The distributed system stores both data and redundant symbols in its distributed storage, so that any lost data can be restored from the remaining data and redundant symbols. The input data will be called *data symbols*, the redundant data will be called *parity symbols*. The distributed systems adopt the *erasure coding* approach where a failed or busy disk corresponds to a deleted symbol. The distributed systems organize their distributed storage in many different racks, and each rack contains several storage nodes that can store both data symbols and parity symbols. Different racks may lie in different geographical regions. The data transmission costs across these racks are higher than for the data transmission inside the rack (typically, 5-20 times higher.).

Despite erasure coding being efficient, it still incurs expensive maintenance costs in the update procedure. Any update of a single data symbol will cause the update of several parity symbols. If data symbols and parity symbols are stored in different racks, this will lead to a number of data transmissions between racks.

An interesting topic in *erasure coding* is the construction of the *codes*, *codes* are the methods to determine the relations of the *data symbols* and the *parity symbols* by its construction. In this research, motivated by the update issue, which has been raised in modern distributed systems, we plan to investigate the Low-Density Parity-Check (LDPC) code performance in distributed storage systems. The attractive property of the LDPC code is its sparse generator matrix. The sparse generator matrix means that a single update would cause only a small number of parity updates. Following this intuition, we will study the application of LDPC codes in distributed storage systems and compare them to different coded distributed storage systems.

In general, we make the following contributions in the paper.

1) We first give a introduction of LDPC code and its application in the distributed storage systems, then we briefly introduce the repair bandwidth and the update bandwidth.
2) We find that the repetition codes perform perfect in terms of update scenario and give a construction of LDPC repetition codes by only using discrete uniform distribution, however, due to the limited performance of repetition codes in terms of repair bandwidth, We propose LDPC ideal codes which is update efficient by our analysis.
3) We simulate the update performance of LDPC ideal codes and verified that this code is update efficient.
4) We propose the initial step of the intersections of the LDPC codes and projective plane theory for the future work.

II. Related Work

Maximum distance separable (MDS) [13] codes are widely used in distributed storage systems. A well-known class of MDS codes are Reed-Solomon (RS) codes [12]. An MDS code encodes k data symbols over a finite field $\mathbb{F}_q$ into n coded symbols. If the code is *systematic*, that includes k data symbols and $n-k$ parity symbols. For MDS codes all the symbols can be reconstructed from any k of the n coded symbols, we call this coding scheme an (n,k) MDS code. Some well-known distributed storage systems are using RS codes as their underlying structure, such as Google File System (Colossus) [5], Facebook Hadoop Distributed File System (HDFS) [15], Yahoo Cloud Object Store [9] and many others.

The update of a data symbol triggers parity updates of all other related parity symbols, recently a number of researches have been conducted towards this issue [14]. Most of them are focusing on how to batch the updates so that multiple data updates can be integrated as one update, in which distributed storage systems are based on RS codes [2] [7]. LDPC code is proposed by Robert Gallager [6], it is defined by a sparse parity-check matrix, which means, the related parity symbols

of one data symbol will be relatively small compared to the RS code.

III. LDPC Codes as the Local Repairable Codes

A. LDPC Codes

In order to define LDPC codes, we first introduce graph-based codes [8]. Given that a bipartite graph $G = (V, W, E)$, G has two bipartition classes V, W, where $|V| = n, |W| = m$ $(m \leq n)$, and E is the edge set. Each edge connects a vertex of W and a vertex of V.

Definition III.1 (Graph-based codes)**.** An (n, M, d) linear code C over a field $\mathbb{F}_q^n$ is called a *graph-based code* if C is defined as follows:

$$C = \{c \in \mathbb{F}_q^n : \forall j \in W, \sum_{i \in \Lambda j} \gamma_{i,j} c_i = 0\}.$$

Where $\gamma_{i,j} \in \mathbb{F}_q$ are non-zero coefficients and Λj is the neighborhood of vertex j.

