Multi-version Coding for Consistent Distributed Storage of Correlated Data Updates

Ramy E. Ali and Viveck R. Cadambe

Abstract

Motivated by applications of distributed storage systems to cloud-based key-value stores, the multi-version coding problem has been recently formulated to store frequently updated data in asynchronous distributed storage systems. Inspired by consistency requirements in distributed computing, the main goal in multi-version coding is to ensure that the latest possible version of the data is decodable, even if all the data updates have not reached all the servers in the system. In this paper, we study the storage cost of ensuring consistency for the case where the various versions of the data are correlated, in contrast to previous work where versions were treated as being independent. We provide multi-version code constructions that show that the storage cost can be significantly smaller than the previous constructions depending on the degree of correlation between the different versions of the data. Our constructions are based on update-efficient codes, Reed-Solomon code, random binning coding and BCH codes.

Specifically, we consider the multi-version coding setting with \( n \) servers and \( \nu \) versions, with a failure tolerance of \( n - c \), where the \( \nu \) message versions form a Markov chain. The message version is distributed uniformly over all binary vectors of length \( K \), and given a version, the subsequent version is uniformly distributed in a Hamming ball of radius \( \delta_K K \) centered around that given version. Previously derived achievable schemes have a per-server storage cost of at least \( \frac{1}{2} \nu K - o(K) \) when \( \nu \leq c \), whereas we propose a scheme that has a storage cost of \( \frac{K}{c} + \frac{\nu - 1}{c} \log \text{Vol}(\delta_K K, K) \), where \( \text{Vol}(r, K) \) is the volume of a ball of radius \( r \) in the \( K \) dimensional Hamming cube. Through a converse result, we show that our multi-version code constructions are nearly-optimal in certain correlation regimes.

I. Introduction

Distributed key-value stores are an important part of modern cloud computing infrastructure. Key-value stores are commonly used by several applications including reservation systems, transactions and multi-player gaming. Owing to their utility, there are numerous commercial and open-source cloud-based key-value store implementations such as Amazon Dynamo [2], Apache Cassandra [3], and CouchDB [4]. In typical distributed key-value stores, the data stored is updated frequently, and the time scales of data updates and access are often comparable to the time scales of dispersing the data to the servers (See [2]). In such settings, ensuring that a client gets the latest, most updated, version of the data can be challenging as the data updates may not have reached all servers. The notion that the latest version of the data must be accessible to the users despite the frequent updates is known as consistency in computer science [2], [5], [6]. The design of a consistent distributed storage system is a topic that has been widely studied in theory of distributed systems (See [5], [7] and references therein). In addition to cloud computing, maintaining consistency in several computing applications where there is data redundancy and frequent updates, for instance for caching in edge computing systems (See [8], [9]).
Recently, the multi-version coding problem \cite{7}, \cite{10}, \cite{11} has been formulated to study the storage cost of ensuring consistency in distributed systems from an information theoretic viewpoint. The analysis of \cite{7}, \cite{10}, \cite{11} was conducted under the modeling assumption that different versions of the data are independent of each other. In this paper, we study the impact of correlations among the different data versions in the multi-version coding framework.

The shared memory emulation problem is a distributed computing theoretic abstraction, that models distributed key-value store implementations. The goal of the shared memory emulation problem is to implement a read-write variable over a distributed system of servers. The multi-version coding problem studies codes for the shared memory emulation problem from an information theoretic perspective. Inspired by the shared memory emulation model, the multi-version coding problem differs from previous studies of distributed storage in information theory, e.g., \cite{12}, \cite{13}, in the following aspects:

1) Asynchrony: a new version of the data may not arrive at all servers simultaneously.
2) Decentralized nature: there is no single encoder that has knowledge of all the versions of the data simultaneously, and a server is not aware of which versions are received by other servers; instead, servers store codeword symbols of the old versions, and then update the stored symbols or append to storage based on the newly arrived data version.
3) Consistency: a client accessing the storage system must be able to retrieve the latest possible version of the data by connecting to a subset of the servers; the cardinality of the subset of the servers that need to make the data accessible is directly related to the desired fault tolerance in the system.

A classical erasure coding approach does not directly apply under the above assumptions. Unlike in replication where servers store the entire data, in erasure coding, servers store codeword symbols and the decoder has to access a sufficient number of servers to recover the data. As a consequence, when a new version arrives in the system, a server cannot directly update the codeword symbol until the version has propagated to a sufficient number of servers. Specifically, in systems with frequently updated data, servers have to store several versions of the data, thereby offsetting the storage cost benefits of erasure coding. The multi-version coding problem analyses the extent of this offset from an information theoretic perspective. By deriving achievable schemes and impossibility results for the multi-version coding framework, reference \cite{7} showed that, interestingly, there is an inevitable price in terms of storage cost for maintaining consistency in an asynchronous, decentralized, distributed storage system. Furthermore, the storage cost grows with the degree of asynchrony in the system.

In this paper, we show that the storage cost can be significantly reduced if the degree of correlation between subsequent updates is significant. We consider a variant of the multi-version coding problem as follows. We consider a distributed storage system with \(n\) servers and the objective is to store a message with \(\nu\) versions, which are totally ordered; messages with higher ordering are referred to as later versions, and lower ordering as earlier versions. Each server receives an arbitrary subset of the \(\nu\) versions, and encodes them. A client who connects to any \(c\) servers can obtain the latest common version among them, as per the total ordering, or a later version. The goal of the multi-version coding is to minimize the storage cost. Here, we assume that each message version is \(K\) bits long, and model the correlation between successive versions in terms of the bit-strings that represent them. More specifically, given a version of the message, we assume that the next version is uniformly distributed in the Hamming ball of radius \(\delta_K K\), centered around that given version.

The multi-version coding problem studied previously \cite{7} assumed that the versions are independent, i.e., \(\delta_K = 1\). In this classical erasure coding model, where \(\nu = 1\), the Singleton bound dictates that the storage cost per server is at least \(\frac{K}{c}\). Note however that for \(\nu > 1\), a server cannot simply store the codeword symbol corresponding to one version, since another server may not have received the corresponding version. In the multi-version coding setting, reference \cite{7} showed via an achievable schemes

---

\(^2\)The shared memory emulation problem, being a theoretical abstraction, is actually a much simplified view of key-value stores. Nonetheless, algorithms and design principles of key-value implementations have much in common with solutions to the shared memory emulation problem.
and converse that, the storage cost per server is at least \( \frac{1}{2} \nu K \delta + o(K) \) when \( \nu \le c \). Since the per-server cost of storing each version is \( \frac{K}{c} \), we may interpret the result as follows: when the versions are independent, to compensate for the asynchrony and still maintain consistency, the servers have to store - from a cost perspective - at least half of all the \( \nu \) versions. In this paper, we derive novel achievable schemes and an approximately tight converse for all regimes of \( \delta_K \). Our results significantly improve upon the result of [7] when the correlation is significant, specifically, when \( \delta_K < 1/2 \). In particular, for the case where \( \delta_K K = m \) - that is, \( m \) bits change in each successive version - we propose a scheme that has a per-server storage cost of \( \frac{K}{c} + (\nu - 1) \frac{m \log K}{c} + o(\log K) \). For the case where \( \delta_K = \delta \), that is the number of bits that change in successive versions is proportional to \( K \), our scheme has a per-server storage cost of \( \frac{K}{c} + (\nu - 1) \frac{K \delta}{c} + o(\log K) \). We derive a converse that shows that for the case of \( \delta_K = \delta \), our scheme is approximately optimal.

Related Work

The idea of exploiting the correlation between the different versions to efficiently update, store or exchange data has a rich history of study in network information theory [14]. In their classic work, Slepian and Wolf [15] studied the problem of compressing correlated distributed sources, where the objective is decoding the data of all sources. Linear code constructions that approach the Slepian-Wolf limits have been proposed in [16]–[20] and references therein.

The idea of exploiting correlated information to design storage and communication systems has been studied for several modern and emerging applications in recent times. Reference [21] considers simultaneous and sequential interactive data exchange problems between two users. By leveraging Slepian-Wolf and belief propagation techniques, practical schemes have been proposed that exploit the correlation to minimize the number of exchanged bits between the users. The problem of encoding incremental updates efficiently is the motivation of the delta encoding/compression techniques used commonly in data storage. References [22], [23], and references therein refine the notion of delta compression by modeling the data updates using the edit distance; in particular, the references develop coding schemes that synchronize a small number of edits between a client and a server efficiently.

The study of exploiting correlations between data updates to improve efficiency in a distributed storage setting, where multiple servers store codeword symbols corresponding to the data, has been of significant recent interest [24]–[30]. References [24], [25] devise coding schemes that use as input, the old and new versions of the data, and outputs an \( n \)-length code that can be used to store both versions of the data efficiently in a distributed storage system. References [26], [27] study capacity-achieving update-efficient codes for binary symmetric and erasure channels, where a small change in the message leads to a codeword which is close to the original codeword. Hence, the constructed codes lead to efficient updates of data in distributed storage systems. We use the results of [26], [27] in one of the multi-version coding constructions in our paper. Reference [30] studies the problem of storing correlated data for distributed caching and content distribution, and shows the utility of the classical Gray-Wyner source coding [31] in the studied application.

Reference [28] studied the communication cost of updating a “stale” server that did not get an updated message, by downloading data from already updated servers in such distributed setting. The reference proposed code constructions and tight bounds for this problem. We note that the problem of [28] has some modeling elements that are common with our approach, specifically, there is a limited degree of asynchrony - a single update does not a reach a particular server. A side information problem is presented in [29], where the goal is to send an updated version to a remote entity that has as side information, an arbitrary linear transform of an old version. The reference shows that the optimal encoding function is related to a maximally recoverable subcode of the linear transform associated with the side information. The problem of [29] is of peripheral interest to some of the solutions of our paper, since we aim to store use codeword symbols of some versions as side information to store other versions of the data.

Although our problem formulation and solutions have some common ingredients with previous works, our setting differs from all the previous works because we are motivated by the shared memory emulation
problem, where the data update time scales are similar to the message propagation time scales in the network. Specifically, in our setting (a) each server in the storage system receives an arbitrary set of message versions, and (b) no node in the system is aware of the versions received by any other node in the system. An important outcome of our study is that correlation between versions can be used to reduce storage costs in distributed systems, despite the asynchrony, decentralized nature and consistency requirements.

Organization of this paper

The rest of this paper is organized as follows. Section II presents the multi-version coding problem, background and the results of [7]. We provide a summary of the main results in Section III. In Section IV, we develop update-efficient multi-version code constructions. In Section V, we construct multi-version codes that are motivated by Slepian-Wolf coding. Section VI provides a lower bound on the storage cost. Finally, conclusions are discussed in Section VII.

II. System Model and Background of Multi-version Codes

We begin with some notation. We use boldface for vectors and capital letters for random variables/vectors. In the $n$-dimensional Euclidean space, the standard basis vectors are denoted by $\{e_1, e_2, \ldots, e_n\}$. For a vector $x$, we denote its Hamming weight by $w_H(x)$. For any two vectors $x_1$ and $x_2$, we denote the Hamming distance between these two vectors by $d_H(x_1, x_2)$. For a positive integer $i$, we denote by $[i]$ the set $\{1, 2, \ldots, i\}$. For any set of ordered indices $S = \{s_1, s_2, \ldots, s_{|S|}\} \subseteq \mathbb{Z}$ where $s_1 < s_2 < \cdots < s_{|S|}$, and for any ensemble of variables $\{X_i : i \in S\}$, the tuple $(X_{s_1}, X_{s_2}, \ldots, X_{s_{|S|}})$ is denoted by $X_S$.

We use BEC($p$) to denote a binary erasure channel with erasure probability $p$. We use $\log(.)$ to denote the logarithm to the base 2 and $H(.)$ to denote the binary entropy function. We will sometimes use the notation $[2^K]$ to denote the set of $K$-length binary strings, that is, we assume an implicit mapping between $[2^K]$ and $\{0, 1\}^K$. Finally, an code of length $n$ and dimension $k$ over alphabet $A$ consists of a mapping $C : A^k \rightarrow A^n$. We refer to a code $C$ of length $n$ and dimension $k$ as an $(n, k)$ code.

