Bandwidth Cost of Code Conversions in the Split Regime

Francisco Maturana and K. V. Rashmi
Carnegie Mellon University, Pittsburgh, PA, USA
Email: fmaturan@cs.cmu.edu, rvinayak@cs.cmu.edu

2022 IEEE International Symposium on Information Theory (ISIT) | 978-1-6654-2159-1/22/$31.00 ©2022 IEEE | DOI: 10.1109/ISIT50566.2022.9834604

***Abstract*—Distributed storage systems must store large amounts of data over long periods of time. To avoid data loss due to device failures, an $[n, k]$ erasure code is used to encode k data symbols into a codeword of n symbols that are stored across different devices. However, device failure rates change throughout the life of the data, and tuning n and k according to these changes has been shown to save significant storage space. Code conversion is the process of converting multiple codewords of an initial $[n^I, k^I]$ code into codewords of a final $[n^F, k^F]$ code that decode to the same set of data symbols. In this paper, we study *conversion bandwidth*, defined as the total amount of data transferred between nodes during conversion. In particular, we consider the case where the initial and final codes are MDS and a single initial codeword is split into several final codewords ($k^I = \lambda^F k^F$ for integer $\lambda^F \geq 2$), called the *split regime*. We derive lower bounds on the conversion bandwidth in the split regime and propose constructions that significantly reduce conversion bandwidth and are optimal for certain parameters.**

An extended version of this paper is available at [1].

I. Introduction

Distributed storage systems use erasure codes to store large amounts of data reliably and without excessive storage overhead [2]–[5]. An $[n, k]$ erasure code encodes k symbols of data into a codeword with n symbols, which are then stored in different storage devices. If the code is maximum-distance-separable (MDS), then the full data can be decoded even after $n - k$ concurrent device failures.

Data in distributed storage systems is usually stored over long periods of time. Kadekodi et al. [6] showed that the failure rate of devices can significantly change over this time and that tuning the parameters n and k to adjust to these changes results in significant savings in storage space. In most cases, this tuning requires changing n and k simultaneously due to practical system constraints [6]. Other reasons to change n and k include adapting to changes in data popularity or space availability. Whenever n and k are changed, all the data that is already encoded must be modified to conform to the newly chosen parameters. The *default approach* to performing this change is to read all the data (decoding if necessary), re-encode with the new n and k, and write back to the storage devices. This results in very high consumption of cluster resources [6], such as network bandwidth, IO, and CPU, which can overwhelm the cluster for periods of several days.

The *code conversion* problem, introduced in [7], provides a theoretical framework to study the problem of efficiently changing the code parameters for already encoded data. Code conversion is the process of changing multiple (already encoded) codewords of an initial code of parameters $[n^I, k^I]$ to multiple codewords of a final code of parameters $[n^F, k^F]$ (Fig. 1). Let $r^I := n^I - k^I$ and $r^F := n^F - k^F$. The main goal of the study of code conversion [7]–[9] is to design the initial and final codes, as well as a conversion procedure, which can convert encoded data more efficiently than the default approach, for given parameters $(n^I, k^I; n^F, k^F)$. Codes designed for this purpose are referred to as *convertible codes*. The initial work on convertible codes [7], [8] addressed this challenge by focusing on the *access cost* of conversion, defined as the number of code symbols that are either read or written during conversion. In [7], [8], the authors showed that access cost can be significantly reduced compared to the default approach.

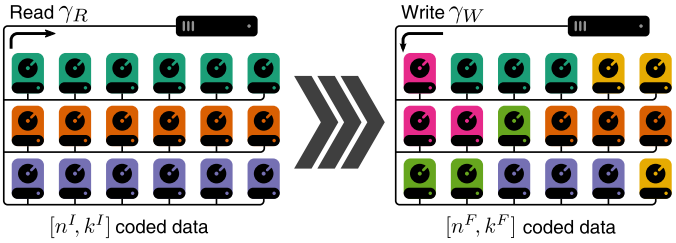

Fig. 1: Example of code conversion from code $[n^I, k^I]$ to code $[n^F, k^F]$. Each color denotes a different codeword. Conversion bandwidth is defined as the total amount of data read or written during conversion, i.e. $(\gamma_R + \gamma_W)$.

In [9], the authors introduced convertible codes optimized for another important metric: network bandwidth. Here, the cost of conversion is measured in terms of *conversion bandwidth*, defined as the total amount of data transferred between nodes during conversion, which is divided into read conversion bandwidth (γ_R) and write conversion bandwidth (γ_W). The work [9] focused exclusively on a parameter regime known as the *merge regime*, which consists of conversions that merge multiple codewords together (i.e. $k^F = \lambda^I k^I$ for integer $\lambda^I \geq 2$), and showed that conversion bandwidth can be significantly reduced compared to both the default approach and the codes that optimize the access cost of conversions.

In this paper, we study optimizing the conversion bandwidth for another important regime called the *split regime*, wherein a single initial codeword is split into final codewords, i.e. $k^I = \lambda^F k^F$ for some integer $\lambda^F \geq 2$. In particular, we derive lower bounds on the conversion bandwidth of codes in the split regime, and we propose constructions that match those bounds.

This work was funded in part by an NSF CAREER award (CAREER-1943409), an NSF CNS award (CNS-1956271), a Google faculty research award, and a Facebook distributed systems research award.