In graph-based codes, each vertex in V is a data symbol and each vertex in W is a parity symbol. It is easy to check that the rate of the graph-based codes is at least $(1 - \frac{m}{n})$, in other words, for n useful data symbols, the graph-based codes generate at most m redundant parity symbols. A graph-based code is called an LDPC code if the degree of each vertex of W in G is small on average in a cluster of graph-based codes.

Remark (LDPC construction). Choose a rate $R \in (0, 1)$ and a sparsity parameter s. Let $G_i = (V, W_i, E_i)$ be a bipartite graph for $i = 1, 2, ..., (1 - R)s$, each bipartite graph G_i has a common vertex set V and disjoint vertex sets W_i, among them, the size of V is n and the size of W_i are all identical to $\frac{n}{s}$. Let $G = (V, W, E)$ be the union bipartite graph, where W (E) is the union of all of the W_i (E_i) for $i = 1, 2, ..., (1-R)s$. Furthermore, coefficients $\gamma_{i,j}$ are random choose in $\mathbb{F}_q^n$ such that $\sum_{i \in \Lambda i} \gamma_{i,j} c_i = 0$. Thereby, we construct an LDPC code by graph G.

If each G_i is an independent uniformly random bipartite graph where the degree of the nodes in V is 1 or 0, and the degree of the nodes in W_i is s, the constructed LDPC code is regular LDPC code. Otherwise, the constructed LDPC code is irregular LDPC code [11].

B. Storage Application

As we obtained an LDPC code based on a bipartite graph from the above, now we continue to illustrate the storage application of the LDPC codes.

We assume that we already have a bipartite graph such as Fig. 1, in this graph, we have 3 parity checks and 7 data nodes. Based on this graph, it is possible to assign the variable node (i.e., parity symbol.) for each parity check. For instance, the parity check w_1 represents $v_1 + v_4 + v_6 + v_7 = 0$, we randomly choose one of the data nodes v_1 as our parity symbol, therefore $v_1 = v_4 + v_6 + v_7$, we call v_4, v_6, v_7 as data symbols. We do this random choosing process for each parity check, finally, we obtain a group of parity symbols and a group of data symbols.

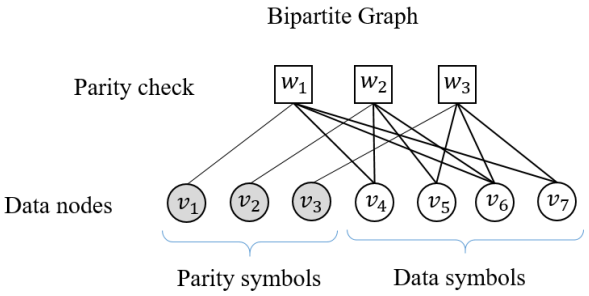

Figure 1. Bipartite graph of an example. The circle represents the data node and the square stands for parity check.

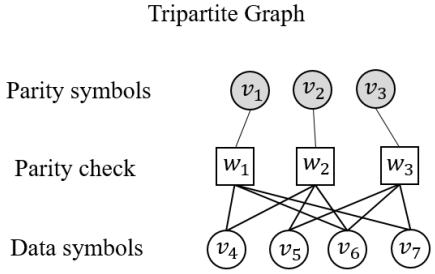

Figure 2. Tripartite graph of the example. The grey circle represents the parity symbol and the hollow circle stands for data symbol.

After that, we can generate a tripartite graph to represent the storage application as shown in Fig. 2, wherein, the group of parity checks connects the group of data symbols and the group of parity symbols, respectively. From this LDPC code, we can divide a file as 4 data symbols and store them into v_4, v_5, v_6 and v_7, the parity symbols can be generated accordingly.

In addition, a parity-check matrix H can also be obtained from the bipartite graph as follows.

