Multi-Version Coding in Distributed Storage

Zhiying Wang
Department of Electrical Engineering,
Stanford University, Stanford, USA
zhyingw@stanford.edu Viveck Cadambe
Research Laboratory of Electronics,
Massachusetts Institute of Technology, Cambridge, USA
viveck@mit.edu

Abstract

We investigate an information theoretic problem motivated by storing multiple versions of a data object in distributed storage systems. Specifically, in a storage system with $n$ server nodes, where there are $\nu$ independent message versions, each server receives message values corresponding to some arbitrary subset of the versions. The versions are assumed to be totally ordered. Each server is unaware of the set of versions at the other servers, and aims to encode the values corresponding to the versions it has. We investigate codes where, from any set of $c$ nodes ($c < n$), the value corresponding to the highest common version, as per the version ordering, available at this set of $c$ nodes is decodable. We aim to design codes that minimize the storage cost.

We present two main results in this paper. First, we show that the storage cost is lower bounded by $(1 - (1 - \frac{1}{c})^\nu)$, measured in terms of the bits of the values. Second, for the cases of $\nu = 2$ and $\nu = 3$, we provide new code constructions that respectively achieve storage costs of $\frac{2c-1}{c^2}$ and $\frac{3c-2}{c^2}$, measured in terms of the bits of the values. Our code constructions are simple in that we do not code across versions. We argue that when the number of versions $\nu$ is much larger than $c$, then replication is close to optimal.

I. INTRODUCTION

There is an enormous interest in recent times to understand the role of coding in distributed storage systems. In this paper, we formulate a new information theoretic problem motivated by applications of distributed storage systems to store data that is changing, where the goal is to expose the latest version of the data. We begin this paper with an informal description of our problem formulation. Later in this section, we describe the motivation for our problem formulation.

Consider a distributed storage system with a set $\mathcal{N}$ of $n$ servers, where $n = |\mathcal{N}|$. Suppose that it stores data $W_1$ using an $n$ length code, such that a client can connect to any subset of $c$ servers and decode $W_1$. Now, suppose an updated version of the data $W_2$ enters the system. Now, suppose that, for reasons that may be related to network failures, $W_2$ arrives at some servers, but not others. Let us denote by $\mathcal{S} \subseteq \mathcal{N}$ the set of servers that have received version $W_2$. We assume that each server is unaware of the set $\mathcal{S}$. The question of interest here is to design a storage strategy for the servers so that, a client can connect to any $c$ servers and decode the latest common version among the $c$ servers. That is, $W_2$ must be decodable from every set of $c$ servers that is a subset of $\mathcal{S}$, and $W_1$ must be decodable from every subset that is not. We intend our storage strategy to be applicable for every possible subset $\mathcal{S} \subseteq \mathcal{N}$. A possible scenario is depicted in Fig. 1 for $n = 3, c = 2$.

Notice that in the storage strategy, a server in $\mathcal{S}$ can store a function of $W_1$ and $W_2$, whereas a server outside of $\mathcal{S}$ stores a function of $W_1$ alone. We now describe two simple approaches that solve this problem.

• Replication: In this strategy, we assume that each server stores the latest version it receives, that is servers in $\mathcal{S}$ store $W_2$ and servers in $\mathcal{N}\backslash\mathcal{S}$ store $W_1$. Denoting the size of $W_1$ by one unit, notice that the storage cost of this strategy is $1$ unit per server, or a total of $n$ units. See Table I for an example.

• Simple Erasure Coding: In this strategy, we use two $(n, c)$ MDS codes, one for each version separately. Each server stores one codeword symbol corresponding to each version it receives. So, each server in $\mathcal{S}$ stores two codeword symbols resulting in a storage cost of $\frac{2}{c}$ unit, whereas, each server in $\mathcal{N}\backslash\mathcal{S}$ stores $\frac{1}{c}$ unit. Notice that in the worst case where $\mathcal{S} = \mathcal{N}$, the total storage cost per server is $\frac{2}{c}$ unit. See Table II for an example.

In this paper, we use worst-case storage costs to measure the performance of our codes for simplicity. Therefore, the per server storage cost of replication is equal to $1$ unit, and that of the simple erasure coding strategy is equal to $\frac{2}{c}$ units. The main contribution of this paper is an information theoretic characterization of the storage cost of such codes. In particular, for the setting described we provide an information theoretic lower bound and code construction for codes that achieve a per server storage cost of $\frac{2c-1}{c^2}$.

Table III is an example of this construction. Furthermore, we generalize this setting to scenarios where there are more than $2$ versions.

A. Background and Motivation

Our problem formulated above incorporates two important aspects that are seldom studied in the literature that applies information theory to distributed storage systems. The first aspect is that we study the problem of storing multiple versions

requirements, and the idea of using erasure coding for consistency has been used for this problem in [5], [6], [7], [8]. In fact, these references come from certain key value stores, for instance, applied to storing data in a stock market, where acquiring the latest stock value is of significant importance.

Asynchrony is inherent to the distributed nature of the storage systems that we consider. In particular, asynchrony occurs due to temporary or permanent failures of servers, or of transmission between the clients and the servers. Indeed, asynchronicity is the de-facto model of study in storage systems in the distributed algorithms literature [1].

The problem of storing multiple versions of the data consistently in distributed asynchronous storage systems forms the basis of celebrated results in distributed computing theory [3]. From a practical perspective, algorithms designed to ensure consistency in asynchronous environments form the basis of several storage systems. We refer the reader to [4] for a detailed description of the Amazon Dynamo key value store, which describes replication based data storage techniques and interesting challenges in ensuring consistency in asynchronous settings. While [3], [4] use replication based techniques for fault tolerance, the idea of using erasure coding for consistency has been used for this problem in [5], [6], [7], [8]. In fact, these references use the idea of simple erasure coding that we referred to earlier in this section.