Sometimes, to keep notation terse, we use the abbreviation MVC to refer to multi-version codes.

A. Multi-version Codes (MVCs)

![Fig. 1: A distributed storage system with $n = 4$ servers storing $\nu = 2$ versions. A user can recover the latest common version or a later version from any $c = 2$ servers.](image-url)
We now present a variant of the definition of the multi-version code from [7], where we model correlations between the various message versions. We consider a distributed storage system with \( n \) servers. The objective is to store \( \nu \) possibly correlated versions of the message where \( W_i \in [2^K] \) is the \( i \)-th version, \( i \in [\nu] \), and \( K \) is the message length in bits. The versions are assumed to be totally ordered, i.e., if \( i > j \), \( W_i \) is interpreted as a later version with respect to \( W_j \). We assume that \( W_1 \rightarrow W_2 \rightarrow \ldots \rightarrow W_\nu \) form a Markov chain. \( W_1 \) is uniformly distributed over the set of all \( K \) length binary vectors. Given \( W_m, W_{m+1} \) is in a Hamming ball of radius \( \delta K \), \( B(W_m, \delta K) \), where

\[
B(W_m, \delta K) = \{ W : d_H(W, W_m) \leq \delta K \}.
\]

We denote the volume of the ball by

\[
Vol(\delta K, K) = |B(W_m, \delta K)| = \delta K^K. \tag{2}
\]

**Remark 1.** Note that for \( \delta K = \delta \), where \( \delta < 1/2 \) is a constant, we have

\[
KH(\delta) - o(K) \leq \log Vol(\delta K, K) \leq KH(\delta), \tag{3}
\]

where the last inequality follows from Stirling’s inequality [32]. We use the above approximation for expository purposes in parts of the paper.

The \( i \)-th server receives an arbitrary subset of versions \( S(i) \subseteq [\nu] \). We refer to the subset of versions available at the \( i \)-th server, \( S(i) \), as the state of that server. We denote the system state by \( S = \{S(1), S(2), \ldots, S(n)\} \in \mathcal{P}([\nu])^n \), where \( \mathcal{P}([\nu]) \) denotes the power set of \( [\nu] \). For the \( i \)-th server with state \( S = S(i) = \{s_1, s_2, \ldots, s_{|S|}\} \), where \( s_1 < s_2 < \cdots s_{|S|} \), the server stores a codeword symbol which is generated by the encoding function \( \varphi_S^{(i)} \) that takes an input \( W_S \) and outputs an element in \( [q] \). For any set of servers \( T \subseteq [n] \), \( u_L = \max_{i \in T} S(i) \) denotes the latest common version among these servers. The objective of the multi-version coding is to design encoding functions that minimizes the storage cost such that the latest common version or a later version of the message is decodable from any set \( T \subseteq [n] \) of \( c \) servers. That is, the system can tolerate any \( n - c \) server failures (erasures). Fig. 1 depicts a possible scenario with \( n = 4 \) servers.

We next provide a formal definition for the multi-version coding formulation.

**Definition 1** (\( \epsilon \)-error Multi-version code (MVC)). An \( \epsilon \)-error \((n, c, \nu, 2^K, q, \delta_K)\) multi-version code (MVC) consists of the following for \( \epsilon > 0 \)

- encoding functions

\[
\varphi_S^{(i)} : [2^K]^{\mid S \mid} \to [q], \text{ for every } i \in [n] \text{ and every state } S \subseteq [\nu],
\]

- decoding functions

\[
\psi_S^{(T)} : [q]^c \to [2^K] \cup \{\text{NULL}\},
\]

that satisfy the following

\[
P(S,T) := \Pr \left[ \psi_S^{(T)} \left( \varphi_{S(t_1)}^{(t_1)}, \ldots, \varphi_{S(t_c)}^{(t_c)} \right) = W_m, \text{ for some } m \geq u_L, \text{ if } \cap_{i \in T} S(i) \neq \emptyset \right] \geq 1 - \epsilon,
\]

for every possible system state \( S \in \mathcal{P}([\nu])^n \) and every set of servers \( T \subseteq [n] \), where \( T = \{t_1, t_2, \ldots, t_c\}, t_1 < t_2 < \cdots < t_c \).

**Definition 2** (Storage Cost of a Multi-version Code). The storage cost of an \( \epsilon \)-error \((n, c, \nu, 2^K, q, \delta_K)\) MVC is equal to \( \log q \) bits.

Reference [7] studied 0-error MVCs with independent versions, that is, the special case of \( \epsilon = 0, \delta_K = 1 \).
Remark 2. In contrast to the original multi-version coding framework [7], our correctness requirement is probabilistic, that is, we require correct decoding with a probability that is at least \(1 - \epsilon\). This relaxation leads to storage cost savings in the setting of correlated versions.

We next describe the results of [7], and provide new achievable schemes when the different versions are correlated.

B. Background - Replication and Simple Erasure Coding

Replication and simple maximum distance separable (MDS) codes provide two natural MVC constructions. Suppose that state of the \(i\)-th server is \(S = \{s_1, s_2, \ldots, s_{|S|}\}\), where \(s_1 < s_2 < \ldots < s_{|S|}\).

- Replication based MVCs: In this scheme, each server stores the latest version it receives. The encoding function is \(\varphi_S^{(i)}(W_S) = W_{s_i}\). Therefore, the storage cost per server is \(K\), that is, the number of bits stored by a server is the number of bits in one version.

- Simple MDS codes based MVCs (MDS-MVCs): In this scheme, an \((n, c)\) MDS code is used to encode each version separately. Specifically, suppose that \(C : [2^K] \to [2^{K/c}]^n\) is an \((n, c)\) MDS code over alphabet \([2^{K/c}]\), and suppose that \(C^{(i)} : [2^K] \to [2^{K/c}]\) denotes the \(i\)-th co-ordinate of the output of \(C\). The MVC is constructed as \(\varphi_S^{(i)}(W_S) = (C^{(i)}(W_{s_1}), C^{(i)}(W_{s_2}), \ldots, C^{(i)}(W_{s_{|S|}}))\). That is, each server stores one codeword symbol for each version it receives. In the worst-case where a server receives all versions, the storage cost is \(\nu K/c\).

Because of the distributed nature of the setting, no server is aware of what versions are present at other servers. Therefore, when MDS codes are used, a server has to store codeword symbols corresponding to multiple versions as the latest version at the server may not have propagated to a sufficient number of servers. In contrast, with replication a server stores only the latest version.

Reference [7] developed multi-version coding schemes and converse results. An important outcome of the study of [7] is that, when the different versions are independent, i.e., if \(\delta_K = 1\), then the storage cost cannot be smaller than \(\nu K/c + \epsilon - o(K)\). In particular, the best possible MVC scheme is, approximately, at most twice as cost-efficient as the better among replication and simple erasure coding.

In this paper, we show that replication and simple erasure coding are significantly inefficient if the different versions are correlated, i.e., if \(\delta_K\) is smaller than 1. Our schemes resemble simple erasure codes in their construction; however, we exploit the correlation between the versions to store fewer bits per server.

III. MAIN RESULTS

In this section, we present the main results of the paper, describing the storage costs of various multi-version coding schemes and impossibility results. The constructions and the proofs are presented in later sections. We begin with some simple schemes in Section III-A that set a comparison point for our schemes. We then describe our code constructions in Sections III-B and III-C. We present our converse results in Section III-D. We tabulate all results in Table I.

A. Motivating scheme

Consider a simple MDS codes based multi-version coding (MDS-MVC) scheme of Section III-B, assume that we use a Reed-Solomon code over a field \(F_p\) of binary characteristic. The generator matrix of a Reed-Solomon code is usually expressed over \(F_p\). However, every element in \(F_p\) is a vector over \(F_2\), and a multiplication over the extension field \(F_p\) is a linear transformation over \(F_2\). Therefore, the generator matrix of the Reed-Solomon code can be equivalently expressed over \(F_2\) as follows

\[ G = (G^{(1)}, G^{(2)}, \ldots, G^{(n)}) \]

where \(G\) is a \(K \times nK/c\) binary generator matrix, and \(G^{(i)}\) has dimension \(K \times K/c\). Because Reed-Solomon codes can tolerate \(n - c\) faults, we know that every matrix of the form \((G^{(t_1)}, G^{(t_2)}, \ldots, G^{(t_c)})\), where
In update-efficient MVC schemes, a server stores apply ideas of delta-compression [34] to store the codeword symbols of the received versions at the servers. In the scheme of Section II-B, the server stores all codeword symbols for successive versions; instead, it may only store the symbols that change from the previous version and their positions. If the versions are encoded using update-efficient codes - where a server stores only a single bit changes in every subsequent version, i.e., \(\delta\) of the previous version and their positions. If the versions are encoded using update-efficient codes [26], the server finds a difference vector \(\mathbf{y}_1^G\) from the \(i\)-th server. The decoder that connects to any \(c\) servers can decode the latest common version \(u_L\) among these servers. This is because, a decoder that connects to server \(i\) can compute \(W_{ul}^T - W_{sl}\) and then evaluate \(W_{sl}^T G(i) + (W_{ul}^T - W_{sl}^T) G(i)\) to get \(W_{ul}^T G(i)\), thereby obtaining sufficient information from \(c\) servers to recover \(W_{ul}\). The worst-case storage cost associated with the case where the server receives all versions - is given by \(\frac{K}{c} + (\nu - 1) \log V o l(\delta_K K, K)\).

The scheme, albeit simple, is quite powerful if \(\delta_K\) is small. For instance, in the special case where only a single bit changes in every subsequent version, i.e., \(\delta_K = 1/K\), the above scheme stores \(\frac{K}{c} + (\nu - 1) \log (K + 1)\) bits, which is significantly smaller than the storage cost of both erasure coding and replication. A desirable feature of this scheme is that a priori knowledge of the parameter \(\delta_K\) is not required. However, the scheme is not causal; having stored a codeword symbol corresponding to \(W_{sl}\), it is not clear how the server would compute \(W_{s_2} - W_{s_1}\).

We now explain a scheme that is causal, whereas, it requires knowledge of the parameter \(\delta_K\). The scheme is also a modification of the simple erasure coding based scheme of Section II-B. Suppose that the \(i\)-th server receives the set of versions \(S = \{s_1, s_2, \cdots, s_{|S|}\} \subseteq [\nu]\). The server encodes \(W_{s_i}\) using the binary code as \(W_{s_i}^T G(i)\). For \(W_{sm}\), where \(m > 1\), the server finds a difference vector \(\mathbf{y}_{sm,m-1}^G\) that satisfies the following:

1. \(\mathbf{y}_{sm,m-1}^T G(i) = (W_{sm}^T - W_{sm-1}^T) G(i)\) and
2. \(w_H(\mathbf{y}_{sm,m-1}^G) \leq |s_m - s_{m-1}| \delta_K K\).

Note that it is not necessary (and even possible to ensure) that \(\mathbf{y}_{sm,m-1}^G = W_{sm} - W_{sm-1}\). Also note that \(\mathbf{y}_{sm,m-1}\) can be represented by \(\log V o l(|s_m - s_{m-1}| \delta_K K, K)\) bits. It is easy to see that the decoder can obtain \(W_{ul}^T G(i)\) from the \(i\)-th server by applying \(W_{sl}^T G(i) + (W_{ul}^T - W_{sl}^T) G(i)\). Therefore, the decoder can obtain the latest common version among any \(c\) servers. The worst-case storage cost is again, \(\frac{K}{c} + (\nu - 1) \log V o l(\delta_K K, K)\). Now, the two schemes we described above motivate the following two questions.

**Q1:** Can we obtain a causal MVC construction that is oblivious to the parameter \(\delta_K\) with a storage cost of \(\frac{K}{c} + (\nu - 1) \log V o l(\delta_K K, K)\)?

**Q2:** Can we obtain a MVC construction - possibly even non-causal and non-oblivious to the parameter \(\delta_K\) - with a significantly smaller cost as compared with \(\frac{K}{c} + (\nu - 1) \log V o l(\delta_K K, K)\)?