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

where each row imposes on a parity check. This matrix indeed represents $[n, k, d]$ code where $n = 7$ represents the codeword length, $k = 4$ represent the data length, $d = 3$ represents the minimum distance of two codewords. A generator matrix G can be calculated by formula $HG^T = 0$, we have the following G for H.

$$G = \begin{vmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 \end{vmatrix}. \tag{2}$$

An storage application of this example can be represented as $(d_1, d_2, d_3, d_4)G$, thus we obtain $(v_1, v_2, v_3, v_4, v_5, v_6, v_7)$ to store. It should be noted that the generator matrix can be not unique as we can do row operation based on G, therefore, (d_1, d_2, d_3, d_4) can be a different vector compared

with (v_4, v_5, v_6, v_7). For example, generator matrix G' as presented can also satisfy the parity check matrix H, namely, $HG'^T = 0$.

$$G' = \begin{vmatrix} 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{vmatrix}. \quad (3)$$

We call that a code is systematic if $(d_1, d_2, d_3, d_4) = (v_4, v_5, v_6, v_7)$, that is to say, the last $n-k$ columns constitute the identity matrix.

We call a code $C^\perp$ as the dual code of the original code C if the parity check matrix H of C is the generator matrix G of $C^\perp$, it is obvious that the parity check matrix H of $C^\perp$ is also the generator matrix G of C [13].

C. Repair Bandwidth

A repair setting is described as following. A node v_k in vertex set V is erased, in order to repair that data of that node, we first identify the node w_j in vertex set W that is connected to the node v_k, then we download the adjacent nodes connected to the node w_j. Because we have $\sum_{v_k \in \Lambda w_j} \gamma_{v_k, w_j} c_{v_k} = 0$ for each $w_j \in W$, the node v_k can be reconstructed by this equation.

Lemma III.1 (Average repair bandwidth [11])**.** *Given an LDPC code by a graph $G = (V, W, E)$, the average repair bandwidth of this LDPC code is $\frac{\sum_{j=1}^{m} d_j(d_j-1)}{|E|}$, where d_j is the degree of node j in W.*

Proof. The erasure of each node in V requires $d_j - 1$ bandwidth in graph G, thereby, all of the single erasure of the node in $|V|$ require $\sum_{j=1}^{m} d_j(d_j - 1)$ bandwidth. We assume the erasure probability of each node is uniformly random, in this case, the erasure of each node is similar to the erasure of the edge E in graph G. It should be noted that one node in V with multiple edges connected, the repair method of this node is identical to its degree, therefore, the total number of erasure possibilities is identical to the number of edges $|E|$. □

D. Update Bandwidth

A distributed storage system usually requires a part of the information to be updated, and most of the contents are needed to keep the same. As shown in Figure. 1 and Figure. 2, a file has been divided into k parts, and these parts generate n symbols. In a single update case, an overall update of the original file is impractical, which requires all of the n symbols to be updated. When a data symbol update occurs in a distributed storage system, we define the update bandwidth B_u as the number of symbols to be updated in the whole system. In general, we know that $1 \leq B_u \leq n$.

Lemma III.2 (Lower-bound of the update bandwidth)**.** *The lower-bound of the update bandwidth for a code is at least its minimum distance d.*

Proof. A code C maps a word x in alphabet F^n into a codeword c, the minimum distance d defines that the minimum distance of every two codewords. In the update setting, an update means a word x_1 is changed to a word x_2, the code maps the word x_1, x_2 in alphabet F^n into the codewords c_1, c_2. From the minimum distance definition, $d = \min_{c_1, c_2 \in C : c_1 \neq c_2} \mathbf{d}(c_1, c_2))$, it exactly identifies the smallest number of different bits of c_1 and c_2, which is indeed the minimum update bandwidth of the code C. □

Lemma III.3 (Upper-bound of the update bandwidth for systematic codes)**.** *The upper-bound of the update bandwidth for a systematic code is at most $n - k + 1$.*

Proof. For a systematic code, the k data symbols are independent, if one data symbol needs an update, it does not influence other data symbols, the maximum number of symbols to be updated is the number of parity symbols $n - k$ and itself. □

As we can observed from Lemma. III.2 and Lemma.III.3, a systematic Reed-Solomon codes has an exactly $n - k + 1$ update bandwidth since it has the minimum distance $n-k+1$. We also notice that a single update would cause at most the updates of all nodes, therefore, the upper-bound of the update bandwidth for any linear codes is at most n.