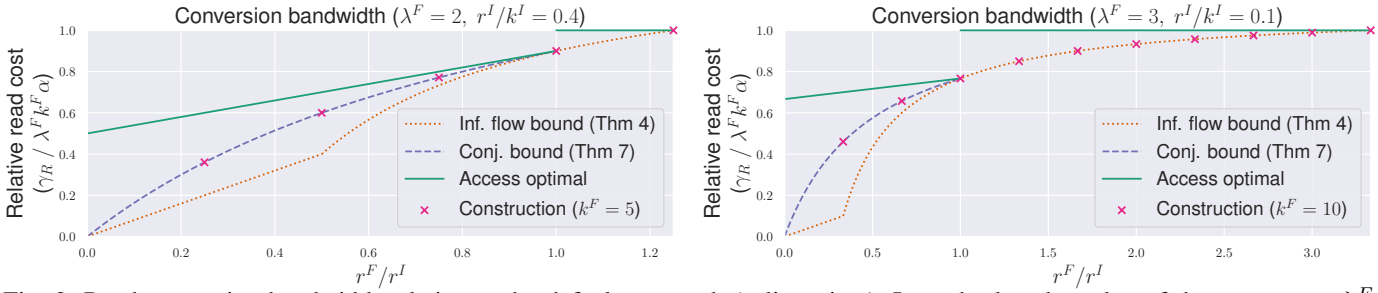

Fig. 2: Read-conversion-bandwidth relative to the default approach (split regime). In each plot, the value of the parameters λ^F and the ratio r^I/k^I are fixed, and the value of the ratio r^F/r^I ranges in $(0, (\lambda^F r^I/k^I)^{-1}]$. By choosing this parametrization, the plotted curves are independent of the value of k^F. For illustration, markers are added on the points that can be achieved by the construction of §IV when k^F takes the given example value.

The split regime is important because it plays a key role in the conversions with arbitrary parameters $(n^I, k^I; n^F, k^F)$ [8].

We first focus on lower bounding conversion bandwidth in the split regime. To do this, we model conversion using an information flow graph with edges of *variable capacities*. Using this model, we derive a lower bound on conversion bandwidth for convertible codes in the split regime satisfying some technical conditions (Theorem 4). This bound shows that savings are not possible when $n^F \geq 2k^F$ but leaves room for significant savings otherwise. However, we show that this bound is not tight. For this reason, we introduce a conjecture (Conjecture 5) about the relationship between the amount of data that needs to be downloaded from different types of code symbols. Assuming this conjecture is true, we derive an additional lower bound (Theorem 7). Finally, we present constructions which achieve the combination of both lower bounds (Theorem 8). When $r^I \leq r^F$, these constructions achieve the lower bound of Theorem 4, i.e. the optimal conversion bandwidth. Otherwise, these constructions achieve the lower bound of Theorem 7, i.e. the optimal conversion bandwidth if Conjecture 5 is true.

The proposed constructions can perform conversion with significantly less conversion bandwidth compared to the default approach. Moreover, these constructions also require less conversion bandwidth than existing convertible codes optimized for access cost [8]. Table I and Fig. 2 compare the read conversion bandwidth of the different approaches (we omit write conversion bandwidth because it is the same for all three approaches).

We start by summarizing the relevant background and related work in §II. Then, we model conversion in the split regime and derive lower bounds for conversion bandwidth in §III. Finally, we present a construction matching those lower bounds in §IV.

TABLE I: Comparison of the read conversion bandwidth (read BW) of different approaches for split conversion.

Approach	Read BW ($r^I < r^F$)	Read BW ($r^I \geq r^F$)
Default	$\lambda^F k^F \alpha$	$\lambda^F k^F \alpha$
Access opt. [8]	$\lambda^F k^F \alpha$	$[(\lambda^F - 1)k^F + r^F]\alpha$
This paper	$\lambda^F k^F \alpha - r^I \left(\frac{k^F}{r^F} - 1\right)$	$\lambda^F r^F \alpha \frac{(\lambda^F-1)k^F + r^I}{(\lambda^F-1)r^F + r^I}$

II. Background and Related Work

A. Vector codes

Let $\mathbb{F}_q$ be the finite field of size q and let $[n, k, \alpha]$ denote a vector code $\mathcal{C}$ over $\mathbb{F}_q$, where $\mathcal{C} \subseteq \mathbb{F}_q^{\alpha n}$ is defined as an $\mathbb{F}_q$-linear subspace with dimension αk. We assume $\mathcal{C}$ has a given basis, which forms the *generator matrix* $\mathbf{G} \in \mathbb{F}_q^{\alpha k \times \alpha n}$ of a code. We denote the *encoding function* of $\mathcal{C}$ as $\mathcal{C}(\mathbf{m}) := \mathbf{mG}$ for message $\mathbf{m} \in \mathbb{F}_q^{\alpha k}$. If $\mathbf{G} = [\mathbf{I} \mid \mathbf{P}]$, (where $\mathbf{I}$ is the identity matrix) the code is said to be *systematic*. Let $[n] := \{1, \ldots, n\}$. The coordinates of codeword $\mathbf{c} \in \mathcal{C}$ are called *subsymbols*, and $\mathbf{c}_i = (c_{\alpha(i-1)+1}, c_{\alpha(i-1)+2}, \ldots, c_{\alpha i})$ is defined as the i-th *symbol* of $\mathbf{c}$ ($i \in [n]$). The code $\mathcal{C}$ is said to be maximum-distance-separable (MDS) iff for all $\mathbf{m} \in \mathbb{F}_q^{\alpha k}$, $\mathbf{m}$ can be decoded from any k symbols of $\mathbf{c} = \mathcal{C}(\mathbf{m})$. Codes with $\alpha = 1$ are called *scalar*. The notation $\mathbf{p}[i]$ is used to denote the i-th coordinate of vector $\mathbf{p}$.