In this paper, we formulate an information theoretic problem in Section II that is inspired by the idea of consistent data

Table captions:

<table>
<thead>
<tr>
<th>Initial</th>
<th>Server 1</th>
<th>Server 2</th>
<th>Server 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ver. 1</td>
<td>W_1</td>
<td>W_2</td>
<td>W_1</td>
</tr>
<tr>
<td>Ver. 2 enters</td>
<td>Ver. 1</td>
<td>W_2</td>
<td>W_1</td>
</tr>
</tbody>
</table>

**TABLE I**

Code C_1 using replication for n = 3, c = 2 and two versions. Every server stores W_1 successfully initially. Then when Version 2 enters the system, the first and second servers receive and store W_2 and remove W_1, but Server 3 receives and does nothing. The client can connect to any c servers and decode the latest version. The storage cost is 1 unit.

of the same data, where the requirement is for clients to acquire the latest possible version of the data; we refer to this requirement as consistency based on the term commonly used in literature related to distributed computing [1]. The second aspect is asynchrony, that is, the idea that each server receives a different set of versions, and each server is unaware of the set of servers that have received a particular version.

Storing multiple versions of the same message consistently is important in several applications. For instance, the idea of requiring the latest version of the object is important in shared memory systems [1] that form the cornerstone of theory and practice of multiprocessor programming [2]. In particular, when multiple threads access the same variable, it is important that the changes made by one thread to the variable are reflected when another thread reads this variable. Another natural example comes from certain key value stores, for instance, applied to storing data in a stock market, where acquiring the latest stock value is of significant importance.

Asynchrony is inherent to the distributed nature of the storage systems that we consider. In particular, asynchrony occurs due to temporary or permanent failures of servers, or of transmission between the clients and the servers. Indeed, asynchronicity is the de-facto model of study in storage systems in the distributed algorithms literature [1].

The problem of storing multiple versions of the data consistently in distributed asynchronous storage systems forms the basis of celebrated results in distributed computing theory [3]. From a practical perspective, algorithms designed to ensure consistency in asynchronous environments form the basis of several storage systems. We refer the reader to [4] for a detailed description of the Amazon Dynamo key value store, which describes replication based data storage techniques and interesting challenges in ensuring consistency in asynchronous settings. While [3], [4] use replication based techniques for fault tolerance, the idea of using erasure coding for consistency has been used for this problem in [5], [6], [7], [8]. In fact, these references use the idea of simple erasure coding that we referred to earlier in this section.

In this paper, we formulate an information theoretic problem in Section II that is inspired by the idea of consistent data

Table captions:

<table>
<thead>
<tr>
<th>Initial</th>
<th>Server 1</th>
<th>Server 2</th>
<th>Server 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ver. 1</td>
<td>p_1, p_2</td>
<td>p_3, p_4</td>
<td>p_1 ⊕ p_2, p_3 ⊕ p_4</td>
</tr>
<tr>
<td>Ver. 2 enters</td>
<td>Ver. 1</td>
<td>q_1, q_2</td>
<td>q_3, q_4</td>
</tr>
</tbody>
</table>

**TABLE II**

Code C_2 with simple erasure coding for n = 3, c = 2 and two versions. Assume each unit is 4 bits, and the bits of the two versions are W_1 = (p_1, p_2, p_3, p_4), W_2 = (q_1, q_2, q_3, q_4). Every version is coded with a (3, 2) MDS code. Initially Version 1 is stored in the servers and any c servers suffice to recover it. When Version 2 enters the system, (q_1 ⊕ q_2, q_3 ⊕ q_4) is lost before reaching Server 3. The client can recover Version 2 if connected to Servers 1 and 2; otherwise it recovers Version 1. The storage cost is 4 bits, or 1 unit.

<table>
<thead>
<tr>
<th>Initial</th>
<th>Server 1</th>
<th>Server 2</th>
<th>Server 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ver. 1</td>
<td>p_1, p_2, p_3</td>
<td>p_1, p_2, p_4</td>
<td>p_1 ⊕ p_2, p_3 ⊕ p_4</td>
</tr>
<tr>
<td>Ver. 2 enters</td>
<td>Ver. 1</td>
<td>q_1, q_2</td>
<td>q_3, q_4</td>
</tr>
</tbody>
</table>

**TABLE III**

Proposed code C_4 for n = 3, c = 2 and two versions. Assume each unit is 4 bits, and the two versions are the same as Table II. Here p_5 = p_1 ⊕ p_2 ⊕ p_3 ⊕ p_4. Initially every server stores 3 bits of Version 1. Connecting to c servers is sufficient to recover Version 1. When Version 2 enters the system, it is coded by a (3, 2) MDS code. (q_1 ⊕ q_2, q_3 ⊕ q_4) is lost before reaching Server 3, and each of the first two servers over-writes 2 bits of Version 1 with Version 2. The latest common version is recoverable. The storage cost is 3 bits, or 3/4 unit.
storage. Our problem formulation is geared towards understanding storage cost, when we aim to obtain the latest possible version from a set of servers. In Section III, we summarize our main results and discuss some insights.

The paper is organized as follows. In Section II we introduce some definitions and formally define the multi-version code and the main results are summarized in Section III. In Section IV we prove a lower bound on the storage cost. In Section V we show explicit code constructions of this problem and in Section VI we show another construction for a special set of scenarios. We conclude in Section VII.