IV. LDPC REPETITION CODES

Since an update causes at least one update (itself) in uncoded distributed storage systems, in the coded distributed storage systems, it is natural that a repetition code has a perfect performance in terms of the update operation because repetition code only causes 2 updates. A repetition code corresponds to generating each encoding symbol by a randomly selected input symbol, that is, copy the input symbol. Since the update of the input symbol will only cause a few updates of generated encoding symbols, this indicates that the update bandwidth of repetition code is the constant number.

Constructing repetition code by LDPC codes is not an easy operation, we should add the constraint to the distribution of LDPC construction. Assume that we wish to copy the data symbol once, we first fix $R = 0.5$ and $s = n$. Given a discrete uniform distribution as follows.

Definition IV.1 (Discrete uniform distribution)**.** For any $i \in [1, n]$, a distribution is called discrete uniform distribution if the cumulative distribution function of this distribution is $F(i; 1, n) = \frac{\lfloor i \rfloor}{n}$.

We use the above discrete uniform distribution to select 2 vertices in the vertex set W_i in a bipartite graph $G_i, i \in [1, n]$. For each selection step inside the graph G_i, we select the vertex one-by-one in graph G_i, if the vertex is already being selected, then discard this selection and do the new selection until we find a new different vertex. In the process of union the graph G_i and G_{i+1}, since we discard the repeated vertex, the vertices in V all have the degree less or equal than 1 in the new union graph and the vertices in W all have the degree 2 in the new union graph.

we define selecting 2 vertices in the vertex set W_i as a distribution of the degree, the probability that the degree in each bipartite graph G_i is 2, is denoted by $\rho(2) = 1$.

According to the classical balls and bins analysis [4], this selection continues until we cover all of the vertices in V, which is bins. n valid selections are remaining since we discard the invalid selections.

Lemma IV.1. *In order to obtain a repetition code with high probability, we need to run the discrete uniform distribution $nln(n)$ times.*

Proof. Let X_i be the index indicating that whether i is selected or not, $X_i = 1$ if it is not selected, otherwise $X_i = 0$. After m discrete uniform distributions, $E(X_i) = Pr[i \ is \ not \ selected] = \sum_j^m (1 - Pr[i \ is \ selected]) = (1 - \frac{1}{n})^m$. All of the X_i yields $E(X) = nE(X_i) = n(1 - \frac{1}{n})^m \leq ne^{-\frac{m}{n}}$, since the probability of $P(exists \ an \ unselected \ i) \leq E(X) \leq ne^{-\frac{m}{n}}$. Let $P(exists \ an \ unselected \ i) << 1$, we get $m >> nln(n)$. □

However, a repetition code is not good for the storage system since it adds a lot of redundancy and its repair performance is relatively lower than other code schemes. Even though repetition code cannot be used practically, it still gives us an intuition that the lower bound of the update bandwidth, namely, the constant update bandwidth. The trade-off between update and repair motivates us to find the code which is acceptable in terms of both bandwidths. As we analyzed in Lemma. III.2 and Lemma.III.3, we know that the update bandwidth with $O(n)$ is not the optimal case, therefore we wish to find the sub-linear algorithm in terms of update bandwidth.

Definition IV.2 (Update efficient [1])**.** We say an (n, M, d) linear code C is update efficient if its update bandwidth is sub-linear to the codeword length n, namely, $o(n)$.

Under Definition. IV.2, it is clear that repetition codes are update efficient, Reed-Solomon Codes are not update efficient.

V. LDPC IDEAL CODES

Although the LDPC repetition codes works poorly in repair scenario, it still gives us a insight into the update efficient codes. For the parity check matrix of an update efficient code, on the one hand, the small hamming weight of each row can make the update efficiently, on the other hand, LDPC codes aim at making the degree of each vertex of W small. In this case, we wish to design a code with a distribution of small degree and obtain the update efficient codes. From LDPC repetition codes, we know that the degree with 2 provides a bad performance in terms of repair bandwidth but a good performance in terms of update bandwidth.