B. Piggybacking framework for constructing vector codes

The *Piggybacking framework* [10], [11] is a framework for constructing an $[n, k, \alpha]$ vector code by using α instances of an $[n, k]$ scalar code as a *base code*, and then adding *piggybacks* to certain subsymbols of the code. The piggybacks are arbitrary functions of data from one instance added to another instance, chosen so as to grant some additional property to the code. Typically, the piggyback added to instance i is only a function of the data encoded by instances $1, \ldots, i-1$. This property ensures that the code can be decoded by sequentially decoding instances $1, \ldots, \alpha$ in order, using the data from the decoded instances to subtract the piggybacks. We will employ piggybacking to design bandwidth efficient convertible codes. Note that if the base code is MDS, then the constructed vector code with piggybacks is also MDS.

C. Convertible codes

Convertible codes [7] are erasure codes designed to enable encoded data to undergo efficient conversion. The objective of conversion is to convert codewords of an initial $[n^I, k^I, \alpha]$ code $\mathcal{C}^I$ into codewords of a final $[n^F, k^F, \alpha]$ code $\mathcal{C}^F$ such that the initial and final codewords decode to exactly the same set of data symbols. Assume, for now, $(n^I, k^I; n^F, k^F)$ are given, and α is arbitrary. In this paper, we focus on the case where both $\mathcal{C}^I$ and $\mathcal{C}^F$ are MDS, and in the so-called *split regime*, where $k^I = \lambda^F k^F$ for some integer $\lambda^F \geq 2$. This corresponds to conversions where a single initial codeword of $\mathcal{C}^I$ is split into λ^F final codewords of $\mathcal{C}^F$. Let $r^I := (n^I - k^I)$

and $r^F := (n^F - k^F)$. Let $\mathbf{m} := (m_i \in \mathbb{F}_q)_{i=1}^{\alpha k^I}$ be the data to be encoded, and let $\mathbf{m}_i := (m_{(i-1)k^F\alpha+j})_{j=1}^{\alpha k^F}$ be the data associated with final codeword $i \in [k^F]$. A split conversion from initial code $\mathcal{C}^I$ to final $\mathcal{C}^F$ is a procedure that takes $\mathcal{C}^I(\mathbf{m})$ as input and outputs $\{\mathcal{C}^F(\mathbf{m}_i) \mid i \in [\lambda^F]\}$. Our objective is to design the codes $(\mathcal{C}^I, \mathcal{C}^F)$ and an efficient conversion procedure for the given parameters $(n^I, k^I = \lambda^F k^F; n^F, k^F)$.

During conversion, there are three types of symbols: 1) *unchanged symbols*, which are the initial symbols that are retained in one of the final codewords (this does not require conversion bandwidth because the symbol does not move); 2) *retired symbols*, which are the remaining initial symbols that are not unchanged; and 3) *new symbols*, which are the remaining final symbols that are not unchanged. During conversion information is downloaded from unchanged and retired symbols, and then used to construct the new symbols.

Convertible codes that have the maximum number of unchanged symbols are called *stable*. Intuitively, more unchanged symbols imply fewer new symbols, which requires reading and writing less data when creating the new symbols. Therefore, to simplify our analysis we focus only on stable convertible codes: with k^F unchanged symbols per final codeword [8].

D. Other related work

Several works have studied problems that can be regarded as special cases of code conversion: [12], [13] studied the bandwidth required by the addition of extra parities to an MDS code ($k^I = k^F$ and $n^I < n^F$); [14] describes two pairs of non-MDS codes that can be converted back and forth; [15] studies a problem in distributed matrix multiplication where parameters are changed via local re-encoding. Another related problem is the *scaling problem* [16]–[28], which consists of converting each codeword of an $[n, k, \alpha]$ code, into a codeword of an $[n + s, k + s, k\alpha/(k + s)]$ code for given integer s. In other words, the amount of data in each codeword is kept constant, but the data is distributed across a different number of devices.

III. Conversion bandwidth of the split regime

In this section we analyze the conversion bandwidth required by MDS convertible codes in the split regime, i.e., the case where $k^I = \lambda^F k^F$ for some integer $\lambda^F \geq 2$.

In order to obtain a lower bound on the conversion bandwidth, we model split conversion as an information flow problem. In this model, we represent the flow of information during conversion as a DAG with edges with variable capacity that represent the transfer of data between nodes. Our objective is to set the capacity of edges in a way that minimizes the conversion bandwidth, while ensuring that the flow conditions necessary for conversion are met.

One challenge is that, as we will show, the bound we obtain through information flow is not achievable in general.[1] This bound is not achievable in general because retired symbols contain data that is associated with more than one final codeword.

[1]Split conversion corresponds to a *multi-source multicast* problem. In this case (unlike the *single-source* case) the information flow bound is not necessarily tight with respect to coding [29].