II. SYSTEM MODEL: MULTI-VERSIO CODES

We begin this section with an informal description of a multi-version code. We present a formal definition later in Definition 1. A word on notation before we proceed. We write $[i] := \{1, 2, \ldots, i\}$, for integer $i \in \mathbb{N}$, and $[l, j] := \{l, l + 1, \ldots, j\}$, for integers $l \leq j \in \mathbb{N}$. We also use them to represent the empty set if $i = 0$ or $l < j$. Also, for any set $S = \{s_1, s_2, \ldots, s_{|S|}\} \subseteq \mathbb{Z}$ where $s_1 < s_2 < \ldots < s_{|S|}$, and for any ensemble of variables $\{X_i : i \in S\}$, we denote the tuple $(X_{s_1}, X_{s_2}, \ldots, X_{s_{|S|}})$ by $X_S$.

The multi-version coding problem that we study is parameterized by positive integers $n, c, \nu, M$ and $q$. We consider a setup with $n$ servers, or equivalently, $n$ denotes the length of the code. Our goal is to store $\nu$ independent versions of the message, where each version of the message is drawn from the set $[M]$ with equal probability. We denote the value of the $i$th version of the message by $W_i \in [M]$ for $i \in [\nu]$. The symbols of the codewords come from the set $[q]$, so the quantity $\log_2 q$ can be interpreted as the number of bits stored in each server in the system. Each server receives a different set of versions. We denote $S(i) \subseteq [\nu]$ to be the set of versions received by the $i$th server. We refer to the set $S(i)$ as the state of the $i$th server. The server encodes and stores the versions that it receives (or equivalently, the versions in its state). More specifically, for the $i$th server, denoting its state $S(i)$ as $S = S(i) = \{s_1, s_2, \ldots, s_{|S|}\}$ where $s_1 < s_2 < \ldots < s_{|S|}$, the $i$th symbol of the codeword is generated by an encoding function $\varphi^{(i)}_S$ that takes an input, $W_S = (W_{s_1}, W_{s_2}, \ldots, W_{s_{|S|}})$, and outputs an element in $[q]$.

We assume that there is a total ordering $\prec$ on the versions, $W_i \prec W_j$, if $i < j$. For any set of servers $T \subseteq [n]$, we refer to $\max_{i \in T} S(i)$ as the latest common version in the set of servers $T$. The purpose of multi-version code design is to generate encoding functions such that, for every subset $T \subseteq [n]$ of $c$ servers, the latest common version in $T$ can be decoded from the corresponding $\log M$ codeword symbols\(^1\) for every possible sequence of states $(\{S(1), \ldots, S(n)\} : S(i) \subseteq [\nu])$. The goal of the problem is to find the smallest possible storage cost per bit stored, or more precisely, to find the smallest possible value of $\frac{\log q}{\log_2 M}$ over all possible $(n, c, \nu, M, q)$ codes.

We now proceed to provide a formal definition of a multi-version code. In the sequel, for an element $S \in \mathcal{P}([\nu]^n)$ where $\mathcal{P}\{[\nu]\}$ denotes the power set of $[\nu]$, we denote $S(i)$ to be the $i$th component of the tuple $S$, for $i = 1, 2, \ldots, n$. Note that $S(i)$ is a subset of $[\nu]$.

**Definition 1 (Multi-version code)** An $(n, c, \nu, M, q)$ multi-version code consists of

- encoding functions $\varphi_S^{(i)} : [M]|S| \rightarrow [q],$
- for every $i \in [n]$ and every $S \subseteq [\nu]$, and
- decoding functions $\psi_S^{(T)} : [q] \rightarrow [M] \cup \{\emptyset\},$
- for every set $S \in \mathcal{P}([\nu]^n)$ and set $T \subseteq [n]$ where $|T| = c$, that satisfy
  \[
  \psi_S^{(T)}(\varphi_{S(t_1)}(W_{S(t_1)}), \ldots, \varphi_{S(t_c)}(W_{S(t_c)}))
  = \begin{cases}
  W_{\max_{i \in T} S(i)} & \text{if } \cap_{i \in T} S(i) \neq \emptyset, \\
  \emptyset, & \text{a.w.,}
  \end{cases}
  \]

for every $W_{[\nu]} \in [M]^n$, where $T = \{t_1, t_2, \ldots, t_c\}$, $t_1 < \cdots < t_c$.

For parameters $n, c, \nu, \nu$, the goal of the multi-version coding problem is to find the infimum, taken over the set of all $(n, c, \nu, M, q)$ codes, of the quantity: $\frac{\log q}{\log M}$. We refer to quantity $\frac{\log q}{\log M}$ as rate of a multi-version code.

A code is called causal if upon arrival of a version at a server, the encoding function is only a function of its stored information and the newly arrived information: for all $S \subseteq [\nu], j \in S, i \in [n]$, exists a function $\varphi_{S,j}^{(i)} : [q] \times [M] \rightarrow [q]$, such that $\varphi_S^{(i)}(a_S) = \varphi_{S,j}^{(i)}(\varphi_{S,j}^{(i)}(a_{S\setminus\{j\}}), a_j), \forall j \in S$.

\(^1\)We note that the quantity $c$ captures, physically, the extent of accessibility of the system. This quantity is similar to the notion of locality in distributed storage systems (see [9], [10], for instance).
In our first main result, we have a lower bound for the optimal rate.

**Theorem 1** For $c \geq 2, \nu \geq 2, n \geq c + \nu - 1$, every $(n, c, \nu, M, q)$ multi-version code satisfies

$$\frac{\log q}{\log M} \geq 1 - (1 - \frac{1}{c})^\nu.$$

We show the proof of this theorem for $\nu = 2$ versions in Section IV. Then we prove for the general case.

In our second main result, we show in Section V by explicit code construction that the above bound on the rate is tight for $\nu = 2$. Moreover, we have a construction for $\nu = 3$ versions and arbitrary number of servers $n$, and number of connected servers $c$.