Our results of Section III-B provide relatively elementary code constructions based that address question Q1. The schemes associated with the results of Section III-B use the idea of update-efficient codes [26, 27]. Our results of Section III-C address question Q2. The constructions associated with Section III-B are inspired by Cover’s random binning proof of the Slepian-Wolf source coding problem [33].

### B. Update-efficient Multi-version Codes

In Section IV, we develop MVC constructions that exploit the correlation to reduce the storage cost. Our update-efficient MVCs are essentially modifications of the MDS-MVCs of Section II-B where we apply ideas of delta-compression [34] to store the codeword symbols of the received versions at the servers. Recall that in MDS-MVCs a server stores a codeword symbol for each version that it receives. In update-efficient MVC schemes, a server stores \(\frac{K}{c}\) bits for the first version that it receives, similar to the scheme of Section II-B. However, unlike the scheme of Section II-B, a server may not store all codeword symbols for successive versions; instead, it may only store the symbols that change from the previous version and their positions. If the versions are encoded using update-efficient codes - where a
small number of changes in message symbols lead to only a few codeword symbols changing - then we can obtain MVCs that are more storage-efficient as compared to MDS-MVCs. We describe the results for two classes of update-efficient codes next: the Kolchin-Generator (KG) construction of [26], [35], and an interpretation of Reed-Solomon codes.

1) KG Update-Efficient MVCs: Reference [26] showed that there exist a sequence of codes with rate arbitrary close to the capacity of the binary erasure channel, indexed by the number of bits encoded $K$, with vanishing probability of error such that their update efficiency - the number of bits in the codeword that change when a single bit in the message changes - is $O(\log K)$. In our model, each successive version differs in at most $\delta_K K$ bits as compared with the previous version of the message. If we use MDS-MVCs with generator matrices chosen as per the construction of [26], then for each subsequent version we only need to store the positions of at most $\delta_K K O(\log K)$ codeword bits that differ from the codeword of the previous version. For a single server storing approximately $\frac{K}{c}$ bits of the first version it receives, storing the position of each change in the codeword of the subsequent version incurs a cost of $\log \frac{K}{c}$ bits. Since the update-efficient code construction of [26] ensures that the number of bits that change for each subsequent version is $\delta_K K O(\log K)$ bits, and there are $\nu - 1$ subsequent versions, the storage cost of the scheme is $\frac{K}{c} + (\nu - 1) \delta_K K O(\log^2 K) + o(\log^2 K)$. We state our result formally next.

**Theorem 1.** [KG Update-efficient MVC] For every $0 < \alpha < 1, \gamma > 1$, there exists a sequence of $(n, c, \nu, 2^K, q, \delta_K)$ multi-version codes, indexed by $K$, whose worst-case storage cost is

$$\frac{K}{c} + \frac{\nu - 1}{c} \min(\delta_K K \gamma (\ln K_\alpha/c), K_\alpha),$$

where $K_\alpha = K/\alpha$, and has a probability of error that goes to 0 as $K \to \infty$.

The proof of the theorem is provided in Section IV. Note that in the theorem, $\alpha$ and $\gamma$ are parameters control the trade-off between the storage cost and the probability of error of the scheme (see Section IV).
and the Appendix). The main technical change as compared with [26] in the above scheme is that the result of [26] is for the random erasure model, and we need to adapt their methods to our setting where the server failures are adversarial. It is also worth noting that the theorem can be relaxed by choosing \( \alpha = \frac{K}{\log p K} \), such that the storage cost of Theorem 1 can be expressed as \( \frac{K}{c} + (\nu + 1)\delta_K K\log (\log^2 K) + o(\log^2 K) \).

2) Reed-Solomon Update-Efficient MVCs: In Reed-Solomon update-efficient MVCs, we simply use Reed-Solomon code of length \( n \) and dimension \( c \), over a field of size \( n_p \), where \( n_p \) is the smallest prime power that is greater than \( n \). That is, we split any message of \( K \) bits into \( \frac{K}{c\log n_p} \) blocks of \( c \log n_p \) bits each, and each block is encoded using this \((n, c)\) Reed-Solomon code. The \( i \)-th server then stores the \( i \)-th codeword symbol of the Reed-Solomon code for each of the \( \frac{K}{c\log n_p} \) blocks. Therefore, for a subsequent version, where at most \( \delta_K K \) bits change in the message, at most \( \delta_K K \) codeword symbols change per server. We can then store the changed codeword symbols and their locations. Therefore, the worst-case storage cost of this scheme is \( \frac{K}{c} + (\nu - 1)\delta_K K \log \left( \frac{K n_p}{c \log n_p} \right) = \frac{K}{c} + (\nu - 1)\delta_K K (\log K + o(\log K)) \). Observe here that the storage cost is asymptotically superior to KG Update-efficient MVC. We state our result formally below.

**Theorem 2.** [Reed-Solomon Update-Efficient MVC] There exists a 0-error \((n, c, \nu, 2^K, q, \delta_K)\) update-efficient multi-version code whose worst-case storage is given by

\[
\frac{K}{c} + (\nu - 1)\min(\delta_K K \log \left( \frac{K n_p}{c \log n_p} \right), K/c),
\]

where \( n_p > n \) is the smallest prime power that is greater than \( n \).

Although our discussion here provides sufficient hints for this rather elementary scheme and the proof of the above theorem, for the sake of completeness, we formally provide details of our scheme and a proof in Section IV-B.

It is worth noting that the update-efficient MVCs schemes do not require a priori knowledge of the parameter \( \delta_K \).

**Discussion:** For the regime of \( \delta_K = o(1/\log K) \), there remains the question of optimality of the Reed-Solomon update-efficient MVCs. The Reed-Solomon update-efficient MVCs whose storage cost is \( \frac{K}{c} + (\nu - 1)\delta_K K (\log K + o(\log K)) \) may be interpreted as follows. For the \( i \)-th server, for the first version it receives, store the \( i \)-th symbol of an \((n, c)\) MDS erasure code to store \( \frac{K}{c} \) bits; for each subsequent version, remaining simply store, in its entirety, a representation of the difference with respect to the first version. The fact that the difference with respect to the first version is (effectively) stored in its entirety motivates the following question, which is essentially a variant of question Q2 posed in Section III-A.

**Can we use erasure coding based ideas for the subsequent versions as well, to obtain a storage cost of** \( \frac{K}{c} + \frac{\nu - 1}{c} \delta_K K (\log K + o(\log K)) \)?

The answer to the above question Q2 is particularly non-trivial because of the nature of the multi-version coding problem. For instance, one set of servers may receive versions \( W_1 \) and \( W_3 \). A second set of servers may not receive \( W_1 \), but instead receive \( W_2 \) and \( W_3 \). The first set of servers has to encode the difference between \( W_1 \) and \( W_3 \), whereas the second set of servers have to encode the difference between \( W_2 \) and \( W_3 \) in a manner that \( W_3 \) is decodable from all the servers. The development or modification of update-efficient code constructions to satisfy this decoding constraint, and still obtain a \( 1/c \) “erasure-coding gain” factor for the difference is non-trivial in such scenarios.

Next, we summarize the results of code constructions that answers the question posed above positively. In particular, we describe code constructions whose storage cost is approximately \( \frac{K}{c} + \frac{\nu - 1}{c} \log Vol(\delta_K K, K) \). Unlike the update-efficient schemes, a priori knowledge of \( \delta_K \) is required in the code construction.

**C. Slepian-Wolf inspired Multi-version Codes**

In Section V, we develop MVCs that are inspired by Slepian-Wolf coding. In Section V-A, we develop a MVC that is based on random binning. The following theorem specifies the worst-case storage cost of this scheme.
Theorem 3. [Random Binning MVC] There exists an \( \epsilon \)-error multi-version code \((n, c, \nu, 2^K, q, \delta_K)\) whose worst-case storage cost is given by

\[
\frac{K}{c} + \frac{(\nu - 1) \log \text{Vol}(\delta_K K, K)}{c} + \frac{\nu(\nu - 1)/2 - \nu \log \epsilon}{c}.
\] (6)

Choosing \( \epsilon = \frac{1}{\log K} \), the above theorem implies that there exist a sequence of MVCs with vanishing error probability and storage cost of \( \frac{K}{c} + \frac{(\nu - 1) \log \text{Vol}(\delta_K K, K)}{c} + o(\log K) \).

For the case where \( \delta_K \) is a constant, our scheme specializes to the scheme presented in our earlier work [1] as given by the following corollary.

Corollary 3.1. For \( \delta_K = \delta \), where \( \delta < 1/2 \) is a constant, the worst-case storage cost is upper-bounded by

\[
\frac{K}{c} + \frac{(\nu - 1) K \text{H}(\delta)}{c} + \frac{\nu(\nu - 1)/2 - \nu \log \epsilon}{c}.
\] (7)

We note that this corollary follows from Remark [1].

The code construction of Theorem 3 is non-constructive. In Section V-B, we consider a variant of multi-version coding assuming that each server has a common baseline version \( W_1 \), that is, \( 1 \in S(i), \forall i \in \left[ n \right] \).

Under the assumption of a common baseline version, we provide linear MVC code constructions inspired by the ideas of [17]. The storage cost of our constructions is given by the following theorem.

Theorem 4. [Linear Binning MVC] In a distributed storage system with a common baseline version \( W_1 \), there exists a 0-error \((n, c, \nu, 2^K, q, \delta_K)\) multi-version code whose worst-case storage cost is given by

\[
\frac{K}{c} + \sum_{i=2}^{\nu} \frac{K(1 - R_{\text{opt}}(K, 2(i - 1)\delta_K K + 1))}{c},
\] (8)

where \( R_{\text{opt}}(N, d) \) is the highest possible rate of a linear binary code with length \( N \) and minimum distance \( d \).

Using the special case of BCH codes, we obtain the following corollary.

Corollary 4.1 (BCH Binning MVC). In a distributed storage system with a common baseline version \( W_1 \), there exists a 0-error \((n, c, \nu, 2^K, q, \delta_K)\) multi-version code whose worst-case storage cost is at most

\[
\frac{K}{c} + \frac{\nu(\nu - 1)/2 \delta_K K \log(K + 1)}{c}.
\] (9)

We notice that this scheme may have a better storage cost as compared with the scheme of Theorem 2 depending on the values of \( \nu \) and \( c \).

The result of Theorem 4 leads to at least two interesting open questions: (i) can we design algorithms in distributed systems that ensure that all servers have a common baseline version?, and (ii) can we design constructive MVC schemes corresponding to the existence results of Theorem 3, where we do not assume a common baseline version at all servers? The answers to both questions are outside the scope of our work, but we discuss some ideas here.

The first question relies on ideas in the field of distributed algorithms, and the idea of checkpointing [36] is used in several distributed systems to take a snapshot of the current state of the system at each server node. This idea may help develop algorithms, where we may be able to achieve a common baseline version at all servers. Furthermore, certain algorithms such as [37] already do ensure, via a finalize label, a sufficient number of servers in the system agree that a particular message version has propagated to
a sufficient number of servers. It is an open question whether our code constructions can be adapted to such algorithms.

Although there are linear code constructions for the Slepian-Wolf setting [17]–[19], the second question is interesting, at least in part, because our setting is slightly more complicated than the usual Slepian-Wolf setting. Specifically, in our setting, the decoder desires to decode only the latest common version, or a later version, whereas in the usual Slepian-Wolf setting, the decoder desires to decode all the messages. Furthermore, our setting has a concept of states, similar in spirit to compound wireless channels - the state of the system can change and the version is to be decoded, and the possible achievable rates, can depend on the state of the system. It is not immediately clear whether the ideas of [17]–[19] can be inherited or adapted to our setting.

D. Lower Bound on The Storage Cost

Lower Bound on the storage cost: In Section VI, we extend the storage cost lower bound derived in [7] for the case where we have correlated versions. We state our converse results in the following theorem and the subsequent corollary.