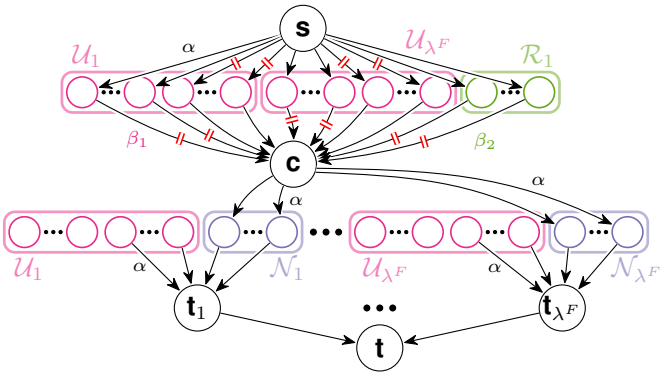

Fig. 3: Information flow graph of split conversion. For clarity, each unchanged symbol is drawn twice, in order to show the initial configuration of the system in the top row of nodes, and the final configuration in the bottom row of nodes. The edges with a red mark depict a graph cut.

Thus, in order to make use of these symbols during conversion, we must also download enough data from unchanged symbols to remove the "interference" from other final codewords. To this end, we introduce a conjecture and derive from it a lower bound which, as we show in §IV, is achievable.

A. Information flow

We model the conversion process using the graph (see Fig. 3) composed by the following nodes:

- source $\mathbf{s}$, representing the whole data $\mathbf{m} \in \mathbb{F}_q^{\alpha k^I}$;
- the set $\mathcal{U}_i$ for $i \in [\lambda^F]$, representing the unchanged symbols of final codeword i;
- the set $\mathcal{R}$ representing retired symbols;
- the set $\mathcal{N}_i$ for $i \in [\lambda^F]$, representing the new symbols of final codeword i;
- data collectors $\mathbf{t}_i$ for $i \in [\lambda^F]$ that represent the decoders for each final codeword;
- a central node $\mathbf{c}$ that computes the new symbols;
- a sink $\mathbf{t}$ collecting the data for all final codewords (i.e. $\mathbf{m}$).

Let (u, v, x) denote and edge from node u to node v with capacity $x \geq 0$. Nodes are connected by the following edges:

- $\{(\mathbf{s}, x, \alpha) \mid x \in \bigcup_i \mathcal{U}_i \cup \mathcal{R}\}$, representing the data stored in the initial symbols;
- $\{(x, \mathbf{c}, \beta_1) \mid x \in \bigcup_i \mathcal{U}_i\}$ representing the data downloaded from unchanged symbols;
- $\{(x, \mathbf{c}, \beta_2) \mid x \in \mathcal{R}\}$, representing the data downloaded from retired symbols;
- $\{(\mathbf{c}, x, \alpha) \mid x \in \bigcup_i \mathcal{N}_i\}$, representing the data written to the new symbols;
- $\{(x, \mathbf{t}_i, \alpha) \mid x \in V_i\}$ for $V_j \subseteq \bigcup_i (\mathcal{U}_i \cup \mathcal{N}_i)$ such that $|V_j| = k^F$ for $j \in [\lambda^F]$, representing decoding of final codeword i;
- $\{(\mathbf{t}_i, \mathbf{t}, \alpha k^F) \mid i \in [\lambda^F]\}$, representing the collection of all the decoded data.

In this paper, we focus on stable codes (see §II-C). Therefore, we have that $|\mathcal{U}_i| = k^F$, $|\mathcal{R}| = r^I$, and $|\mathcal{N}_i| = r^F$ $(i \in [\lambda^F])$. The total conversion bandwidth γ will be given by the total

size of the information communicated between nodes during conversion, which corresponds to the following equation:

$$\begin{aligned} &\gamma := \gamma_R + \gamma_W, \\ &\text{where } \gamma_R := \lambda^F k^F \beta_1 + r^I \beta_2 \text{ and } \gamma_W := \lambda^F r^F \alpha. \end{aligned} \tag{1}$$

We refer to γ_R as the *read conversion bandwidth* and to γ_W as the *write conversion bandwidth*. Our objective is to set (β_1, β_2) to minimize γ while ensuring an information flow of size αk^I (the size of the data $\mathbf{m}$) is feasible. Since γ_W is constant with respect to (β_1, β_2), our analysis will focus on γ_R.

Note that our model assumes a uniform amount of data downloaded from unchanged symbols and retired symbols. This is without loss of generality, since any stable convertible code with non-uniform downloads, can be made uniform by repeating the code a sufficient number of times and rotating the assignment of symbols to nodes with each repetition.

Our first lemma expresses the constraint which arises from considering the cut shown in Fig. 3.

Lemma 1: For all stable MDS $(n^I, k^I = \lambda^F k^F; n^F, k^F)$ convertible code:

$$\lambda^F \min\{r^F, k^F\}\alpha \leq \lambda^F \min\{r^F, k^F\}\beta_1 + r^I \beta_2. \tag{2}$$

Proof: For each $j \in [\lambda^F]$, consider a sink $\mathbf{t}_j$ that connects to all symbols in a final codeword but a set $S_j \subseteq \mathcal{U}_j$ of size $\min\{k^F, r^F\}$. Consider the cut defined by $\{\mathbf{s}\} \cup \bigcup_{j=1}^{\lambda^F} S_j \cup \mathcal{R}$. This cut yields (2) after simplification. ∎

Using (1), we can show that when $r^F \geq k^F$, no savings in conversion bandwidth are possible over the default approach.