**Theorem 2** For $c \geq 2$, there exists an optimal $(n, c, 2, M, q)$ code with $\frac{\log q}{\log M} = 1 - (1 - \frac{1}{c})^2 = \frac{2c - 1}{c^2}$. There exists an $(n, c, 3, M, q)$ code with $\frac{\log q}{\log M} = \frac{3c - 2}{c^2}$.

Our constructions outperform replication, which achieves a rate of 1, and simple MDS erasure coding, which, for sufficiently large $q$, achieves a rate of $\frac{c}{2}$. Our code constructions are, in fact, quite simple, that is, we do not code across versions. The main idea in our approach is to carefully allocate the storage “budget” of $\log_2 q$ among the various versions in a server’s state, and code within the versions. It is an interesting question whether for $\nu \geq 3$, coding across versions may help to improve the storage cost.

Previous studies [6], [7] in ensuring consistency have studied the impact of having multiple versions in the system on storage cost. The results of [6], [7] seem to suggest that, when multiple versions appear in the system simultaneously, replication is a good strategy, whereas, when the number of concurrent versions in the system is relatively small, then simple coding has significant benefits. Note that the quantity $\nu$ is a measure of the number of simultaneous versions that are in the storage system. Interestingly, our results reflect these insights at extremal points. Interestingly, when $\nu$ is large compared with $c$, replication is close to optimal, and approaches optimality as $\nu$ tends to infinity. On the other hand, when $\nu$ is relatively small compared with $c$, simple erasure coding is close to optimal. Exploration of codes for moderate values of $\nu$ is an interesting open question that can yield significant benefits to the applications discussed here. Another interesting future question is to explore the possibility of incorporating our lower bounds and/or our code constructions to obtain insights under formal models of consistency.

**Theorem 3.** For positive integers, $N, f, \nu, M, q$, suppose that an $(N, N - f, \nu, M, q)$ exists. Then, there exists SWMR atomic shared memory algorithm for a system with $N$ server nodes that satisfies atomicity and liveness so long as the number of writes concurrent with a read that is no bigger than $\nu$, and incurs a storage cost of $\frac{\log q}{\log M}$.

We prove the theorem by constructing an algorithm. Our algorithm is described in Fig. ??.
IV. PROOF OF A LOWER BOUND

In this section we find a lower bound on the rate of the multi-version code for arbitrary values of the number of connected servers \( c \geq 2 \), the number of versions \( \nu \geq 2 \), and the number of servers \( n \geq c + \nu - 1 \). By definition any valid code satisfies the encoding and decoding functions. We link the storage size \( \log q \) and the version size \( \log M \) using this definition, as well as manipulations of some entropy quantities. The key idea in the proof is to bound the mutual information between the information stored in a set of \( c - 1 \) servers and all the \( \nu \) versions. In fact, there are quite a few different ways to write such mutual information by the chain rule, each resulting in a different inequality. In the proof for general \( \nu \), we average these inequalities with proper weights and obtain the desired rate bound.

We first list some notations and conditions. Then we show two helpful lemmas. To provide an intuitive understanding of the proof techniques, we show a proof for the case of \( \nu = 2 \) versions, which is slightly different from the main proof for the general case. The bound for arbitrary number of versions is Theorem 1.

In the following sections, we will use \( \alpha = \log_2 q \) and \( B = \log_2 M \) to represent the number of bits at each server and for each version, respectively. Server \( i \) stores a symbol from the set \([q]\), and every symbol is associated with certain probability to appear. Therefore, we use \( X_i \) to represent the random variable of the symbol stored at Server \( i \), \( i \in [n] \), taking values from \([q]\). The value of the message of Version \( j \) is drawn from the set \([M]\) with equal probability, therefore by abuse of notation we use \( W_j \) as the random variable of the value of Version \( j \), \( j \in [\nu] \), distributed uniformly at random over \([M]\). We use \( H(\cdot) \) to represent the entropy function and \( I(\cdot;\cdot) \) the mutual information function. We use \( P \) to denote set of integers \( \{1, \ldots, c-1\} \), and \( P_i \) the set of integers \( P \setminus \{i\} \), for \( i \in [c-1] \). When \( c = 2 \), define \( P_1 \) as the empty set.

Recall that for random variables \( \{Y_i, i \in \mathcal{I}\} \) and subset \( Q \subseteq \mathcal{I} \), \( Y_Q \) represents the tuple of random variables indexed by \( Q \). If \( Q \) is empty, we define \( Y_Q \) as a constant, then entropy \( H(Y_Q) = 0 \), and conditional entropy \( H(\cdot|Y_Q) = H(\cdot) \).

Since the multi-version code should work under any states, let us focus on the following states: Servers \( 1, 2, \ldots, c \) have all \( \nu \) versions, and Server \( c+i \) has versions \( \nu-i \), for \( i \in [\nu-1] \). That is, we consider for servers \( [c+c-1] \) the states

\[
S(i) = \begin{cases} 
\nu, & i \in [c], \\
[c+c-1], & i \in [c+c-1].
\end{cases}
\]

where \( S(i) \) represents the state of the \( i \)th server.

Next we translate the setup of the rate problem under the states (1) to four entropy expressions. We know that each server stores no more than \( \log_2 q = \alpha \) amount of information: for \( i \in [n] \),

\[
H(X_i) \leq \alpha. \tag{2}
\]

All the \( \nu \) versions are independent and of size \( \log_2 M = B \) for \( i \in [\nu] \), \( Q \subseteq [\nu] \setminus \{i\} \),

\[
H(W_i|W_Q) = H(W_i) = B. \tag{3}
\]

The information at Server \( i \) is a function of its versions \( W_{S(i)} \):

\[
H(X_i|W_{S(i)}) = 0. \tag{4}
\]

For any \( i \in [\nu] \) and set \( Q \subseteq [c+c-1], |Q| = c-1 \), the latest common version \( W_i \) can be uniquely determined by servers \( Q \cup \{c+c-1\} \):

\[
H(W_i|X_Q, X_{c+c-1}) = 0. \tag{5}
\]

Before proving the main result, we first prove two lemmas about some mutual information. The first lemma shows a recursion that relates versions \([i]\) to versions \([i-1]\).