Definition V.1. The ideal distribution is $\rho(1), \rho(2), ..., \rho(a)$, where $a < n$ is the fixed maximum degree of the vertices of W, and $\rho(2) = \frac{2}{a}, \rho(i) = \frac{2}{i(i-1)}$ for $i = 3, 4, ..., a$.

In ideal distribution, $\rho(3) = \frac{2}{3*2} = \frac{1}{3}$, which is relatively high in the degree distribution. The ideal distribution guarantees that the obtained parity check matrix is sparse for relatively smaller a, also we restrain the number of degree 2 relatively small for given $a > 2$. As we know, it is impossible to select one parity check row with degree 1, because it contradicts with the definition of the graph-based codes. We call an LDPC code as the LDPC ideal code if its degree distribution is the ideal distribution.

Theorem V.1. *The LDPC ideal codes is update efficient.*

Proof. From the ideal distribution, we generate a parity check matrix as we desired. The degree of each vertex in W is i with the probability $\rho(i)$, and the update bandwidth of a vertex in V is indeed equals to the degree of its neighbour in W. Thus the average update bandwidth is $\sum_i^a i\rho(i) = \sum_i^a \frac{2}{i-1} \approx 2ln(a) < o(n)$. □

We construct LDPC repetition codes by discrete uniform distribution, we also construct LDPC ideal codes by discrete uniform distribution as follows.

First of all, We fix R and s, for example $R = 0.5$, $s = n$, then we choose an appropriate a as our maximum degree, the purpose of this process is to generate $\frac{n}{R}$ rows. The a should be larger if R approaches 1, in this case, the parity check matrix can cover as more vertices as it can. Secondly, from the ideal distribution, a discrete degree is obtained, we use discrete uniform distribution to obtain the random vertex in V one by one. For each $nlnn$ uniform selections, we discard the selections if it selects the same vertex, then we assign the selected vertices to the row by the degree which is obtained by ideal distribution.

Theorem V.2. *In order to obtain an LDPC ideal code, we need to run the discrete uniform distribution $\frac{2ln(a)nlnn}{R}$ times and the ideal distribution $\frac{n}{R}$ times.*

Proof. It is obvious that we need to run the ideal distribution $\frac{n}{R}$ times since we need to generate $\frac{n}{R}$ rows. The expected sum of the degree from ideal distribution is the multiplication of average degree and generated rows, i.e., $2ln(a)\frac{n}{R} = \frac{2nln(a)}{R}$. In average, every n degrees costs $nlnn$ selections, thus $\frac{2nln(a)}{R}$ degrees need $\frac{\frac{2nln(a)}{R} * nlnn}{n} = \frac{2ln(a)nlnn}{R}$ selections. □

It should be noted that, while we let $\rho(a) = 1$ for fixed a, and we can obtain a regular LDPC codes.

VI. EVALUATION

We generate the LDPC codes with $R = \frac{1}{2}$, which means, the number of parity symbols is equal to the number of data symbols. By using ideal distribution and discrete uniform distribution, we demonstrate the results of the generated code as shown in Figure. 3.

In more details, the top two figures show the degree of each row when the number of parity symbol is equal to 100 (unsorted and sorted, respectively.). The bottom two figures show the degree of each row when the number of parity

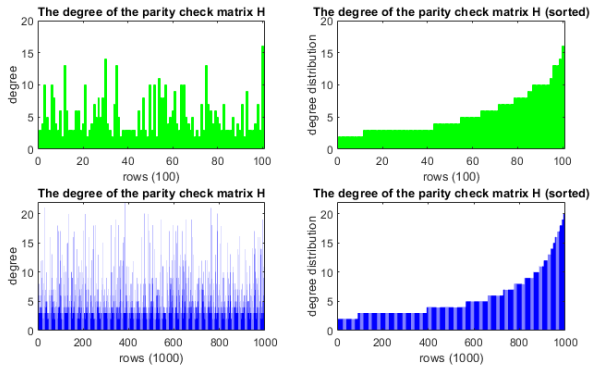

Figure 3. The degree distribution of the parity check matrix ($n = 100$ and $n = 1000$).