Corollary 2: When $r^F \geq k^F$, we have $\gamma_R \geq \lambda^F k^F \alpha$. ∎

In other words, the default approach has optimal conversion bandwidth when $r^F \geq k^F$. For this reason, we will only focus on the case $r^F < k^F$.

To obtain a lower bound on γ, we will minimize it subject to (2) with β_1 and β_2 as variables.

Lemma 3: Assume $r^F < k^F$. Then, the value of γ is minimized subject to (2) when:

$$\beta_1 = \max\left\{1 - \frac{r^I}{\lambda^F r^F}, 0\right\}\alpha, \qquad \beta_2 = \min\left\{1, \frac{\lambda^F r^F}{r^I}\right\}\alpha.$$

Proof sketch: Note that β_2 offers the better "bang for the buck" for satisfying (2), because each unit of β_2 contributes r^I costing r^I, while each unit of β_1 contributes $\lambda^F r^F$ costing $\lambda^F k^F$. Thus, it is intuitively better to increase β_2 first as much as possible and necessary. Then, we set β_1 to satisfy (2). ∎

By replacing into (1), we obtain the following lower bound.

Theorem 4: For all stable MDS $(n^I, k^I = \lambda^F k^F; n^F, k^F)$ convertible code:

$$\gamma_R \geq \begin{cases} \lambda^F k^F \alpha - r^I \alpha \max\left\{\frac{k^F}{r^F} - 1, 0\right\} & \text{if } r^I \leq \lambda^F r^F, \\ \lambda^F \min\{r^F, k^F\}\alpha & \text{otherwise.} \end{cases}$$ ∎

This bound shows that there is potential for conversion bandwidth savings when $k^F > r^F$, because the bound is strictly lower than the default approach ($\lambda^F k^F \alpha$) in this region. Unfortunately, this bound is not always achievable, as we see next.

For example, suppose we have have a stable convertible code with $k^F > r^F$, $r^I = \lambda^F r^F$ and that we set $\beta_1 = 0$ and $\beta_2 = \alpha$. This assignment satisfies Theorem 4 (and it leads to a feasible flow in Fig. 3). However, as shown by previous work on access cost of conversion [8], it is not possible to perform conversion in this case by accessing fewer than $(\lambda^F - 1)k^F + r^F$ symbols. Furthermore, it can be shown that any assignment that makes $\beta_1 > 0$ necessarily leads to a higher conversion bandwidth than the lower bound of Theorem 4. The fundamental problem in this case is that to create new symbols for a particular final codeword we need to remove the interference from all other final codewords. This is not possible if the conversion procedure does not access a sufficient number of symbols.

For this reason, we introduce the following conjecture, which lower bounds the amount of data that needs to be downloaded from unchanged symbols based on the above intuition.

Conjecture 5: In the information flow model presented in this section, for all stable MDS $(n^I, k^I = \lambda^F k^F; n^F, k^F)$ convertible code we must have:

$$\lambda^F \beta_1 \geq (\lambda^F - 1)\beta_2. \tag{3}$$

We incorporate this constraint into the minimization of γ and obtain a different solution, which limits the amount of data downloaded from retired symbols when $r^I > r^F$.

Lemma 6: Assume $r^F < k^F$. Then, the minimum value of γ subject to (2) and (3) is achieved by Lemma 3 when $r^I < r^F$, and otherwise by:

$$\beta_1 = \frac{(\lambda^F - 1)r^F \alpha}{(\lambda^F - 1)r^F + r^I}, \qquad \beta_2 = \frac{\lambda^F r^F \alpha}{(\lambda^F - 1)r^F + r^I}.$$

Proof sketch: By (3), we set $\beta_2 = \min\left\{\alpha, \frac{\lambda^F}{\lambda^F - 1}\beta_1\right\}$. We then set β_1 in order to satisfy (2). When $r^I < r^F$, (3) is not tight, and we thus obtain the same values that Lemma 3. Otherwise, we obtain the stated values of β_1 and β_2. ∎

By replacing back into (1), we obtain the following lower bound based on Conjecture 5.

Theorem 7: If Conjecture 5 holds, then for all $(n^I, k^I = \lambda^F k^F; n^F, k^F)$ convertible code with $r^I \geq r^F$ and $r^F \leq k^F$:

$$\gamma_R \geq \lambda^F r^F \alpha \frac{(\lambda^F - 1)k^F + r^I}{(\lambda^F - 1)r^F + r^I}.$$ ∎

As we shall see in §IV, the proposed constructions achieve the combination of the lower bounds of Theorems 4 and 7. Thus, we finish this section by comparing the conversion bandwidth of our approach with that of the default approach and existing convertible codes optimized for access cost [8]. Since in all approaches the write conversion bandwidth is equal ($\lambda^F r^F \alpha$), we focus on the read conversion bandwidth. Table I includes the expressions for the read conversion bandwidth of different approaches. Figure 2 plots the lower bounds on read conversion bandwidth relative to the default approach for some example parameters. These results show that our approach can achieve significant savings in conversion bandwidth with respect to the default approach and access-optimal convertible codes.

IV. Explicit constructions

In this section, we present constructions for convertible codes in the split regime that optimize for conversion bandwidth. The constructions employ the Piggybacking framework [11].

Theorem 8: The constructions presented in this section achieve the optimal conversion bandwidth when $r^I \leq r^F$. Furthermore, they achieve the optimal conversion bandwidth when $r^I > r^F$ if Conjecture 5 is true. ∎

These construction require less conversion bandwidth than the default approach *and* the access optimal approach (regardless of Conjecture 5) as long as $r^F < k^F$ (Corollary 2). Due to space constraints, we only describe the construction for the case $r^I \geq r^F$. The construction for the case $r^I < r^F$ is similar.