**Lemma 1** Recall \( P = [c-1] \), and \( P_i = P \setminus \{i\} \). The following inequality is satisfied for \( d \in [\nu] \):

\[
I(X_P; W_{[d-1]}|W_{[d-1]}) \geq B - \alpha + \frac{1}{c-1} \sum_{i=1}^{c-1} I(X_{c+c-1-d}; W_{[d-1]}|X_{P_i}).
\]

And similarly, for any \( i \in [c-1], j \in [c+c-1-d] \setminus P_i \):

\[
I(X_{[j]} \cup P_i; W_{[d-1]}|W_{[d-1]}) \geq B - \alpha + I(X_{c+c-1-d}; W_{[d-1]}|X_{P_i}).
\]
Proof: We only show the first inequality, the other one follows similarly. Notice that for any \( i \in P \), since \( X_P, X_{c+\nu-d} \) determines \( W_d \), and \( W_d \) is independent of other versions,

\[
I(X_P; W_d | W_{(d-1)}) = I(X_P, X_{c+\nu-d}; W_d | W_{(d-1)}) - I(X_{c+\nu-d}; W_d | W_{(d-1)}, X_P)
\]
\[
= H(W_d | W_{(d-1)}) - H(W_d | X_P, X_{c+\nu-d}, W_{(d-1)})
- H(X_{c+\nu-d} | W_{(d-1)}, X_P) + H(X_{c+\nu-d} | W_{d}, X_P)
\]

\[
\geq B - H(X_{c+\nu-d} | W_{(d-1)}, X_P) + H(X_{c+\nu-d} | W_d, X_P)
\]

where \( (3)(5) \) implies \( B \geq B - H(X_{c+\nu-d} | W_{(d-1)}, X_P) + H(X_{c+\nu-d} | W_d, X_P) \).

Remark: For the special case of \( c = 2 \), the above lemma becomes

\[
I(X_P; W_d | W_{(d-1)}) \geq B - \alpha + I(X_{c+\nu-d}; W_{(d-1)}).
\]

For the special case of \( d = 1 \), the lemma becomes for every subset \( Q \subseteq [c + \nu - 2], |Q| = c - 1 \),

\[
I(X_Q; W_i) \geq B - \alpha.
\]  

(6)

Notice that in (6) \( \alpha \) can be interpreted as the amount of information stored at Server \( c + \nu - 1 \), and \( B \) as the amount of information of Version 1. Intuitively, we can understand (6) like this: the information stored at servers in \( Q \) combined with the information at Server \( c + \nu - 1 \) is sufficient to recover Version 1.

The next lemma is similar to the fact that the sum of entropy of single variables is no less than the joint entropy.

Lemma 2 The following holds for \( c \geq 3, j \in [\nu] \):

\[
\frac{1}{c-2} \sum_{i=1}^{c-1} I(X_P; W_{[j+1,\nu]} | W_j) \geq I(X_P; W_{[j+1,\nu]} | W_j).
\]

Proof: First notice that

\[
I(X_P; W_{[j+1,\nu]} | W_j) = H(X_P | W_j) - H(X_P | W_j)
\]

(4)

\[
H(X_P | W_j) - 0.
\]

Similarly

\[
I(X_P; W_{[j+1,\nu]} | W_j) = H(X_P | W_j).
\]

Now we can write left hand side in the lemma as

\[
LHS = \frac{1}{c-2} \sum_{i=1}^{c-1} H(X_P | W_j)
\]

\[
= \frac{1}{c-2} \sum_{i=1}^{c-2} H(X_P | W_j) + \frac{1}{c-2} H(X_{c-1} | W_j)
\]

\[
= \frac{1}{c-2} \sum_{i=1}^{c-2} H(X_P | W_j) + \frac{1}{c-2} \sum_{i=1}^{c-2} H(X_i | X_{i-1}, W_j)
\]

\[
\geq \frac{1}{c-2} \sum_{i=1}^{c-2} H(X_P | W_j) + \frac{1}{c-2} \sum_{i=1}^{c-2} H(X_i | X_P, W_j)
\]

\[
= H(X_P | W_j),
\]

which is equal to the right hand side.

The following theorem shows the lower bound for \( \nu = 2 \) versions.


**Theorem 4** An \((n, c, 2, M, q)\) multi-version code must have a rate satisfying
\[
\frac{\log q}{\log M} \geq \frac{2c-1}{c^2}.
\]