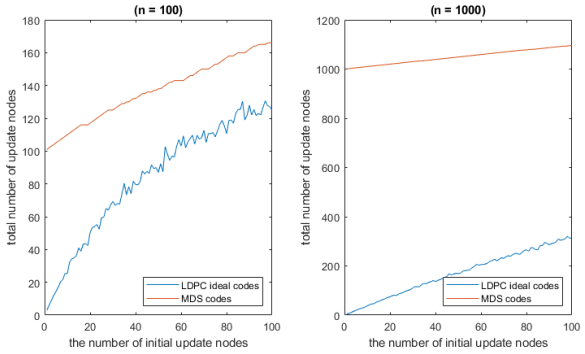

Figure 4. The number of initial update nodes and the total number of update nodes.

symbol is equal to 1000 (unsorted and sorted, respectively.). It is clear to see the degree distribution of the rows is ideal distribution by sorted figures. The number of rows with degree 3 is larger than others. Besides, the fewer number of rows with larger degree. We also verified the rank of the different parity check matrix H, the rank of H is n since we assigned each column with identical weight (that is, we use $nlnn$ discrete uniform distributions to ensure each coordinate would be selected.).

We simulated the update scenarios with LDPC ideal codes and MDS code (systematic generator matrix), the results of the total number of update nodes, and initially, the number of update nodes are as shown in Figure 5. The left figure illustrates the results when $n = 100$, the right figure illustrates the results when $n = 1000$. From the overall results, the total number of update nodes of LDPC ideal codes is slightly larger than the initial number of update nodes, it is consistent when $n = 100$ and $n = 1000$. Both figures show that the total number of update nodes of MDS codes is significantly larger than the total number of update nodes of LDPC ideal codes. Which explicates the better performance of LDPC ideal codes in terms of update scenario.

By demonstrating different simulation results in Figure 5, we plotted the results when the initially numbers of update nodes are equal to $1, 2, 3$. The total numbers of update nodes

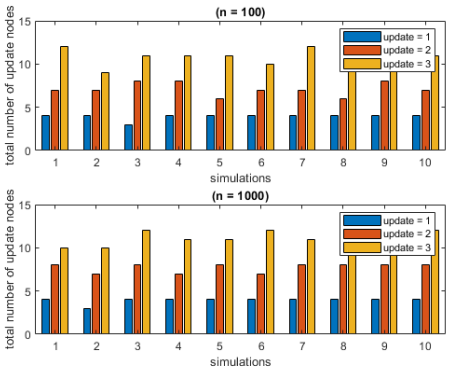

Figure 5. The total number of update nodes with different simulations.

are equal to almost $4, 7, 10$ in average when the the initially numbers of update nodes are equal to $1, 2, 3$, respectively. Which means, each update node might cause 3 or 4 additional updates by given parity check matrix H. Thus, the update bandwidth of our proposed LDPC ideal codes is relatively small.

VII. Future work

Though we given a precisely update efficient LDPC codes and verified our results by simulations, we are still care about the general cases of other update efficient LDPC codes. To uniform and observe those LDPC codes, a promising direction is investigating the construction of such codes by using projective plane theory.

A finite projective plane is a system of subsets of a finite set with given properties.

Definition VII.1 (Finite projective plane [10])**.** Let X be a finite set, and let L be a system of subsets of X, the pair (X, L) is called a finite projective plane if it satisfies the following conditions:

1. There exists a 4-element set $F \in X$ such that $|L' \cap F| \leq 2$ holds for each set $L' \in L$.

2. Any two distinct sets $L_1, L_2 \in L$ intersect in exactly one element, i.e., $|L_1 \cap L_2| = 1$.

3. For any two distinct elements $x_1, x_2 \in X$, there exists exactly one set $L' \in L$ such that $x_1 \in L'$ and $x_2 \in L'$.