**Theorem 5.** [Storage Cost Lower Bound] An $\epsilon$-error $(n, c, \nu, 2^K, q, \delta_K)$ multi-version code with correlated versions such that $W_1 \rightarrow W_2 \rightarrow \ldots \rightarrow W_\nu$ form a Markov chain. $W_m \in [2^K]$ and given $W_m$, $W_{m+1}$ must satisfy

$$\log q \geq \frac{K + (\nu - 1) \log \text{Vol}(\delta_K K, K)}{c + \nu - 1} + \frac{\log(1 - e^{2^m_n}) - \log \frac{(c+\nu-1)^\nu}{\nu!}}{c + \nu - 1}. \quad (10)$$

**Corollary 5.1.** For $\delta_K = \delta$, where $\delta < 1/2$ is a constant, we have

$$\log q \geq \frac{K}{c + \nu - 1} + \frac{\nu - 1}{c + \nu - 1} K H(\delta) + o(K). \quad (11)$$

We note that the above corollary follows from Remark 1.

The above corollary implies that the storage cost characterized in Corollary 3.1 is approximately optimal if $\nu < c$ for $\delta_K = \delta$, that is, $\delta_K$ is a constant. Specifically, since $1/(c + \nu - 1) \geq 0.5/c$ if $\nu < c$, the storage cost of the converse is at least half of $\frac{K}{c} + \frac{\nu - 1}{c} K H(\delta) + o(K)$, which is the achievable storage cost in Corollary 3.1.

IV. UPDATE-EFFICIENT MULTI-VERSION CODES

In this section, we develop simple multi-version coding schemes that exploit the correlation between the different versions and have smaller storage cost as compared with [7]. In these schemes, the servers do not know the correlation degree $\delta_K$ in advance. We begin by recalling the definition of the update efficiency of a code from [26].

**Definition 3 (Update efficiency).** For a code $C$ of length $N$ and dimension $K$ with encoder $C : \mathbb{F}^K \rightarrow \mathbb{R}^N$, the update efficiency of the code is the maximum number of codeword symbols that must be updated when a single message symbol is changed. Formally, the update efficiency is expressed as follows

$$t = \max_{W, W' \in \mathbb{F}^K} \max_{d_H(W, W') = 1} d_H(C(W), C(W')). \quad (12)$$

We observe that the update efficiency of a linear code is the maximum row weight of the generator matrix of this code, hence it is at least the minimum distance of the code.

**Definition 4 (Update efficiency of a server).** Suppose that $C^{(i)} : \mathbb{F}^K \rightarrow \mathbb{R}^{N/n}$ denotes the $i$-th co-ordinate of the output of $C$ stored by the $i$-th server. The update efficiency of the $i$-th server is the maximum
number of codeword symbols that must be updated in this server when a single message symbol is changed. Formally, the update efficiency of the \( i \)-th server is given by

\[
\ell(i) = \max_{W, W' \in \mathbb{K}} d_H(C(i)(W), C(i)(W')).
\]  

(13)

Suppose that \( G = (G^{(1)}, G^{(2)}, \cdots, G^{(n)}) \) is the generator matrix of a linear code \( C \), where \( G^{(i)} \) is of dimension \( K \times N/n \) and corresponds to the \( i \)-th server. The update efficiency of the \( i \)-th server is the maximum row weight of \( G^{(i)} \).

**Definition 5** (Maximum update efficiency per server). The maximum update efficiency per server is the maximum number of codeword symbols that must be updated in any server when a single message symbol is changed. Formally, the maximum update efficiency per server is given by

\[
t_s = \max_{i \in [n]} \ell(i).
\]  

(14)

An \((N, K)\) code \( C \) is referred to as an update-efficient code if it has an update efficiency of \( o(N) \).

References [26] and [27] have constructed update-efficient codes with update efficiency \( O(\log K) \) that can tolerate random erasures, achieve rates arbitrarily close to the capacity over \( \text{BEC}(p) \) and have arbitrary small probability of error.

A. KG Update-efficient MVC

The KG code construction in [26] has been proposed for the random failures model, where each server fails with probability \( p \) independently from the other servers. In this Section, we provide the proof of the simple scheme of Theorem 1, where we show that the KG code also can tolerate any \( n - c \) server failures with high probability.

**Proof of Theorem 1** We use the \((nK_{\alpha}/c, K)\) binary code \( C \) given by the following construction.

**Construction 1** (KG Update-Efficient MVC). Consider the \((nK_{\alpha}/c, K)\) code \( C_r \) characterized by the generator matrix \( G = [g_{ij}] \) whose entries are chosen randomly and independently of each other based on the distribution

\[
\Pr[g_{ij} = 1] = \frac{\ln K_{\alpha} + x}{K_{\alpha}} \quad i \in [K], j \in [nK_{\alpha}/c],
\]  

(15)

where \( K_{\alpha} = K/\alpha, x = a \log K_{\alpha} \) for some constants \( a > 0 \) and \( \alpha < 1 \). There exists a deterministic code with a generator matrix \( \overline{G} = (\overline{G}^{(1)}, \overline{G}^{(2)}, \cdots, \overline{G}^{(n)}) \) in this ensemble that can tolerate any \( n - c \) server failures with probability that goes to 1 as \( K \to \infty \) and has a maximum update efficiency per server that is at most \( \gamma \ln K_{\alpha}/c \), where \( \gamma > 1 \). Suppose that the \( i \)-th server receives the set of versions \( S = \{s_1, s_2, \ldots, s_{|S|}\} \).

For \( W_{s_1} \), the server stores \( W_{s_1}^T \overline{G}^{(i)} \). For \( W_{s_m} \), where \( m > 1 \), the server may store the coordinates that have been updated or store \( W_{s_m}^T \overline{G}^{(i)} \). Therefore, it stores \( \min(|s_m - s_{m-1}|, K \ell(i) \log(K_{\alpha}/c), K_{\alpha}/c) \) bits.

We show that this construction can tolerate any \( n - c \) failures with probability that goes to 1 as \( K \to \infty \) in the Appendix.

The worst-case storage cost corresponds to the case where a server receives all versions. The storage cost per server in this case is at most

\[
\frac{K_{\alpha}}{c} + \sum_{j=1}^{N-1} \min(d_H(W_{j+1}, W_j) t_s \log(K_{\alpha}/c), K_{\alpha}/c),
\]
Hence, the worst-case storage cost is given by
\[ \frac{K_\alpha}{c} + \frac{\nu - 1}{c} \min(\delta_K K \gamma (\ln K_\alpha) \log(K_\alpha/c), K_\alpha). \]

**Remark 3.** In the achievable scheme given by Theorem 1 the server stores the index of each bit that changes independently. Since the number of bits that change from a version to the next version is at most \( \delta_K K t_s \), we have \( \sum_{j=0}^{\delta_K K t_s} \binom{K_\alpha/c}{j} \) possibilities of the indices of these bits and hence the following storage cost is also achievable
\[ \frac{K_\alpha}{c} + (\nu - 1) \log \sum_{j=0}^{\delta_K K t_s} \binom{K_\alpha/c}{j}, \quad (16) \]
where \( t_s < \frac{\gamma \ln K_\alpha}{c} \). However, in this scheme the servers need to know \( \delta_K \) a priori.

**B. Reed-Solomon Update-efficient MVC**

The previous update-efficient multi-version coding scheme has an update efficiency of \( O(\log K) \) which grows with the file size \( K \). Theorem 2 presents a practical update-efficient multi-version coding scheme that is based on Reed-Solomon code and has an update efficiency of \( n \). Hence, the update efficiency grows with the number of servers, but not with the size of the file. Moreover, the maximum update-efficiency per server is 1. We prove the theorem next.

**Proof of Theorem 2.** We start by describing the code construction.

**Construction 2 (Reed-Solomon Update-Efficient MVC).** Suppose that the \( i \)-th server receives the versions \( S = \{s_1, s_2, \ldots, s_{|S|}\} \). We divide a version \( W_s, j \in [S] \), into \( \frac{K}{c \log n_p} \) blocks, each of length \( c \log n_p \).

In each block, we represent every consecutive string of \( \log n_p \) bits by a symbol in \( F_{n_p} \). We denote the representation of \( W_s \) over \( F_{n_p} \) by \( W_{s_j} \). We encode each block by a \((n,c)\) Reed-Solomon code with a generator matrix \( \tilde{G} \) which is given by
\[ \tilde{G} = \begin{pmatrix} 1 & 1 & \cdots & 1 \\ \lambda_1 & \lambda_2 & \cdots & \lambda_n \\ \lambda_1^2 & \lambda_2^2 & \cdots & \lambda_n^2 \\ \vdots & \vdots & \cdots & \vdots \\ \lambda_1^{c-1} & \lambda_2^{c-1} & \cdots & \lambda_n^{c-1} \end{pmatrix}, \quad (17) \]
where \( \Lambda = \{\lambda_1, \lambda_2, \cdots, \lambda_n\} \subset F_{n_p} \) is a set of distinct elements.

For \( W_{s_1} \), the \( i \)-th server stores \( W_{s_1}^T G^{(i)} \), where \( G^{(i)} \) is given by
\[ G^{(i)} = \begin{pmatrix} \tilde{G} e_i & 0 & \cdots & 0 \\ 0 & \tilde{G} e_i & \cdots & 0 \\ \vdots & \vdots & \cdots & \vdots \\ 0 & 0 & \cdots & \tilde{G} e_i \end{pmatrix}. \quad (18) \]

For \( W_{s_m} \), where \( m > 1 \), the server may only store the updated symbols from the old version \( W_{s_{m-1}} \) or store all symbols, thus it stores at most \( \min(|s_m - s_{m-1}| \delta_K K (\log(c \log n_p) + \log n_p), K/c) \) bits of \( W_{s_m} \).
The worst-case storage cost corresponds to the case where a server receives all versions. Therefore, the storage cost per server is at most
\[
\frac{K}{c} + \sum_{i=1}^{\nu-1} \min(d_H(W_{i+1}, W_i), \log \left( \frac{Kn_p}{c \log n_p} \right), K/c).
\]
Hence, the worst-case storage cost is given by
\[
\frac{K}{c} + (\nu - 1) \min(\delta K \log \left( \frac{Kn_p}{c \log n_p} \right), K/c).
\]

V. SLEPIAN-WOLF INSPIRED MULTI-VERSION CODES

In this section, we introduce a random binning based argument showing the existence of a multi-version coding scheme satisfying the result of Theorem 3. Later on, in Section V-B, we construct a linear coding construction, showing Theorem 4.

A. Random Binning Multi-version Code.

Recall that Slepian-Wolf coding [15], [38] is a distributed data compression technique for correlated sources that are drawn in independent and identical manner according to a given distribution. In the Slepian-Wolf setting, the decoder is interested in decoding the data of all sources. In the multi-version coding problem, the decoder is interested in decoding the latest common version, or a later version, among a set of \(c\) servers. Inspired by the random binning proof for the Slepian-Wolf problem [33], [39], we build multi-version coding constructions where older versions earlier provide side information to the decoder to recover the latest common version. Similarly, in our constructions, a version that is a latest common version in some state may act as a side information to the decoder in a different state.

We note that our model slightly differs from the standard Slepian-Wolf setting. First, in our setting, we do not aspire to decode all the versions, but we only want to ensure that the latest common version is decodable for every state. Second, unlike the standard Slepian-Wolf setting, we are interested in a broader correlation model, where the bits of subsequent versions are not drawn in an independent and identical manner given the previous versions. For instance, given \(W_1\), in the standard Slepian-wolf setting, \(W_2\) is concentrated in a spherical shell of radius \(\delta K ± o(K)\) for some constant \(\delta\). However, our setting is more general. Despite these differences, the idea of random binning naturally leads to a proof in our paper. An interesting aspect of our proof is the argument that a point with the rate prescribed in Theorem 3 indeed exists in the feasible rate region for the given error probability.

The lossless source coding problem with a helper [40], [41] may seem to be related to our approach, since the side information of older versions that is used to decode the latest common version in our approach may be interpreted as helpers. In an optimal strategy for the helper setting, the helper side information is encoded via a joint typicality encoding scheme, whereas the random binning is used for the message. However, note that in the multi-version coding setting, the versions that may be a side information for one state may required to be decoded in another state. For this reason, a random binning scheme for all versions - even those that may be helpers in certain states - leads to schemes with a reasonable storage cost.