**Proof:** Consider the collection of random variables \(X_{[c-1]}\).
\[
(c-1)\alpha \geq H(X_{[c-1]}) = I(X_{[c-1]}; W_1, W_2) = I(X_{[c-1]}; W_1) + I(X_{[c-1]}; W_2|W_1).
\]
But we have by (6) the first term satisfies
\[
I(X_{[c-1]}; W_1) \geq B - \alpha.
\]
And we also have similar to the proof in Lemma 2 for the second term
\[
I(X_{[c-1]}; W_2|W_1) = H(X_{[c-1]}|W_1).
\]
Moreover, we show next that the following holds: without loss of generality
\[
H(X_{[c-1]}|W_1) \geq \frac{c-1}{c} B. \tag{7}
\]
If (7) holds, note that our proof will be complete because
\[
(c-1)\alpha \geq B - \alpha + \frac{c-1}{c} B.
\]
Again similar to the proof in Lemma 2, the inequality (7) holds because
\[
\sum_{i=1}^c H(X_{[c]}\setminus\{i\}|W_1) \geq (c-1)H(X_{[c]}|W_1)
\]
\[
= (c-1)(H(X_{[c]}, W_1) - H(W_1))
\]
\[
= (c-1)(H(X_{[c]}, W_1, W_2) - H(W_2|X_{[c]}, W_1) - H(W_1))
\]
\[
= (c-1)(H(W_1, W_2) - 0 - H(W_1))
\]
\[
= (c-1)H(W_2) = (c-1)B. \tag{8}
\]
Here (8) comes from (4)(5), and (9) comes from (3). Without loss of generality, we can assume that \(H(X_{[c-1]}|W_1)\) is no less than the average value of items in the sum, otherwise we can simply permute the Servers 1, \ldots, c. Therefore the bound follows.

In this proof, (7) evaluates the storage size of Version \(\nu\). Notice that the above proof used the fact that we are considering the largest storage cost among all servers. If it happens that (7) is not satisfied for some permutations of the first \(c\) servers, then the lower bound we get for this permutation would be smaller and not tight. This “weakness” is removed in the proof for the general case, and we will not use result similar to (7).

Next we move to the general parameters. Let us define some constants that will later serve as the weights when we add up some useful inequalities:
\[
c_j = \frac{c-2}{c-1} \left( \frac{c}{c-1} \right)^{\nu-1-j}, \quad j = 1, 2, \ldots, \nu - 1.
\]
\[
c_0 = 1 - \sum_{j=1}^{\nu-1} c_j.
\]
Notice that for the special case of \(c = 2\), \(c_0 = 1\), and \(c_j = 0\) for all \(j > 0\). It is easy to check that the following equality holds for \(d \in [\nu]\) and all \(c \geq 2\):
\[
\sum_{j=0}^{d-1} \frac{c_j}{c-1} + \sum_{j=d}^{\nu-1} \frac{c_j}{c-2} = \frac{1}{c-1} \left( \frac{c}{c-1} \right)^{\nu-d}. \tag{10}
\]
And for \(d \in [2, \nu]\) the following equality holds:
\[
\sum_{j=0}^{d-1} \frac{c_j}{c-1} + \sum_{j=d}^{\nu-1} \frac{c_j}{c-2} = \frac{c_{d-1}}{c-2}. \tag{11}
\]
To make the proof easier, we use the following series of values, for \( d = 1, 2, \ldots, \nu \), \( g(d) \) is defined as
\[
\sum_{j=0}^{d-1} c_j \left( \frac{1}{c - 2} \sum_{i=1}^{c-1} I(X_{P_i}; X_j) + I(X_{P_j}; X_{[j+1, d]}|X_{[j+1, d]}) \right);
\]
and \( f(d) \) is
\[
f(d) = g(d) + \sum_{j=d}^{\nu-1} \frac{c_j}{c - 2} \sum_{i=1}^{c-1} I(X_{\{c+\nu-j-1\}\cup P_i}; X_{[d]}) \cdot
\]
When \( d = \nu \), the last addend in \( f(d) \) becomes zero. For the special case of \( c = 2 \), we treat the first addend in \( g(d) \) as zero. Now we are ready to prove the general lower bound.

**Proof of Theorem 1:**

We write the capacity of the first \( c - 1 \) servers in \( \nu \) different ways for \( j = 0, 1, \ldots, \nu - 1 \):
\[
(c - 1)\alpha
\]
\[
\geq \frac{1}{c - 2} \sum_{i=1}^{c-1} I(X_{P_i}; X_{[j]})
\]
\[
= \frac{1}{c - 2} \sum_{i=1}^{c-1} (I(X_{P_i}; X_{[j]}) + I(X_{P_i}; X_{[j+1, \nu]}|X_{[j]}))
\]
\[
\geq \frac{1}{c - 2} \sum_{i=1}^{c-1} I(X_{P_i}; X_{[j]}) + I(X_{P_i}; X_{[j+1, \nu]}|X_{[j]}).
\]
(12)

Here (12) follows from the chain rule, and (13) follows from Lemma 2. In the following, we will omit the indices in the summation of \( i \), and simply write \( \sum_{i=1}^{c-1} (\cdot) = \sum_i (\cdot) \). Next, we can sum up the above expressions using coefficients \( c_j \)’s:
\[
(c - 1)\alpha
\]
\[
\geq \sum_{j=0}^{\nu-1} c_j \left( \frac{1}{c - 2} \sum_{i} I(X_{P_i}; X_{[j]}) + I(X_{P_i}; X_{[j+1, \nu]}|X_{[j]}) \right)
\]
\[
= f(\nu).
\]

We claim the following recursive expression for \( d = 2, 3, \ldots, \nu \).
\[
f(d) \geq f(d - 1) + c_{d-1} \frac{c-1}{c-2}(B - \alpha).
\]
(14)