Basically, the ideas are as follows. The (p-ary) dual code $C^{\perp}(2, q)$ (where q is a power of the prime p) is spanned by pairwise differences of characteristic vectors of lines (the dimension is actually smaller). It is also true thus that $C^{\perp}(2, q)$ is a subcode of $C(2, q)$ (dualization of codes is an involutory mapping, i.e. applied twice to get back the original. The code $C^{\perp}(2, q)$ is then called a self-orthogonal code). If q is a odd prime then the difference of dimensions of $C^{\perp}(2, q)$ and $C(2, q)$ is one.

Consequently, for odd prime q, we have H sparse, consisting of some differences of incidence vectors of lines of $PG(2, q)$ (i.e. rows of the incidence matrix: a 0-1 matrix, columns indexed by points of $PG(2, q)$, rows indexed by lines of

$PG(2, q)$, entry equal to 1 if and only if the point belongs to the line). G is anyway sparse, as all rows can be taken as incidence vectors of lines. In this case, we have an example of a code with rate very close to 1/2, with both G and H sparse.

VIII. Conclusion

This paper explores a precisely update efficient LDPC codes and verified the results by simulations. We provide a rigorous treatment of update problem in the distributed system. Our analysis highlights the existence of update efficient LDPC codes. In the given context, we also propose the initial step of the intersections of the LDPC codes and projective plane theory for the future work.

References

[1] N. P. Anthapadmanabhan, E. Soljanin, and S. Vishwanath. Update-efficient codes for erasure correction. In *2010 48th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pages 376–382. IEEE, 2010.

[2] J. C. Chan, Q. Ding, P. P. Lee, and H. H. Chan. Parity logging with reserved space: Towards efficient updates and recovery in erasure-coded clustered storage. In *12th USENIX Conference on File and Storage Technologies (FAST 14)*, pages 163–176, 2014.

[3] P. L. Dragotti and M. Gastpar. *Distributed source coding: theory, algorithms and applications*. Academic Press, 2009.

[4] D. P. Dubhashi and D. Ranjan. Balls and bins: A study in negative dependence. *BRICS Report Series*, 3(25), 1996.

[5] A. Fikes. Storage architecture and challenges. *Talk at the Google Faculty Summit*, 535, 2010.

[6] R. Gallager. Low-density parity-check codes. *IRE Transactions on information theory*, 8(1):21–28, 1962.

[7] Y. Hu, L. Cheng, Q. Yao, P. P. Lee, W. Wang, and W. Chen. Exploiting combined locality for wide-stripe erasure coding in distributed storage. In *19th USENIX Conference on File and Storage Technologies (FAST 21)*, pages 233–248, 2021.

[8] J. Mosheiff, N. Resch, N. Ron-Zewi, S. Silas, and M. Wootters. Ldpc codes achieve list decoding capacity. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 458–469. IEEE, 2020.

[9] P. Narayanan, S. Samal, and S. Nanniyur. Yahoo cloud object store-object storage at exabyte scale, 2017.

[10] J. Ne et al. *Invitation to discrete mathematics*. Oxford University Press, 2009.

[11] H. Park, D. Lee, and J. Moon. Ldpc code design for distributed storage: Balancing repair bandwidth, reliability, and storage overhead. *IEEE Transactions on Communications*, 66(2):507–520, 2017.

[12] I. S. Reed and G. Solomon. Polynomial codes over certain finite fields. *Journal of the society for industrial and applied mathematics*, 8(2):300–304, 1960.

[13] R. M. Roth. Introduction to coding theory. *IET Communications*, 47, 2006.

[14] Z. Shen and P. P. Lee. Cross-rack-aware updates in erasure-coded data centers: Design and evaluation. *IEEE Transactions on Parallel and Distributed Systems*, 31(10):2315–2328, 2020.

[15] M. Xia, M. Saxena, M. Blaum, and D. A. Pease. A tale of two erasure codes in hdfs. In *13th USENIX Conference on File and Storage Technologies (FAST 15)*, pages 213–226, 2015.