Before proving Theorem 3 we introduce the following useful definitions.
**Definition 6** ($\delta_K$-possible Set of Tuples). The set $A_{\delta_K}$ of $\delta_K$-possible set of tuples $\langle w_{u_1}, w_{u_2}, \ldots, w_{u_L} \rangle$ is defined as follows

\[
A_{\delta_K}(W_{u_1}, W_{u_2}, \ldots, W_{u_L}) = \\
\{ (w_{u_1}, w_{u_2}, \ldots, w_{u_L}) : w_{u_i} \in [2^K], \\
w_{u_2} \in B(w_{u_1}, \delta_K(u_2 - u_1)K), w_{u_3} \in B(w_{u_2}, \delta_K(u_3 - u_2)K), \\
\ldots, w_{u_L} \in B(w_{u_{L-1}}, \delta_K(u_{u_L} - u_{u_{L-1}})K) \},
\]

where $u_1 < u_2 < \cdots < u_L$.

We omit the dependency on the messages and simply write $A_{\delta_K}$, when it is clear from the context. Similarly, we can also define the set of possible tuples $w_{F_1}$ given a particular tuple $w_{F_2}$, $A_{\delta_K}(W_{F_1} | W_{F_2})$, where $F_1, F_2$ be two subsets of $\{ u_1, u_2, \ldots, u_L \}$.

We next provide a proof of Theorem 3.

**Proof of Theorem 3.** We first describe the random binning construction.

**Construction 3** (Random binning multi-version code).

- **Random code generation:** For a version $s_j$ the encoder assigns an index at random from $\{ 1, 2, \ldots, 2^\nu R_j \}$ uniformly and independently to each sequence of length $K$ bits. The set of sequences which have the same index form a bin.

- **Encoding:** The server stores the corresponding bin index to each version that it receives. The decoder is also aware of the mapping used in the binning scheme. Assume that the $i$-th server receives the set of ordered versions $S = \{ s_1, s_2, \ldots, s_{|S|} \} \subseteq [\nu]$, where $s_1 < s_2 < \cdots < s_{|S|}$. The encoding function of the $i$-th server is defined as follows

\[
\varphi^{(i)}_S = (\varphi^{(i)}_{s_1}, \varphi^{(i)}_{s_2}, \ldots, \varphi^{(i)}_{s_{|S|}}),
\]

where

\[
\varphi^{(i)}_{s_j} : [2^K] \rightarrow \{ 1, 2 \ldots, 2^\nu R_j \},
\]

for $j \in \{ 1, 2, \ldots, |S| \}$, where $R_{s_j}/c$ is the compression rate of version $s_j$. In particular, we choose the rates as follows

\[
KR_{s_1} = K + (s_1 - 1) \log Vol(\delta_K K, K) + (s_1 - 1) - \log \epsilon,
\]

\[
KR_{s_j} = (s_j - s_{j-1}) \log Vol(\delta_K K, K) + (s_j - 1) - \log \epsilon, \quad j \in \{ 2, 3, \ldots, |S| \}.
\]

- **Decoding:** For every set $S \in \mathcal{P}([\nu])^n$ and set $T = \{ t_1, t_2, \ldots, t_{|c|} \} \subseteq [n]$ of $c$ servers, the decoder employs the possible set decoding strategy that we will explain next to recover the latest common version among those servers. Suppose that a version $s_j, j \in S$, is received by a set of servers $\{ i_1, i_2, \ldots, i_{|r|} \} \subseteq [n]$, then the bin index corresponding to this version is given by

\[
\varphi_{s_j} = (\varphi^{(i_1)}_{s_j}, \varphi^{(i_2)}_{s_j}, \ldots, \varphi^{(i_{|r|})}_{s_j}).
\]

Assume that $W_{u_L}$ is the latest version among these servers and that the versions $W_{u_1}, W_{u_2}, \ldots, W_{u_{L-1}}$ have been received by some servers out of those $c$ servers before receiving $W_{u_L}$. We define this set of versions formally as follows

\[
S_T = \{ u_1, u_2, \ldots, u_L \} = \left( \bigcup_{t \in T} S(t) \right) \setminus \{ u_L + 1, u_L + 2, \ldots, \nu \},
\]

where $u_1 < u_2 < \cdots < u_L$. Given the bin indices $(b_{u_1}, b_{u_2}, \ldots, b_{u_L})$, the decoder first finds all tuples $\langle w_{u_1}, w_{u_2}, \ldots, w_{u_L} \rangle$ such that $(\varphi_{u_1}(w_{u_1}) = b_{u_1}, \varphi_{u_2}(w_{u_2}) = b_{u_2}, \ldots, \varphi_{u_L}(w_{u_L}) = b_{u_L})$
and \((w_{u_1}, w_{u_2}, \ldots, w_{u_L}) \in A_{\delta_K}\). If all of these tuples have the same latest common version \(w_{u_L}\), the decoder declares \(\hat{w}_{u_L} = w_{u_L}\). Otherwise, the decoder declares an error.

Denoting \(E\) as the event that there is an error, we can write

\[
E = \{ \exists (w'_{u_1}, w'_{u_2}, \ldots, w'_{u_L}) \in A_{\delta_K} : w'_{u_1} \neq w_{u_1} \text{ and } \varphi_u(w'_u) = \varphi_u(w_u) \quad \forall u \in S_T \}.
\]

The error event in decoding can be equivalently expressed as follows:

\[
E = \bigcup_{I \subseteq S_T : \exists u_L \in I} E_I,
\]

where

\[
E_I = \{ \exists w'_u \neq w_u \quad \forall u \in I : \varphi_u(w'_u) = \varphi_u(w_u) \quad \forall u \in I
\]

and \((w'_{u_L}, w_{S_T \setminus I}) \in A_{\delta_K}\),

for \(I \subseteq S_T\). By the union bound, the probability of error in decoding the latest common version among these \(c\) servers is upper-bounded as follows

\[
P_e = P(E) = P\left( \bigcup_{I \subseteq S_T : \exists u_L \in I} E_I \right) \leq \sum_{I \subseteq S_T : \exists u_L \in I} P(E_I),
\]

and we require that \(P_e < \epsilon\). Therefore, we require the following for every \(I \subseteq S_T\) such that \(u_L \in I\)

\[
P(E_I) < \epsilon 2^{-(L-1)},
\]

We now proceed in a case by case manner. We first consider the case where \(u_{L-1} \notin I\), later we will consider the case where \(u_{L-1} \in I\). For the case where \(u_{L-1} \notin I\), we have

\[
E_I \subset \hat{E}_{u_{L-1}} := \{ \exists w'_{u_L} \neq w_{u_L} : \varphi_{u_L}(w'_{u_L}) = \varphi_{u_L}(w_{u_L})
\]

and \((w_{u_{L-1}}, w'_{u_L}) \in A_{\delta_K}\).

Consequently, we have

\[
P(E_I) < P(\hat{E}_{u_{L-1}}),
\]

and we can upper-bound \(P(\hat{E}_{u_{L-1}})\) as follows

\[
P(\hat{E}_{u_{L-1}})
\]

and we require that \(P_e < \epsilon\). Therefore, we require the following for every \(I \subseteq S_T\) such that \(u_L \in I\)

\[
P(E_I) < \epsilon 2^{-(L-1)},
\]

We now proceed in a case by case manner. We first consider the case where \(u_{L-1} \notin I\), later we will consider the case where \(u_{L-1} \in I\). For the case where \(u_{L-1} \notin I\), we have

\[
E_I \subset \hat{E}_{u_{L-1}} := \{ \exists w'_{u_L} \neq w_{u_L} : \varphi_{u_L}(w'_{u_L}) = \varphi_{u_L}(w_{u_L})
\]

and \((w_{u_{L-1}}, w'_{u_L}) \in A_{\delta_K}\).

Consequently, we have

\[
P(E_I) < P(\hat{E}_{u_{L-1}}),
\]

and we can upper-bound \(P(\hat{E}_{u_{L-1}})\) as follows

\[
P(\hat{E}_{u_{L-1}})
\]

and we require that \(P_e < \epsilon\). Therefore, we require the following for every \(I \subseteq S_T\) such that \(u_L \in I\)

\[
P(E_I) < \epsilon 2^{-(L-1)},
\]

We now proceed in a case by case manner. We first consider the case where \(u_{L-1} \notin I\), later we will consider the case where \(u_{L-1} \in I\). For the case where \(u_{L-1} \notin I\), we have

\[
E_I \subset \hat{E}_{u_{L-1}} := \{ \exists w'_{u_L} \neq w_{u_L} : \varphi_{u_L}(w'_{u_L}) = \varphi_{u_L}(w_{u_L})
\]

and \((w_{u_{L-1}}, w'_{u_L}) \in A_{\delta_K}\).

Consequently, we have

\[
P(E_I) < P(\hat{E}_{u_{L-1}}),
\]

and we can upper-bound \(P(\hat{E}_{u_{L-1}})\) as follows

\[
P(\hat{E}_{u_{L-1}})
\]

and we require that \(P_e < \epsilon\). Therefore, we require the following for every \(I \subseteq S_T\) such that \(u_L \in I\)

\[
P(E_I) < \epsilon 2^{-(L-1)},
\]

We now proceed in a case by case manner. We first consider the case where \(u_{L-1} \notin I\), later we will consider the case where \(u_{L-1} \in I\). For the case where \(u_{L-1} \notin I\), we have

\[
E_I \subset \hat{E}_{u_{L-1}} := \{ \exists w'_{u_L} \neq w_{u_L} : \varphi_{u_L}(w'_{u_L}) = \varphi_{u_L}(w_{u_L})
\]

and \((w_{u_{L-1}}, w'_{u_L}) \in A_{\delta_K}\).

Consequently, we have

\[
P(E_I) < P(\hat{E}_{u_{L-1}}),
\]

and we can upper-bound \(P(\hat{E}_{u_{L-1}})\) as follows

\[
P(\hat{E}_{u_{L-1}})
\]

and we require that \(P_e < \epsilon\). Therefore, we require the following for every \(I \subseteq S_T\) such that \(u_L \in I\)

\[
P(E_I) < \epsilon 2^{-(L-1)},
\]
Choosing $R_{u_L}$ to satisfy 

$$KR_{u_L} \geq \log Vol((u_{L-1} - u_L)\delta_{K,K}) + (L - 1) - \log \epsilon$$

ensures that $P(E_{T}) < \epsilon 2^{-(L-1)}$.

Now, we consider the case where $u_{L-1} \in \mathcal{I}$. In this case, we consider the following two cases. First, we consider the case where $u_{L-2} \notin \mathcal{I}$, later will consider the case where $u_{L-2} \notin \mathcal{I}$. For the case where $u_{L-2} \notin \mathcal{I}$, we have

$$E_{T} \subseteq \tilde{E}_{u_{L-2}} := \{ \exists w'_{u_{L-1}} \neq w_{u_{L-1}}, w'_{u_L} \neq w_{u_L} : \varphi_{u_{L-1}}(w'_{u_{L-1}}) = \varphi_{u_{L-1}}(w_{u_{L-1}}), \varphi_{u_{L}}(w'_{u_L}) = \varphi_{u_{L}}(w_{u_{L}})$$

and $(w_{u_{L-2}}, w'_{u_{L-1}}, w'_{u_L}) \in A_{\delta_K}$.