It is proved as follows:
\[
f(d) = \sum_{j=0}^{d-2} c_j \left( \frac{1}{c - 2} \sum_{i} I(X_{P_i}; X_{[j]}) + I(X_{P_i}; X_{[j+1, d-1]}|X_{[j]}) \right)
\]
\[
+ \sum_{j=d}^{\nu-1} \frac{c_j}{c - 2} \sum_{i} I(X_{\{c+\nu-j-1\}\cup P_i}; X_{[d-1]})
\]
\[
= \frac{c_{d-1}}{c - 2} \sum_{i} I(X_{P_i}; X_{[d-1]}) + \sum_{j=0}^{d-1} c_j I(X_{P_i}; X_{[d-1]})
\]
\[
+ \frac{c_{d-1}}{c - 2} \sum_{i} I(X_{\{c+\nu-j-1\}\cup P_i}; X_{[d-1]})
\]
\[
\geq g(d - 1) + \sum_{j=d}^{\nu-1} \frac{c_j}{c - 2} \sum_{i} I(X_{\{c+\nu-j-1\}\cup P_i}; X_{[d-1]})
\]
\[
+ \left( \frac{c_{d-1}}{c - 2} \sum_{i} I(X_{P_i}; X_{[d-1]}) + \left( \sum_{j=0}^{d-1} \frac{c_j}{c - 1} + \sum_{j=d}^{\nu-1} \frac{c_j}{c - 2} \right) \right)
\]
\[
\times \sum_{i} (I(X_{c+\nu-d}; X_{[d-1]}|X_{P_i}) + B - \alpha).
\]
(16)
\[
g(d-1) + \sum_{j=d}^{\nu-1} \frac{c_j}{c-2} \sum_{i} I(X_{c + \nu - j - 1} \cup P_i; W_{d-1}) \\
+ \frac{c d - 1}{c - 2} \left( I(X_{c + \nu - d} \cup P_i; W_{d-1}) + B - \alpha \right) \\
= f(d-1) + c d - 1 \frac{c - 1}{c - 2} (B - \alpha).
\]

Here (15) follows from the chain rule, (16) follows from Lemma 1, and (17) follows from (11) and the chain rule. Moreover, the first term \(f(1)\) in the series satisfies:

\[
f(1) = c_0 I(X_1; W_1) + \sum_{j=1}^{\nu-1} \frac{c_j}{c-2} \sum_{i} I(X_{c + \nu - j - 1} \cup P_i; W_1) \\
\geq \left( c_0 + \frac{c - 1}{c - 2} \sum_{j=1}^{\nu-1} c_j \right) (B - \alpha) \\
= \left( \frac{c}{c - 1} \right)^{\nu-1} (B - \alpha).
\]

Here (18) follows from (6), and (19) follows from (10). Therefore, we can bound the information in the first \(c\) servers as:

\[
(c - 1)\alpha \geq f(\nu) \geq f(\nu - 1) + B - \alpha \geq \ldots \\
\geq (B - \alpha) \left( 1 + \frac{c}{c - 1} + \cdots + \left( \frac{c}{c - 1} \right)^{\nu-1} \right).
\]

Hence, simplifying the above expression we have shown the lower bound.

Note here that if \(c = 2\), all the proof steps follows. We have marked the special values of some of the quantities and inequalities (e.g., remark after Lemma 1) in the text.

V. Code Construction

In this section we give a code construction for \(\nu = 2, 3\) versions that follows closely as the example in the introduction. The storage cost in the construction is no more than the replication or simple erasure coding solutions, i.e., \(\log \frac{q}{\log M} \leq \min\{1, \nu/c\}\), for \(c \geq 2\). Again we use \(\alpha = \log_q q, B = \log_2 M\).

The main idea of the construction is as follows. Firstly, versions are encoded separately using MDS codes (with different code rates), such that any \(B\) amount of coded symbol of a version suffices to recover it. When a large enough finite field is used, or equivalently when a sufficiently large \(q\) is used, such MDS codes always exist. So from now on, we only concentrate on the size of the coded symbol, not on the actual choice of MDS codes. Secondly, we assign to each version a certain amount of storage size based on the server state. As a result, a version is recoverable if the connected servers have at least \(B\) amount of storage size assigned for this version. The amount of assigned storage size for each version is obtained by linear programming for our constructions.

**Construction 1** Construct a code for \(\nu = 2, 3\) versions. Let the storage cost of each server be \(\alpha = \frac{\nu c - \nu + 1}{\nu} B, \nu = 2, 3\). Let \(\alpha_1 = \frac{2(c-1)}{c-2} B, \alpha_2 = \frac{c-2}{c-2} B\). The encoding function only depends on the state, not on the server index. Namely, the encoding function satisfies \(\varphi_S^{(1)} = \varphi_S^{(2)} = \cdots = \varphi_S^{(S)}\). For all states of a server, assign the storage size as in Table IV for \(\nu = 2\) versions, and Table V for \(\nu = 3\) versions. Every version is coded separately using a corresponding MDS code.

For example, in Table III, we can set \(\beta_1 = \alpha = 3B/4, \alpha_1 = B/4, \beta_2 = \alpha_2 = B/2\). Notice that Table III does not use a \((9, 4)\) MDS code for Version 1, but the MDS property is in fact a sufficient condition.

The above construction is indeed a multi-version code, as Theorem 2 claims.

**Proof of Theorem 2:** First it is easy to see that the every state has a storage cost of \(\alpha\). Then we need to check that for every \(c\) connected servers, the newest common version is reconstructible. We prove for \(\nu = 3\) versions, and the case of \(\nu = 2\) by

<table>
<thead>
<tr>
<th>State</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version 1</td>
<td>(\alpha)</td>
<td>(\alpha - B/c)</td>
<td>(\alpha - B/c)</td>
</tr>
<tr>
<td>Version 2</td>
<td>(\alpha)</td>
<td>(B/c)</td>
<td>(B/c)</td>
</tr>
</tbody>
</table>

**TABLE IV**

Storage assignment for a code with \(\nu = 2\). Blank entries indicate that the state does not include the version, e.g., State 1 contains only Version 1. The storage cost is \(\alpha = \frac{2c-1}{c-2} B\).
Similarly in the second case, let \( c \) be the number of connected servers at States 1, 7, respectively, then we have
\[
c_1 \alpha + (c - c_1) \geq B.
\]
By the MDS property, Version 1 can be recovered. When \( \nu = 2 \), the code is optimal by Theorem 1.