1) Base code: We utilize an $[n^I, k^I]$ systematic code with a Vandermonde matrix with evaluation points $(\xi_1, \ldots, \xi_{r^I})$ as the parity matrix. A code of this form is guaranteed to be MDS when choosing ξ_t ($t \in [r^I]$) and field size as specified by the general construction in [7]. Nonetheless, in practice it is often possible to search for ξ_t that generate an MDS code over a given finite field. Let $\mathbf{h}_t := (h_1^{(t)}, \ldots, h_{k^I}^{(t)})^T = (1, \xi_t, \ldots, \xi_t^{k^I-1})^T$ be parity encoding vector $t \in [r^I]$ of the base code. In our construction, we use the property that $(h_1^{(t)}, \ldots, h_{k^F}^{(t)}) = \xi_t^{-(i-1)k^F}(h_{(i-1)k^F+1}^{(t)}, \ldots, h_{ik^F}^{(t)})$ for all $t \in [r^I]$ and $i \in [\lambda^F]$.

2) Piggybacking construction: We now describe the construction (assuming $r^F < k^F$). Recall that during conversion, we download β_1 from each unchanged symbol, and β_2 from each retired symbol, which are set as discussed in §III. If we set the set the size of each symbol as $\alpha := ((\lambda^F - 1)r^F + r^I)$, then $\beta_1 := (\lambda^F - 1)r^F$ and $\beta_2 := \lambda^F r^F$. For simplicity, we divide α into blocks: for a given $\ell \in [\alpha]$ we define (ℓ_1, ℓ_2) as follows.

$$(\ell_1, \ell_2) := \begin{cases} \left(\left\lceil \frac{\ell}{r^F} \right\rceil, (\ell - 1 \bmod r^F) + 1\right) & \text{if } \ell \leq \lambda^F r^F, \\ (\lambda^F + 1,\ \ell - \lambda^F r^F) & \text{otherwise.} \end{cases}$$

To describe the encoding vectors of our code, we decompose each encoding vector of the base code into λ^F vectors of length k^F, corresponding to the data associated with each final codeword. Then, we represent each of these vector in the αk^I-dimensional space corresponding to the whole data $\mathbf{m}$ (by filling the additional dimensions with zeros). Specifically, we define $\mathbf{p}_{t,\ell}^{(i)} \in \mathbb{F}_q^{\alpha k^I}$ as the column vector such that $\mathbf{m}\mathbf{p}_{t,\ell}^{(i)}$ corresponds to the encoding of the data under the base code for parity $t \in [r^I]$, final codeword $i \in [\lambda^F]$, and instance $\ell \in [\alpha]$. We achieve this by setting $\mathbf{p}_{t,\ell}^{(i)}[(i-1)k^F\alpha + (j-1)\alpha + \ell] := \mathbf{h}_t[(i-1)k^F + j]$ for $j \in [k^F]$ and 0 everywhere else.

We specify how to construct $\mathbf{q}_{t,\ell}^I \in \mathbb{F}_q^{\alpha k^I}$, which is the encoding vector for instance $\ell \in [\alpha]$ of parity $t \in [r^I]$ of the initial codeword, and $\mathbf{q}_{t,\ell}^{F(i)} \in \mathbb{F}_q^{\alpha k^I}$ which is the encoding vector for instance $\ell \in [\alpha]$ of parity $t \in [r^F]$ of final codeword $i \in [\lambda^F]$. The construction is designed so that the final codewords are all encoded under the same final code. Figure 4 shows a diagram for this construction. The construction has three important elements:

1) *Permutation*: In the initial code, the first λ^F blocks of the data symbols associated with final codeword i are circularly shifted to the right $i-1$ times (denoted with letters **A**-**C**). This reordering is logical (no data is moved) and used for describing the code only.
2) *Projection*: For parities 1 through r^F (**P** blocks), we use the base code without modification to encode each data column. During conversion, we download blocks $\{2, \ldots, \lambda^F\}$ from each data symbol (blocks **B** and **C**) and subtract their interference from the corresponding parity symbols to obtain the first block of each final codeword (**P**) blocks).
3) *Piggybacks*: For parities (r^F+1) through r^I (**Q** blocks), we use the base code and add piggybacks to block $\ell_1 \in [\lambda^F]$ that contain the subsymbols of block $(\lambda^F + 1)$ of final codeword ℓ_1 (transposed). During conversion, we recover the piggybacks by using the downloaded data (blocks **B** and **C**). Note that the piggybacks will still have extra data remaining from the unaccessed block (**A**). However, the final code can still be sequentially decoded (the same way that codes in the piggyback framework are decoded).

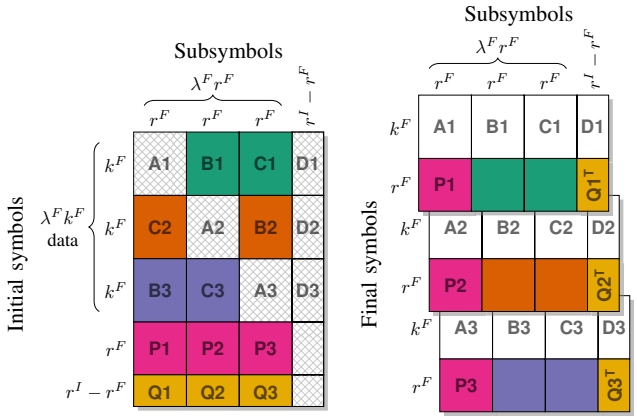

Fig. 4: Diagram of convertible code construction in the split regime when $r^I > r^F$ and $\lambda^F = 3$.