Therefore, we have

$$P(E_{T}) < P(\tilde{E}_{u_{L-2}}),$$

and we can upper-bound $P(\tilde{E}_{u_{L-2}})$ as follows

$$P(\tilde{E}_{u_{L-2}}) \leq \sum_{(w_{u_{L-2}}, w_{u_{L-1}}, w_{u_L})} p(w_{u_{L-2}}, w_{u_{L-1}}, w_{u_L}) \sum_{w'_{u_{L-1}} \neq w_{u_{L-1}}, w'_{u_L} \neq w_{u_L}} P(\varphi(w'_{u_{L-1}}) = \varphi(w_{u_{L-1}}), \varphi(w'_{u_L}) = \varphi(w_{u_L}))$$

$$\leq \sum_{(w_{u_{L-2}}, w_{u_{L-1}}, w_{u_L})} p(w_{u_{L-2}}, w_{u_{L-1}}, w_{u_L}) 2^{-K(R_{u_{L-1}} + R_{u_L})}$$

$$\leq 2^{\log Vol((u_{L-1} - u_L)\delta_{K,K}) + \log Vol((u_{L-1} - u_{L-2})\delta_{K,K})}.$$
We notice that
\[ \log Vol(m\delta K, K) \leq m \log Vol(\delta K, K), \forall m \in \mathbb{Z}^+. \] (33)

Therefore, it suffices if the rates satisfy
\[
K \sum_{j=1}^{L} R_{u_j} \geq \sum_{j=1}^{L} (u_j - u_{j-1}) \log Vol(\delta K, K) + (L - 1) - \log \epsilon,
\]
\[ \forall i \in \{2, 3, \cdots, L\}. \]
\[
K \sum_{j=1}^{L} R_{u_j} \geq K + \sum_{j=2}^{L} (u_j - u_{j-1}) \log Vol(\delta K, K) + (L - 1) - \log \epsilon. \] (34)

The rates chosen in (21), (22) satisfy the above inequalities, therefore, our construction has a probability of error bounded by \( \epsilon \). The worst-case storage cost is when a server receives all versions. Therefore, the worst-case storage cost is given by
\[ \frac{K - \log \epsilon}{c} + \frac{(\nu - 1)(\log Vol(\delta K, K) - \log \epsilon + \nu/2)}{c}. \]

**Remark 4.** From a technical standpoint, the proof of the theorem above uses ideas that resemble simultaneous non-unique decoding [42], which is previously used in the several multi-user scenarios including the broadcast and interference channels, to decode the latest common version. In particular, to decode \( W_{u_1} \), the decoder picks the unique \( w_{u_1} \) such that \( (w_{u_1}, w_{u_2}, \ldots, w_{u_L}) \) is a possible sequence for some \( w_{u_1}, w_{u_2}, \ldots, w_{u_{L-1}} \), which is consistent with the bin index. We use this strategy since unlike the Slepian-Wolf problem where all the messages are to be decoded, we are only required to decode one of the messages (the latest common version), using the older versions as side information. The discussion in [43], which examined the necessity of non-unique decoding, motivates the following question: Can we use the decoding ideas of Slepian-Wolf - where all the messages are decoded - however, for an appropriately chosen subset of messages to recover the same rates? In other words, if we take the union of the unique decoding rate regions over all possible subsets of \( \{W_1, W_2, \ldots, W_{u_L}\} \), does the rate allocation of (21), (22), lie in this region. Below we provide a counter-example to answer this question in the negative.

We consider the case where \( \delta_K = \delta \), \( c = 2 \) and \( \nu = 3 \). Consider the state where server 2 does not receive \( W_1 \). Then, the storage allocation of our scheme is given by Table II. We use \( KR_i \) to denote the total number of bits stored for message \( i \) in Table II.

| Version | Server 1 | Server 2 |
|---------|----------|----------|
| \( W_1 \) | \( K + o(K) \) | \( - \) |
| \( W_2 \) | \( KH(\delta) + o(K) \) | \( \frac{K + KH(\delta) + o(K)}{2} \) |
| \( W_3 \) | \( KH(\delta) + o(K) \) | \( \frac{K H(\delta) + o(K)}{2} \) |

**TABLE II: Storage Allocation of Example 1.**

Now we examine unique decoding based decoders that aim to recover \( W_3 \). It is clear that the decoder cannot recover the \( K \) bits of \( W_3 \) by without using side information, since the total number of bits of \( W_3 \) stored is only \( KH(\delta) + o(K) \). Now consider the case where a unique decoding based decoder uses
the subset \( \{W_1, W_3\} \). The rates \( R_1^{\text{unique}, W_1, W_3}, R_3^{\text{unique}, W_1, W_3} \) for a vanishing probability of error would be

\[
KR_3^{\text{unique}, W_1, W_3} \geq KH(\delta \ast \delta) + o(K),
\]

\[
K(R_1^{\text{unique}, W_1, W_3} + R_3^{\text{unique}, W_1, W_3}) \geq K + KH(\delta \ast \delta) + o(K),
\]

where \( \delta \ast \delta = 2\delta(1 - \delta) \). Note however that we store, in total, \( KR_3 = KH(\delta) + o(K) \) bits for \( W_3 \), which is fewer than \( KR_3^{\text{unique}, W_1, W_3} \) for all \( \delta \).

Now consider the case where a unique decoding based decoder uses \( \{W_2, W_3\} \) for decoding. In this case, the decoder requires

\[
KR_3^{\text{unique}, W_2, W_3} \geq KH(\delta) + o(K),
\]

\[
K(R_2^{\text{unique}, W_2, W_3} + R_3^{\text{unique}, W_2, W_3}) \geq K + KH(\delta) + o(K)
\]

We notice \( K(R_2 + R_3) = K/2 + 2KH(\delta) + o(K) < K(R_2^{\text{unique}, W_2, W_3} + R_3^{\text{unique}, W_2, W_3}) \), for \( \delta < H^{-1}(0.5) \).

Finally, consider the case where a unique decoding based decoder uses all three messages \( \{W_1, W_2, W_3\} \). In this case, the rate tuples \( (R_1^{\text{unique}, W_1, W_2, W_3}, R_3^{\text{unique}, W_1, W_2, W_3}) \) have to satisfy seven inequalities, including:

\[
K(R_1^{\text{unique}, W_1, W_2, W_3} + R_3^{\text{unique}, W_1, W_2, W_3}) \geq K + KH(\delta \ast \delta) + o(K)
\]

Clearly \( K(R_1 + R_3) < K(R_1^{\text{unique}, W_1, W_2, W_3} + R_3^{\text{unique}, W_1, W_2, W_3}) \). Thus, the union of the unique decoding rate regions for vanishing error probabilities, taken over all possible subsets of \( \{W_1, W_2, W_3\} \), does not include the rate tuple of Table [1].

**B. Linear Binning Multi-version Code.**

Next, we propose a multi-version code construction, where the binning is based on cosets of linear codes, for the case where all servers initially have a common baseline version \( W_1 \). The main idea of this scheme is encoding each version received with respect to the baseline version. We next provide the proof of Theorem 4.

**Proof of Theorem 4** We begin by illustrating the code construction.

**Construction 4** (Linear Binning Multi-version Code).

Suppose that the \( i \)-th server receives the set of versions \( S = \{1, s_2, \cdots, s_{|S|}\} \subseteq [\nu] \).

1) Encoding: We encode the baseline version \( \overline{W}_1 \) using the code \( C_u \) specified by Construction 2, hence the server stores \( \overline{W}_1^T G^{(i)} \). For any other version \( W_{s_m} \), where \( m \in \{2, 3, \cdots, s_{|S|}\} \), the encoding is as follows.

- The server first finds the syndrome of this version (or equivalently which coset it belongs to) using a \( (K, KR_{\text{opt}}(K, 2(s_m - 1)\delta_K K + 1), 2(s_m - 1)\delta_K K + 1) \) binary code \([44], [45]\) as follows

\[
\overline{f}_{s_m}^T = W_{s_m}^T H_{s_m}^T,
\]

where \( H_K(1 - R_{\text{opt}}(K, 2(s_m - 1)\delta_K K + 1)) \times K \) is the parity check matrix of the code and \( R_{\text{opt}}(N, d) \) is the highest rate of a binary code with length \( N \) and minimum distance \( d \).

- The syndrome \( \overline{f}_{s_m} \) is then encoded using \( C_u \) and the \( i \)-th server stores \( C_u^{(i)}(\overline{f}_{s_m}) \), which is equivalent to \( K(1 - R_{\text{opt}}(K, 2(s_m - 1)\delta_K K + 1))/c \) bits of this version.

Therefore, the multi-version code is given by \( \varphi_S^{(i)}(W_S) = (C_u^{(i)}(\overline{W}_1), C_u^{(i)}(\overline{f}_{s_2}), \cdots, C_u^{(i)}(\overline{f}_{s_{|S|}})) \).

2) Decoding: suppose that \( W_{u_L} \) is the latest common version among the \( c \) servers that the decoder are connecting to. The decoding is done as follows.
The decoder first finds the syndrome $r_{uL}$ of $W_{uL}$ or equivalently the index of the coset to which it belongs.

The decoder then selects the closest vector in this coset to $W_1$ and announces it to be the latest common version [17].

The worst-case storage cost is when the server receives all versions and is given by

$$K + \frac{\sum_{i=2}^{\nu} K(1 - R_{\text{opt}}(K, 2(i - 1)\delta_K K + 1))}{c}.$$  

Proof of Corollary 4.1: In this scheme, we use a $(K, KR_{\text{BCH}}(K, 2(s_m - 1)\delta_K K + 1), 2(s_m - 1)\delta_K K + 1)$ binary BCH code [44], [45] to find the syndromes, where $R_{\text{BCH}}(N, d)$ denotes the rate of a binary BCH code with length $N$ and minimum distance $d$. For $K = 2^m - 1$, where $m$ is an integer, the dimension of the code is bounded as follows [46]

$$KR_{\text{BCH}}(K, 2(s_m - 1)\delta_K K + 1) \geq K - (s_m - 1)\delta_K K \log(K + 1).$$  

In this case, the worst-case storage cost is at most

$$K + \frac{\sum_{i=2}^{\nu} K(1 - R_{\text{BCH}}(K, 2(i - 1)\delta_K K + 1))}{c} \leq K + \frac{\sum_{i=2}^{\nu} (i - 1)\delta_K K \log(K + 1)}{c} = \frac{K}{c} + \frac{\nu(\nu - 1)/2 \delta_K K \log(K + 1)}{c}.$$  

VI. LOWER BOUND ON THE STORAGE COST

In this section, we modify the lower bound derived in [7] for the case where we have correlated versions, and we require successful decoding with probability that is at least $1 - \epsilon$. In this section, we briefly summarize the idea of the proof for $\nu = 2$.

Proof of Theorem 5: Consider any $\epsilon$-error $(n, c, \nu = 2, 2^K, q)$ multi-version code, and consider the first $c \leq n$ servers $T = \lfloor c \rfloor$ for the converse. Suppose we have two versions $W_2 = (W_1, W_2)$. We partition the set of possible tuples $A_{\delta_K}$ as follows

$$A_{\delta_K} = A_{\delta_K,1} \cup A_{\delta_K,2},$$

where $A_{\delta_K,1}$ is the set of all tuples $(w_1, w_2) \in A_{\delta_K}$ for which we can decode the latest common version or a later version successfully for all $S \in \mathcal{P}(\nu)^n$. $A_{\delta_K,2}$ is the set of tuples where we cannot decode successfully at least for one state $S \in \mathcal{P}(\nu)^n$, which can be expressed as follows

$$A_{\delta_K,2} = \bigcup_{S \in \mathcal{P}(\nu)^n} A_{\delta_K}^{(S)}.$$  

where $A_{\delta_K}^{(S)}$ is the set of tuples for which we cannot decode successfully given a state particular $S \in \mathcal{P}(\nu)^n$. Consequently, we have

$$|A_{\delta_K,2}| \leq \sum_{S \in \mathcal{P}(\nu)^n} |A_{\delta_K}^{(S)}|.  \quad (39)$$
For any state $S \in \mathcal{P}(\nu)^n$, we require that $P_e(S, T) < \epsilon$. Since all tuples in the set $A_{\delta K}$ are equiprobable, we have

$$P_e(S, T) = \frac{|A_{\delta K,2}^{(S)}|}{|A_{\delta K}|},$$

consequently, we have

$$|A_{\delta K,1}| = |A_{\delta K}| - |A_{\delta K,2}^{(S)}|$$

$$\geq |A_{\delta K}| - \sum_{S \in \mathcal{P}([\nu])^n} |A_{\delta K,2}^{(S)}|$$

$$> |A_{\delta K}| - \sum_{S \in \mathcal{P}([\nu])^n} \epsilon |A_{\delta K}|$$

$$> |A_{\delta K}|(1 - \epsilon 2^\nu n).$$

(41)

Suppose that $(W_1, W_2) \in A_{\delta K,1}$. Because of the decoding requirements, if $W_1$ is available at all servers, then the decoder must be able to obtain $W_1$, and if $W_2$ is available at all servers, then the decoder must return $W_2$. Therefore, there must be two states $S_1, S_2 \in \mathcal{P}(\nu)^n$ such that

- $S_1$ and $S_2$ differ only in the state of one server indexed by $B \in [c]$.
- $W_1$ can be recovered from the first $c$ servers in state $S_1$ and $W_2$ can be recovered from the first $c$ servers in $S_2$.