Here is a general construction for arbitrary states.

**Construction 2** A \((n, c, \nu, M, q)\) code with storage cost \( \alpha = \frac{\nu c - \nu + 1}{c} B \). For every state \( S \subseteq [\nu] \) and every Version \( i, i \in [2, \delta] \), assign a storage of size
\[
\alpha_i = \nu B/c.
\]
Assign the remaining storage size to Version 1,
\[
\alpha_1 = \alpha - \sum_{i \in S \setminus \{1\}} \alpha_i.
\]
Encode every version with a corresponding MDS code.

Notice that in this construction, every version is stored as an \((n, c)\) MDS code, as the simple erasure coding scheme. Only Version 1 is assigned different storage size according to the state. The following result is similar to the case of \( \nu = 2, 3 \), and the proof is omitted.

**Theorem 5** There exists an \((n, c, \nu, M, q)\) multi-version code with rate \( \frac{\nu c - \nu + 1}{c} \).

We would like to point out the low complexity to update information in the servers in the above constructions. Whenever a different version arrives, the server only need to over-write part of previously stored versions by the newly arrived version. That is, the server can always encode based on its stored information and the newly arrived version. Therefore, the above construction is causal. For example, in Table V, suppose a server has Versions 2 and 3 initially (at State 6), and then Version 1 arrives (State 7). The server simply keeps \( B/c \) amount of Version 2, and leaves the rest storage of size \( \alpha_2 \) for Version 1.

### VI. Code Restricted to Certain States

In this section, we present a code construction for general \( \nu \) but restricted to a special set of server states.

First let us define what we mean by restriction. In Definition 1, if a multi-version code only decodes correctly for a subset of server states \( S \subseteq \mathcal{P}([\nu])^n \), we say it is restricted to states \( S \).

We say that a version is fresh if it is newest in one of the connected servers. In the following code, we only try to recover a fresh and common version. This notion can be applied in situations where a version is obsolete to the client if it is out-of-date to the servers. For fixed \( c \) and every set of connected servers \( T = \{t_1, \ldots, t_c\} \subseteq [n] \), we say an element \( S \in S \) is fresh and common if the newest common version is also fresh:
\[
\max \cap_{i=1}^c S(t_i) = \left( \cap_{i=1}^c S(t_i) \right) \cap \left( \cup_{i=1}^c \max S(t_i) \right).
\]

Now we construct codes for this special set of states.

**Construction 3** We construct an \((n, c, \nu, M, q)\) code restricted to fresh and common states \( S \). If Version \( i \) is the newest at a server, \( i \in [\nu] \), assign size of \( \alpha_i \); otherwise assign size of \( \beta_i = \alpha - \sum_{j=1}^{i-1} \alpha_j \) to it. Here
\[
\alpha_i = \frac{B - \alpha}{c - 1} \left( \frac{c}{c - 1} \right)^{i-1},
\]
and

\[ \alpha = \sum_{i=1}^{\nu} \alpha_i = B \left( 1 - \left( 1 - \frac{1}{c} \right)^\nu \right). \]

Then every version is coded with an MDS code.

For another example, let us assume that Server \( c + \nu - i \) has versions \([i], i \in [\nu], \) and Servers \( 1, 2, \ldots, c - 1 \) have all the versions. Therefore, we can assign rate as in Table VI.

**Theorem 6** Construction 3 is an \((n, c, \nu, M, q)\) code with a storage cost \(\alpha = B \left( 1 - \left( 1 - \frac{1}{c} \right)^\nu \right)\) restricted to fresh and common states \(S\).

**Proof:** First notice that the amount of information in each server is exactly \(\alpha\). Second we will prove that we can recover the fresh and common Version \(i\). We know there exists a connected server where Version \(i\) is the newest. Thus we have \(\beta_i\) amount of Version \(i\) from that server. Also the other \((c - 1)\) connected servers must contain Version \(i\), and the size is at least \(\min\{\alpha_i, \beta_i\} = \alpha_i\). However, one can check that

\[ \beta_i + (c - 1)\alpha_i = B. \]

Thus by the MDS property, Version \(i\) can be recovered. \(\square\)

In fact, Construction 1 coincides with Construction 3 when \(\nu = 2\). Moreover every set of \(c\) states is also a set of fresh and common states, for \(\nu = 2\). But this is not true for general number of versions. For example, when \(\nu = 3, c = 2\), if the connected servers have Versions 1, 2 and 1, 3, respectively, we know the storage size for Version 1 is \(\alpha_1 = 1/8\) in both servers. Then the newest common version, Version 1, is not reconstructible.

Construction 1 is causal, since the encoding simply over-writes part of previously stored information by information of the newly arrived version.

**VII. Concluding Remarks**

In this paper, we have proposed a new coding problem, where information of different versions are stored and requested differently in a system. We have given a lower bound on the worst-case storage capacity and provide a simple coding scheme that is optimal for two versions.

The general tight bound is still an open problem, and bounds for other interesting restricted states can also be developed.

If average-case is of interest, our problem could be formulated in terms of storage size per server per state, and one can optimize given the workload distributions of the servers.

Besides storage cost, communication cost is another important issue in distributed storage or algorithms, which indicates the amount of information transmitted to disperse and update information. Therefore, one could generalize our problem and ask the tradeoff between communication and storage cost.

Similar to data deduplication, one would often see dependent versions of information. Storage capacity bound and efficient code constructions under this scenario will be another future direction.

**Acknowledgment**

The authors would like to thank Prof. Nancy Lynch, Prof. Tsachy Weissman, and Prof. Muriel Médard for their for their valuable advice and helpful comments.

**References**