The remaining parity subsymbols are generated from the accessed data blocks (**B** and **C**). Finally, parity symbol $t \in [r^F]$ in final codeword $i \in [\lambda^F]$ is scaled by $\xi_t^{-(i-1)k^F}$ to ensure that all final codewords are encoded by the same final code (as described in §IV-1). Let $\overrightarrow{\ell}(i) := ((\ell_1 - i \bmod \lambda^F)k^F + \ell_2)$ be the instance index after permutation. Then, the encoding vectors for the initial and final codes are defined as:

$$\mathbf{q}_{t,\ell}^I := \begin{cases} \sum_{i=1}^{\lambda^F} \mathbf{p}_{t,\overrightarrow{\ell}(i)}^{(i)} & \text{if } t \leq r^F,\ \ell_1 \leq \lambda^F, \\ \sum_{i=1}^{\lambda^F} \mathbf{p}_{t,\overrightarrow{\ell}(i)}^{(i)} + \underbrace{\mathbf{p}_{\ell_2,(\lambda^F-1)r^F+t}^{(\ell_1)}}_{\text{piggyback}} & \text{if } t > r^F,\ \ell_1 \leq \lambda^F, \\ \sum_{i=1}^{\lambda^F} \mathbf{p}_{t,\ell}^{(i)} & \text{otherwise.} \end{cases}$$

$$\mathbf{q}_{t,\ell}^{F(i)} := \begin{cases} \xi_t^{-(i-1)k^F} \mathbf{p}_{t,\ell}^{(i)} & \text{if } \ell_1 \leq \lambda^F, \\ \underbrace{\xi_t^{-(i-1)k^F}}_{\text{scaling factor}} (\mathbf{p}_{t,\ell}^{(i)} + \underbrace{\mathbf{p}_{r^F+\ell_2,t}^{(i)}}_{\text{extra data}}) & \text{otherwise.} \end{cases}$$

This construction achieves the bound of Theorem 7. Furthermore, the constructed code is MDS because it uses the piggyback framework and the base code is MDS.

REFERENCES

[1] F. Maturana and K. V. Rashmi, "Bandwidth cost of code conversions in the split regime," 2022. [Online]. Available: https://arxiv.org/abs/2205.06793

[2] S. Ghemawat, H. Gobioff, and S. Leung, "The Google file system," in *Proceedings of the 19th ACM Symposium on Operating Systems Principles 2003, SOSP 2003, Bolton Landing, NY, USA, October 19-22, 2003*, M. L. Scott and L. L. Peterson, Eds. ACM, 2003, pp. 29–43.

[3] D. Borthakur, R. Schmidt, R. Vadali, S. Chen, and P. Kling, "HDFS RAID - Facebook," available on: http://www.slideshare.net/ydn/hdfs-raid-facebook. Accessed: 2019-07-23.

[4] C. Huang, H. Simitci, Y. Xu, A. Ogus, B. Calder, P. Gopalan, J. Li, and S. Yekhanin, "Erasure coding in Windows Azure storage," in *2012 USENIX Annual Technical Conference, Boston, MA, USA, June 13-15, 2012*, G. Heiser and W. C. Hsieh, Eds. USENIX Association, 2012, pp. 15–26. [Online]. Available: https://www.usenix.org/conference/atc12/technical-sessions/presentation/huang

[5] Apache Software Foundation, "Apache hadoop: HDFS erasure coding," available on: https://hadoop.apache.org/docs/r3.0.0/hadoop-project-dist/hadoop-hdfs/HDFSErasureCoding.html. Accessed: 2019-07-23.

[6] S. Kadekodi, K. V. Rashmi, and G. R. Ganger, "Cluster storage systems gotta have HeART: improving storage efficiency by exploiting disk-reliability heterogeneity," in *17th USENIX Conference on File and Storage Technologies, FAST 2019, Boston, MA, February 25-28, 2019*, A. Merchant and H. Weatherspoon, Eds. USENIX Association, 2019, pp. 345–358. [Online]. Available: https://www.usenix.org/conference/fast19/presentation/kadekodi

[7] F. Maturana and K. V. Rashmi, "Convertible codes: new class of codes for efficient conversion of coded data in distributed storage," in *11th Innovations in Theoretical Computer Science Conference, ITCS 2020, January 12-14, 2020, Seattle, Washington, USA*, ser. LIPIcs, T. Vidick, Ed., vol. 151. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020, pp. 66:1–66:26.

[8] F. Maturana, V. S. C. Mukka, and K. V. Rashmi, "Access-optimal linear MDS convertible codes for all parameters," in *IEEE International Symposium on Information Theory, ISIT 2020, Los Angeles, California, USA, June 21-26, 2020*, 2020.

[9] F. Maturana and K. V. Rashmi, "Bandwidth cost of code conversions in distributed storage: fundamental limits and optimal constructions," in *IEEE International Symposium on Information Theory, ISIT 2021, Melbourne, Australia, July 12-20, 2021*. IEEE, 2021, pp. 2334–2339.