Therefore both $W_1$ and $W_2$ are decodable from the $c$ codeword symbols of the first $c$ servers in state $S_1$, and the codeword symbol of the $B$-th server in state $S_2$. Thus, we require the following

$$c q^{c+1} \geq |A_{\delta K,1}|$$

$$> |A_{\delta K}|(1 - \epsilon 2^\nu n)$$

Because $W_1$ is uniformly distributed in the set of all $K$ length binary vectors and given $W_1$, $W_2$ is uniformly distributed in a Hamming ball of radius $\delta K K$ centered around $W_1$, then we have

$$|A_{\delta K}| = 2^K Vol(\delta K K, K).$$

In this case, the storage cost can be lower-bounded as follows

$$\log q \geq \frac{K + \log Vol(\delta K K, K)}{c + 1} + \frac{\log(1 - \epsilon 2^\nu n) - \log c}{c + 1}.$$  \hspace{1cm} (42)

The lower bound in Corollary 5.1 follows since

$$Vol(\delta K K, K) = \sum_{j=0}^{\delta K K} \binom{K}{j} \geq \binom{K}{\delta K K}$$

$$= \prod_{i=0}^{\delta K K - 1} \frac{K - i}{\delta K K - i}$$

$$\geq \left( \frac{1}{\delta K} \right)^{\delta K K}.$$
VII. Conclusion

In this paper, we have proposed multi-version codes to efficiently store correlated updates of data in a distributed asynchronous storage system. These constructions are based on update-efficient codes, random binning and linear codes that exploit the correlation between the different versions of the message and reduce the storage cost. An outcome of our results is that the correlation between versions can be used to reduce storage costs in distributed systems, even if there is no single server or client node who is aware of all data versions, even in applications where consistency is important. In addition, our converse result shows that these constructions are nearly-optimal in certain regimes. The development of coding schemes that do not require a baseline version is an open research question, which would require interesting generalizations of previous code constructions for the Slepian-Wolf problem [17], [18]. It is also of interest to explore the question of optimality of our binning based constructions for the regime where \( \delta_K K = o(K) \).

**Appendix**

**Proof of Construction**

Here, we prove that the multi-version code given by Construction 1 can tolerate any \( n - c \) failures with probability that goes to 1 as \( K \rightarrow \infty \).

**Proof.** We need to prove that \( K \times K_\alpha \) submatrix \( G_E \) of \( G \) corresponding to any \( c \) servers is a full rank matrix with high probability as \( K \rightarrow \infty \). The result follows from manipulations of the results of [26], [35].

We start by recalling the following definitions from [35].

**Definition 7** (Critical Set). The set \( C = \{ r_1, r_2, \ldots, r_m \} \) of row indices of an \( r \times l \) matrix \( A \) in \( \mathbb{F}_2 \) is a critical set provided that

\[
\mathbf{a}_{r_1} + \mathbf{a}_{r_2} + \cdots + \mathbf{a}_{r_m} = 0,
\]

where \( \mathbf{a}_i \) is the \( i \)-th row of \( A \).

**Definition 8** (Independent Critical Sets). The critical sets \( C_1, C_2, \ldots, C_s \) are independent if

\[
b_1 C_1 \triangle b_2 C_2 \triangle \cdots \triangle b_s C_s = \phi
\]

if and only if \( b_1 = b_2 = \cdots = b_s = 0 \), where \( \triangle \) denotes the symmetric difference and \( b_i, i = 1, 2, \ldots, s \) are binary variables.

Denote the rank of \( A \) by \( r(A) \), the maximum number of independent critical sets by \( s(A) \) and number of zero rows by \( s_0(A) \). Since there is one-to-one correspondence between the critical sets and the solution of the equation \( A^T x = 0 \), then \( r(A) + s(A) = r \).

**Lemma 6.**

\[
\Pr[s_0(G_E) > 0] \leq \alpha e^{-x},
\]

and for \( x = a \ln K_\alpha \), where \( a > 0 \), we have

\[
\Pr[s_0(G_E) > 0] \leq \frac{\alpha}{K_\alpha^a},
\]

thus, as \( K \rightarrow \infty \), \( \Pr[s_0(G_E) > 0] \rightarrow 0 \).

**Proof.** The probability that a specific row is a zero row is given by,

\[
\left(1 - \frac{\ln K_\alpha + x}{K_\alpha}\right)^{K_\alpha} \leq \frac{1}{K_\alpha} e^{-x},
\]
where the inequality follows since \(1 - x \leq e^{-x}\) for \(0 \leq x \leq 1\). By the union bound, we have

\[
\Pr[s_0(G_E) > 0] \leq e^{-x}.
\]

Thus, for \(x = a \ln K_\alpha\), we have

\[
\Pr[s_0(G_E) > 0] \leq \frac{\alpha}{K_\alpha}.
\]

The next lemma establishes the fact that, with high probability, the critical sets of the matrix \(G_E\) consist only of zero rows. This result implies that \(G_E\) is a full rank matrix with high probability.

**Lemma 7.** If \(K \to \infty\), then

\[
\Pr[s(G_E) \neq s_0(G_E)] \to 0.
\]  

**Proof.** Consider any set of \(l\) rows of \(G_E\), these rows constitute a critical set if each column of the submatrix induced by these rows contains even number of 1’s. This probability is given by

\[
\left(\frac{1}{2} \left(1 + \left(1 - 2 \frac{\ln K_\alpha + x}{K_\alpha}\right)^l\right)\right)^{K_\alpha}.
\]

Moreover, the probability that these \(l\) rows are zero rows is given by

\[
\left(1 - \frac{\ln K_\alpha + x}{K_\alpha}\right)^{K_\alpha l}.
\]

Denote the number of critical sets by \(N_C\). The mean number of critical sets that do not consist of zero rows is given by

\[
\mathbb{E}[N_C] = \sum_{l=0}^{K} \binom{K}{l} \left(\frac{1}{2} \left(1 + \left(1 - 2 \frac{\ln K_\alpha + x}{K_\alpha}\right)^l\right)\right)^{K_\alpha} - \sum_{l=0}^{K} \binom{K}{l} \left(1 - \frac{\ln K_\alpha + x}{K_\alpha}\right)^{K_\alpha l}
\]

\[
= \sum_{l=0}^{K} \binom{K}{l} \frac{1}{2^{K_\alpha l}} r_l^{K_\alpha} - \left(1 + \left(1 - \frac{\ln K_\alpha + x}{K_\alpha}\right)^{K_\alpha}\right)^{K}
\]

\[
= S(K) - \left(1 + \left(1 - \frac{\ln K_\alpha + x}{K_\alpha}\right)^{K_\alpha}\right)^{K},
\]

where \(S(K) = \sum_{l=0}^{K} \binom{K}{l} \frac{1}{2^{K_\alpha l}} r_l^{K_\alpha}\) and \(r_l = 1 + \left(1 - 2 \frac{\ln K_\alpha + x}{K_\alpha}\right)^l\). Moreover, as \(K \to \infty\), we have

\[
\left(1 + \left(1 - \frac{\ln K_\alpha + x}{K_\alpha}\right)^{K_\alpha}\right)^{K} = \left(1 + \frac{1}{K_\alpha} e^{-x}\right)^{K}
\]

\[
= e^{\alpha e^{-x}}.
\]
Let \( a = \left( 1 - 2 \frac{\ln K_{\alpha} + \epsilon}{K_{\alpha}} \right) \) and consider the summation \( S(K) \),

\[
S(K) = \sum_{l=0}^{K} \binom{K}{l} \frac{1}{2K_{\alpha}} l^{K_{\alpha}}
\]

\[= \sum_{l=0}^{K} \binom{K}{l} \frac{1}{2K_{\alpha}} (1 + a^{l})^{K_{\alpha}}
\]

\[= \sum_{l=0}^{K} \binom{K}{l} \frac{1}{2K_{\alpha}} \sum_{i=0}^{K_{\alpha}} \binom{K_{\alpha}}{i} a^{li}
\]

\[= \sum_{i=0}^{K_{\alpha}} \binom{K_{\alpha}}{i} (1 + a^{i})^{K_{\alpha}}
\]

\[= \sum_{i=0}^{K_{\alpha}} a_{i}
\]

where \( a_{i} = \left( \binom{K_{\alpha}}{i} \right) \frac{1}{2K_{\alpha}} (1 + a^{i})^{K_{\alpha}} \). Next, we divide \( S(K) \) into the following summations

\[
S_{1} = \sum_{0 \leq i \leq i_{1}} a_{i},
S_{2} = \sum_{i_{1} < i \leq i_{2}} a_{i},
S_{3} = \sum_{i_{2} < i \leq i_{3}} a_{i},
S_{4} = \sum_{i_{3} < i \leq i_{4}} a_{i},
S_{5} = \sum_{i_{4} < i \leq K_{\alpha}} a_{i},
\]

where \( i_{1} = \zeta K_{\alpha} \), \( i_{2} = \frac{K_{\alpha}}{2} (1 - \zeta) \), \( i_{3} = \frac{K_{\alpha}}{2} - \frac{1}{2} K_{\alpha}^{1/2 + 1/10} \) and \( i_{4} = \frac{K_{\alpha}}{2} + \frac{1}{2} K_{\alpha}^{1/2 + 1/10} \).

We next show that \( S_{4} \) is the major contributor to the summation \( S \) and it is upper-bounded by \( e^{\alpha e^{r}} \). That implies that \( E_{c} = 0 \). We start by \( S_{1}(K) \),

\[
S_{1}(K) = \sum_{i \leq \zeta K_{\alpha}} \binom{K_{\alpha}}{i} \frac{1}{2K_{\alpha}} r_{i}^{K_{\alpha}}
\]

\[\leq r_{0}^{K_{\alpha}} \frac{1}{2K_{\alpha}} \sum_{i \leq \zeta K_{\alpha}} \binom{K_{\alpha}}{i}
\]

\[= 2^{-K_{\alpha}(1-\alpha)} \sum_{i \leq \zeta K_{\alpha}} \binom{K_{\alpha}}{i}
\]

\[\leq 2^{-K_{\alpha}(1-\alpha) - H(\zeta)},
\]

where the first inequality follows since \( r_{i}^{K_{\alpha}} \) is a decreasing function of \( i \) and the second inequality follows for \( \zeta < 0.5 \). For sufficiently small \( \zeta \) such that \( \zeta K_{\alpha} \) is an integer, \( S_{1} \to 0 \) as \( K_{\alpha} \to \infty \).

Next, we consider \( S_{2} \)

\[
S_{2}(K) = \sum_{\zeta K_{\alpha} < i \leq K_{\alpha}(1-\zeta)} a_{i}.
\]
First, observe that \( a_i \) is a monotone increasing function of \( i \) where \( \zeta K_\alpha \leq i \leq K_\alpha (1 - \zeta)/2 \). That follows since

\[
\frac{a_{i+1}}{a_i} = \left( \frac{K_\alpha}{i} \right)^{r_{i+1}} \geq K_\alpha - i r_{i+1} \left( \frac{K_\alpha}{i} \right)^{r_{i+1}} = K_\alpha - i r_i \left( \frac{K_\alpha}{i} \right)^{r_i} \geq \frac{K_\alpha - K_\alpha (1 - \zeta)/2}{K_\alpha (1 - \zeta)/2 - 1} \left( \frac{1 + (1 - 2(\ln K_\alpha + x)/K_\alpha)^i + 1}{1 + (1 - 2(\ln K_\alpha + x)/K_\alpha)^i} \right)^K
\]

\[
= \frac{K_\alpha - K_\alpha (1 - \zeta)/2}{K_\alpha (1 - \zeta)/2 - 1} \left( 1 - \frac{(1 - 2(\ln K_\alpha + x)/K_\alpha)^i + 1 - 2(\ln K_\alpha + x)/K_\alpha)^i + 1}{1 + (1 - 2(\ln K_\alpha + x)/K_\alpha)^i} \right)^K \geq \frac{1 + \zeta}{1 - \zeta - \frac{2}{K_\alpha}} \left( 1 - 2 \frac{\ln K_\alpha + x}{K_\alpha} \left( 1 - 2 \frac{\ln K_\alpha + x}{K_\alpha} \right)^i \right)^K.
\]

For sufficiently large \( K_\alpha \), we have

\[
\frac{1 + \zeta}{1 - \zeta - \frac{2}{K_\alpha}} \geq (1 + \zeta)
\]

and for \( \zeta K_\alpha \leq i \leq K_\alpha (1 - \zeta)/2 \), we have

\[
\left( 1 - 2 \frac{\ln K_\alpha + x}{K_\alpha} \right)^i \leq e^{-2i(\ln K_\alpha + x)/K_\alpha} \leq K_\alpha^{-2i} e^{-2\zeta x}.
\]

Therefore, for sufficiently large \( K_\alpha \), we have

\[
\frac{a_{i+1}}{a_i} \geq (1 + \zeta) \left( 1 - 2 e^{-2\zeta x} \frac{\ln K_\alpha + x}{K_\alpha^{1+2\zeta}} \right)^K \geq 1.
\]

\[
S_2(K) = \sum_{i_1 < i \leq i_2} a_i, \leq a_{i_2} K_\alpha
\]

\[
= \left( K_\alpha \right)^{i_2} \frac{1}{2K_\alpha} \left( 1 + \left( 1 - 2 \frac{\ln K_\alpha + x}{K_\alpha} \right)^i \right)^K K_\alpha
\]

for \( x = a \ln K_\alpha \), we have

\[
\left( 1 + \left( 1 - 2 \frac{\ln K_\alpha + x}{K_\alpha} \right)^i \right)^K \leq \left( 1 + e^{-2(1-\zeta)(\ln K_\alpha + x)} \right)^K \leq e^{a K_\alpha^{\zeta-a(1-\zeta)}}.
\]

Moreover, we also have that [39, Ch. 11]

\[
\left( K_\alpha \right)^{i_2} \leq 2 K_\alpha H\left( \frac{1 - \zeta}{2} \right).
\]
Therefore, $S_2(K)$ can be upper-bounded as follows

$$S_2(K) \leq 2^{-K_\alpha(1-H_{\alpha}(\frac{1-\xi}{2}))} e^{\alpha K_\alpha^{\xi-a(1-\xi)}} K_\alpha,$$

thus we can choose $\zeta < \frac{a}{1+a}$, hence $\zeta - a(1 - \zeta) < 0$ which implies that $S_2 \to 0$ for sufficiently large $K$.

Next, we consider $S_3$

$$S_3(K) = \sum_{i_2 < i \leq i_3} \binom{K_\alpha}{i} \frac{1}{2K_\alpha} \left( 1 + \left( 1 - 2 \frac{\ln K_\alpha + x}{K_\alpha} \right)^{i_2} \right)^K \sum_{i_2 < i \leq i_3} \binom{K_\alpha}{i} \frac{1}{2K_\alpha} \leq e^{\alpha K_\alpha^{\xi-a(1-\xi)}} \sum_{i_2 < i \leq i_3} \binom{K_\alpha}{i} \frac{1}{2K_\alpha},$$

Let $X$ be a $B(K_\alpha, 1/2)$ random variable and $Z$ be the standard normal random variable. By the de Moivre-Laplace theorem [47, Ch. 4], for sufficiently large $K$, we have,

$$\sum_{i_2 < i \leq i_3} \binom{K_\alpha}{i} \frac{1}{2K_\alpha} \leq \sum_{i_2 < i \leq i_3} \binom{K_\alpha}{i} \frac{1}{2K_\alpha} = \Pr[X \leq i_3] = \Pr[Z \leq -K_\alpha^{1/10}] = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{-K_\alpha^{1/10}} e^{-u^2/2} du < \frac{1}{\sqrt{2\pi}} \frac{1}{K_\alpha^{1/10}} e^{-K_\alpha^{1/5}/2},$$

where the last inequality follows since [48]

$$\text{erfc}(x) = \frac{2}{\sqrt{\pi}} \int_x^{\infty} e^{-t^2} dt = \frac{1}{\sqrt{\pi}} \int_{-x}^{\infty} e^{-u^2/2} du < \frac{1}{x\sqrt{\pi}} e^{-x^2}.$$

Therefore $S_3$ is upper-bounded as follows

$$S_3(K) \leq e^{\alpha K_\alpha^{\xi-a(1-\xi)}} \frac{1}{\sqrt{2\pi}} \frac{1}{K_\alpha^{1/10}} e^{-K_\alpha^{1/5}/2},$$

thus if $\zeta < \frac{a}{1+a}$, $S_3 \to 0$ for sufficiently large $K$. 

Next, we consider $S_4(K)$. Let $i = \frac{K_\alpha}{2} + \frac{u}{2} K^{1/2}$, then we have

\[
S_4 = \sum_{i_3 < i_4} \binom{K_\alpha}{i} \frac{1}{2K_\alpha} r^K_i
\]

\[
= \sum_{|u| \leq K^{1/10}} \binom{K_\alpha}{i} \frac{1}{2K_\alpha} \left( 1 + \left( 1 - 2 \frac{\ln K_\alpha + x}{K_\alpha} \right)i \right)^K
\]

Moreover, for $i = K_\alpha/2$ we have

\[
\left( 1 - 2 \frac{\ln K_\alpha + x}{K_\alpha} \right)^{K_\alpha/2} \leq \frac{1}{K_\alpha} e^{-x},
\]

and hence

\[
r_{K_\alpha/2}^K \leq e^{\alpha e^{-x}}.
\]

Also, by de Moivre-Laplace theorem, we have

\[
\sum_{i: |u| \leq K^{1/10}} \binom{K_\alpha}{i} \frac{1}{2K_\alpha} \to 1.
\]

Finally, $S_5(K) \to 0$ since by deMoivre-Laplace theorem we have

\[
\sum_{i: u \geq K^{1/10}} \binom{K_\alpha}{i} \frac{1}{2K_\alpha} \to 0.
\]

Thus, $S(K) \leq e^{\alpha e^{-x}}$ and the mean number of critical sets that do not consist of zero rows $\mathbb{E}[N_C] = 0$. Therefore, $\Pr[N_C > 0] \to 0$.

From lemma 6 and lemma 7 as $K \to \infty$, we have

\[
\Pr[r(G_E) < K] = \Pr[s(G_E) > 0] \to 0.
\]

(46)

Hence, the code $C_r$ can tolerate any $n - c$ server failures with high probability as $K \to \infty$.

We next argue that there exists a generator matrix $G$ in this ensemble such that $r(G_E) = K$ and $w_H(g_l) = O(\ln K)$, $\forall l \in [K]$ with high probability, where $g_l$ be the $l$-th row of $G$. Let $G = (G^{(1)}, G^{(2)}, \cdots, G^{(n)})$, where $G^{(i)}$ is the generator matrix of the $i$-th server. Let $g_l$ be the $l$-th row of $G$. The expected row weight of $G$ is given by

\[
\mathbb{E}[w_H(g_l)] = \frac{n(a + 1) \ln K_\alpha}{c},
\]

for $l \in [K]$ and the expected row weight of $G^{(i)}$ where $i \in [n]$ is given by

\[
\mathbb{E}[w_H(g^{(i)}_l)] = \frac{(a + 1) \ln K_\alpha}{c},
\]

(47)

where $g^{(i)}_l$ be the $l$-th row of $G^{(i)}$. Moreover, we have

\[
\Pr[w_H(g^{(i)}_l) \geq \frac{(1 + \beta)(a + 1) \ln K_\alpha}{c}] \leq K_\alpha^{-\frac{\beta^2(a+1)}{3e}},
\]
where the last inequality follows from Chernoff bound for $0 < \beta < 1$. By the union bound, we have
\[
\Pr[\exists l \in [K] : w_H(g^{(i)}_l) \geq \frac{(1 + \beta)(a + 1) \ln K_\alpha}{c}] \leq \alpha K_\alpha^{1 - \frac{\beta^2(a + 1)}{3c}},
\]
where we choose $a$ such that $\frac{\beta^2(a + 1)}{3c} < 1$. Hence,
\[
\lim_{K \to \infty} \Pr[\exists l \in [K] : w_H(g^{(i)}_l) \geq \frac{(1 + \beta)(a + 1) \ln K_\alpha}{c}] = 0. \tag{48}
\]
Consequently, we have
\[
\lim_{K \to \infty} \Pr[w_H(g^{(i)}_l) < \frac{(1 + \beta)(a + 1) \ln K_\alpha}{c} \forall l \in [K],
\]
r($G_E$) = $K$, \tag{49}

hence, there exists a generator matrix $G$ in this ensemble such that
\[
w_H(g^{(i)}_l) < \frac{(1 + \beta)(a + 1) \ln K_\alpha}{c} \forall l \in [K],
\]
r($G_E$) = $K$, \tag{50}

where $G_E$ is a submatrix of $G$ corresponding to any $c$ servers. Hence, the update efficiency of the code is upper-bounded as follows
\[
t \leq \frac{n\gamma \ln K_\alpha}{c}, \tag{52}
\]
where $\gamma = (1 + \beta)(a + 1)$.

REFERENCES

[1] R. E. Ali and V. R. Cadambe, “Consistent distributed storage of correlated data updates via multi-version coding,” in *Information Theory Workshop (ITW), 2016, IEEE*, pp. 176–180.
[2] G. DeCandia, D. Hastorun, M. Jampangi, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, “Dynamo: Amazon's highly available key-value store,” in *SOSP*, vol. 7, 2007, pp. 205–220.
[3] E. Hewitt, *Cassandra: the definitive guide*. “O'Reilly Media, Inc.”, 2010.
[4] J. C. Anderson, J. Lehnardt, and N. Slater, *CouchDB: the definitive guide*. O’Reilly Media, Inc., 2010.
[5] N. A. Lynch, *Distributed Algorithms*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1996.
[6] L. Lamport, “How to make a multiprocessor computer that correctly executes multiprocess programs,” *IEEE Transactions on Computers*, vol. 100, no. 9, pp. 690–691, 1979.
[7] Z. Wang and V. R. Cadambe, “Multi-version coding—an information-theoretic perspective of consistent distributed storage,” *IEEE Transactions on Information Theory*, 2017.
[8] Y. Lin, B. Kemme, M. Patino-Martinez, and R. Jimenez-Peris, “Enhancing edge computing with database replication,” in *Reliable Distributed Systems, 2007. SRDS 2007. 26th IEEE International Symposium on*. IEEE, 2007, pp. 45–54.
[9] K. M. Konwar, N. Prakash, N. Lynch, and M. Medard, “A layered architecture for erasure-coded consistent distributed storage,” *arXiv preprint arXiv:1703.01286*, 2017.
[10] Z. Wang and V. R. Cadambe, “Multi-version coding in distributed storage,” in *IEEE International Symposium on Information Theory (ISIT), 2014*, pp. 871–875.
[11] ———, “On multi-version coding for distributed storage,” in *Allerton Conference on Communication, Control, and Computing*, 2014, pp. 569–575.
[12] A. G. Dimakis, P. B. Godfrey, Y. Wu, M. J. Wainwright, and K. Ramchandran, “Network coding for distributed storage systems,” *IEEE Transactions on Information Theory*, vol. 56, no. 9, pp. 4539–4551, 2010.
[13] I. Tamo and A. Barg, “A family of optimal locally recoverable codes,” *IEEE Transactions on Information Theory*, vol. 60, no. 8, pp. 4661–4676, 2014.
[14] A. El Gamal and Y.-H. Kim, *Network information theory*. Cambridge university press, 2011.
[15] D. Slepian and J. K. Wolf, “Noiseless coding of correlated information sources,” *IEEE Transactions on Information Theory*, vol. 19, no. 4, pp. 471–480, 1973.
[16] A. Wyner, “Recent results in the shannon theory,” *IEEE Transactions on information Theory*, vol. 20, no. 1, pp. 2–10, 1974.