[10] K. V. Rashmi, N. B. Shah, and K. Ramchandran, "A piggybacking design framework for read-and download-efficient distributed storage codes," in *2013 IEEE International Symposium on Information Theory, ISIT 2013, Istanbul, Turkey, July 7-12, 2013*. IEEE, 2013, pp. 331–335.

[11] ——, "A piggybacking design framework for read-and download-efficient distributed storage codes," *IEEE Transactions on Information Theory*, vol. 63, no. 9, pp. 5802–5820, 2017.

[12] K. V. Rashmi, N. B. Shah, and P. V. Kumar, "Enabling node repair in any erasure code for distributed storage," in *2011 IEEE International Symposium on Information Theory Proceedings, ISIT 2011, St. Petersburg, Russia, July 31 - August 5, 2011*, A. Kuleshov, V. M. Blinovsky, and A. Ephremides, Eds. IEEE, 2011, pp. 1235–1239.

[13] S. Mousavi, T. Zhou, and C. Tian, "Delayed parity generation in MDS storage codes," in *2018 IEEE International Symposium on Information Theory, ISIT 2018, Vail, CO, USA, June 17-22, 2018*. IEEE, 2018, pp. 1889–1893.

[14] M. Xia, M. Saxena, M. Blaum, and D. Pease, "A tale of two erasure codes in HDFS," in *Proceedings of the 13th USENIX Conference on File and Storage Technologies, FAST 2015, Santa Clara, CA, USA, February 16-19, 2015*, J. Schindler and E. Zadok, Eds. USENIX Association, 2015, pp. 213–226. [Online]. Available: https://www.usenix.org/conference/fast15/technical-sessions/presentation/xia

[15] X. Su, X. Zhong, X. Fan, and J. Li, "Local re-encoding for coded matrix multiplication," in *IEEE International Symposium on Information Theory, ISIT 2020, Los Angeles, California, USA, June 21-26, 2020*, 2020.

[16] G. Zhang, W. Zheng, and J. Shu, "ALV: A new data redistribution approach to RAID-5 scaling," *IEEE Transactions on Computers*, vol. 59, no. 3, pp. 345–357, 2010.

[17] W. Zheng and G. Zhang, "Fastscale: accelerate RAID scaling by minimizing data migration," in *9th USENIX Conference on File and Storage Technologies, San Jose, CA, USA, February 15-17, 2011*, G. R. Ganger and J. Wilkes, Eds. USENIX, 2011, pp. 149–161. [Online]. Available: http://www.usenix.org/events/fast11/tech/techAbstracts.html#Zheng

[18] C. Wu and X. He, "GSR: A global stripe-based redistribution approach to accelerate RAID-5 scaling," in *41st International Conference on Parallel Processing, ICPP 2012, Pittsburgh, PA, USA, September 10-13, 2012*. IEEE Computer Society, 2012, pp. 460–469.

[19] G. Zhang, W. Zheng, and K. Li, "Rethinking RAID-5 data layout for better scalability," *IEEE Transactions on Computers*, vol. 63, no. 11, pp. 2816–2828, 2014.

[20] J. Huang, X. Liang, X. Qin, P. Xie, and C. Xie, "Scale-RS: an efficient scaling scheme for RS-coded storage clusters," *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 6, pp. 1704–1717, 2015.

[21] S. Wu, Y. Xu, Y. Li, and Z. Yang, "I/O-efficient scaling schemes for distributed storage systems with CRS codes," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 9, pp. 2639–2652, 2016.

[22] X. Zhang, Y. Hu, P. P. C. Lee, and P. Zhou, "Toward optimal storage scaling via network coding: from theory to practice," in *2018 IEEE Conference on Computer Communications, INFOCOM 2018, Honolulu, HI, USA, April 16-19, 2018*. IEEE, 2018, pp. 1808–1816.

[23] Y. Hu, X. Zhang, P. P. C. Lee, and P. Zhou, "Generalized optimal storage scaling via network coding," in *2018 IEEE International Symposium on Information Theory, ISIT 2018, Vail, CO, USA, June 17-22, 2018*. IEEE, 2018, pp. 956–960.

[24] X. Zhang and Y. Hu, "Efficient storage scaling for MBR and MSR codes," *IEEE Access*, vol. 8, pp. 78 992–79 002, 2020.

[25] B. K. Rai, V. Dhoorjati, L. Saini, and A. K. Jha, "On adaptive distributed storage systems," in *IEEE International Symposium on Information Theory, ISIT 2015, Hong Kong, China, June 14-19, 2015*. IEEE, 2015, pp. 1482–1486.

[26] B. K. Rai, "On adaptive (functional MSR code based) distributed storage systems," in *2015 International Symposium on Network Coding, NetCod 2015, Sydney, Australia, June 22-24, 2015*. IEEE, 2015, pp. 46–50.

[27] S. Wu, Z. Shen, and P. P. C. Lee, "On the optimal repair-scaling trade-off in locally repairable codes," in *2020 IEEE Conference on Computer Communications, INFOCOM 2020, Virtual Conference, July 6-9, 2020*. IEEE, 2020.

[28] Y. Hu, X. Zhang, P. P. C. Lee, and P. Zhou, "NCScale: toward optimal storage scaling via network coding," *IEEE/ACM Transactions on Networking*, pp. 1–14, 2021.

[29] R. W. Yeung, *A First Course in Information Theory*. Boston, MA: Springer US, 2002.