Multi-Version Coding in Distributed Storage

Zhiying Wang
Center for Science of Information,
Stanford University, Stanford, USA
zhiyingw@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 ν 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 ν 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 $n = 3, c = 2$.

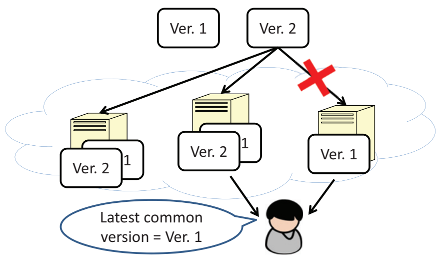

Fig. 1. Version 2 enters the system. The servers are indexed $1, 2, 3$ from left to right. The second version is dispersed and stored at Servers 1 and 2, but does not reach Server 3.

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 for 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

		Server 1	Server 2	Server 3
Initial	Ver. 1	W_1	W_1	W_1
Ver. 2 enters	Ver. 1			W_1
	Ver. 2	W_2	W_2	

TABLE I
Code $\mathcal{C}_1$ 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.

		Server 1	Server 2	Server 3
Initial	Ver. 1	p_1, p_2	p_3, p_4	$p_1 \oplus p_2, p_3 \oplus p_4$
Ver. 2 enters	Ver. 1	p_1, p_2	p_3, p_4	$p_1 \oplus p_2, p_3 \oplus p_4$
	Ver. 2	q_1, q_2	q_3, q_4	

TABLE II
Code $\mathcal{C}_2$ 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, $(q_1 \oplus q_2, q_3 \oplus q_4)$ 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.

information theory to distributed storage systems. The first aspect is that we study the problem of storing multiple versions 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

		Server 1	Server 2	Server 3
Initial	Ver. 1	p_1, p_2, p_3	p_1, p_2, p_4	p_1, p_2, p_5
Ver. 2 enters	Ver. 1	p_3	p_4	p_1, p_2, p_5
	Ver. 2	q_1, q_2	q_3, q_4	

TABLE III
Proposed code $\mathcal{C}_3$ 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 $p_5 = p_1 \oplus p_2 \oplus p_3 \oplus p_4$. 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. $(q_1 \oplus q_2, q_3 \oplus q_4)$ 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.

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 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. Details of our results are shown in the last two sections.

II. System Model: Multi-version 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 $S = \{s_1, s_2, \ldots, s_{|S|}\} \subset \mathbb{Z}$ 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, ν, M and q. We consider a setup with n servers, or equivalently, n denotes the length of the code. Our goal is to store ν independent versions of the message, where each version of the message is drawn from the set $[M]$. We denote the value of the ith 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 $\mathbf{S}(i) \subseteq [\nu]$ to be the set of versions received by the ith server. We refer to the set $\mathbf{S}(i)$ as *the state* of the ith server. Denote $\mathbf{S} = (\mathbf{S}(1), \ldots, \mathbf{S}(n)) \in \mathcal{P}([\nu])^n$ as the sequence of states, where $\mathcal{P}([\nu])$ denotes the power set of $[\nu]$. The server encodes and stores the versions that it receives (or equivalently, the versions in its state). More specifically, for the ith server, denoting its state $\mathbf{S}(i)$ as $S = \mathbf{S}(i) = \{s_1, s_2, \ldots, s_{|S|}\}$ where $s_1 < s_2 < \ldots < s_{|S|}$, the ith symbol of the codeword is generated by an encoding function $\varphi_S^{(i)}$ 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, with $W_i \prec W_j$ if $i < j$. For any set of servers $T \subseteq [n]$, we refer to $\max \cap_{i \in T} \mathbf{S}(i)$ 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 c codeword symbols[1] for every possible sequence of states $\{\mathbf{S} : \mathbf{S} \in \mathcal{P}([\nu])^n\}$. 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 $\frac{\log q}{\log M}$ over all possible (n, c, ν, M, q) codes.

We now proceed to provide a formal definition of a multi-version code.

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

- *encoding functions*

$$\varphi_S^{(i)} : [M]^{|S|} \to [q],$$

for every $i \in [n]$ *and every* $S \subseteq [\nu]$, *and*

- *decoding functions*

$$\psi_{\mathbf{S}}^{(T)} : [q]^c \to [M] \cup \{\emptyset\},$$

for every set $\mathbf{S} \in \mathcal{P}([\nu])^n$ *and set* $T \subseteq [n]$ *where* $|T| = c$, *that satisfy*

$$\psi_{\mathbf{S}}^{(T)} \left(\varphi_{\mathbf{S}(t_1)}^{(t_1)}(W_{\mathbf{S}(t_1)}), \ldots, \varphi_{\mathbf{S}(t_c)}^{(t_c)}(W_{\mathbf{S}(t_c)}) \right) = \begin{cases} W_{\max \cap_{i \in T} \mathbf{S}(i)}, & \textit{if } \cap_{i \in T} \mathbf{S}(i) \neq \phi, \\ \emptyset, & \textit{o.w.}, \end{cases}.$$

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

For parameters n, c, ν, the goal of the multi-version coding problem is to find the infimum, taken over the set of all (n, c, ν, 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.

III. Main Results

In this section, we summarize the main results of this paper. We present two main results in this paper, Theorem 1 and Theorem 2.

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, ν, 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. The proof for arbitrary values of ν can be found in the full version of this paper [11].

[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 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 $\frac{\nu}{c}$. 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.

Interestingly, when ν is large compared with c, replication is close to optimal, and approaches optimality as ν tends to infinity. On the other hand, when ν is relatively small compared with c, simple erasure coding is close to optimal. Exploration of codes for moderate values of ν 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.

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 ν versions.

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. And its proof can be found in the full version of this paper [11].

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 $[M]$, therefore by abuse of notation we use W_j as the random variable of the value of Version j, $j \in [\nu]$, distributed over $[M]$. We use $H(\cdot)$ to represent the entropy function and $I(\cdot;\cdot)$ the mutual information function.

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 ν versions, and Server $c+i$ has versions $[\nu-i]$, for $i \in [\nu-1]$. That is, we consider for servers $[c+\nu-1]$ the states

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

where $\mathbf{S}(i)$ represents the state of the ith server. Since the code should work for any distribution for each version W_i over $[M]$, we consider the case when it is uniformly distributed. That is, we focus on the case when $H(W_i) = \log_2 M = B$.

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 ν versions are independent and of size $\log_2 M = B$: for $i \in [\nu]$, $Q \subseteq [\nu]\backslash\{i\}$,

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

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

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

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

$$H(W_i|X_Q, X_{c+\nu-i}) = 0. \tag{5}$$

Before proving the main result, we first prove two lemmas about some mutual information.

Lemma 1 *The following is satisfied when $\nu = 2$,*

$$I(X_{[c-1]}; W_1) \geq B - \alpha. \tag{6}$$

Proof: The left hand side is

$$\begin{aligned}
&I(X_{[c-1]}; W_1) \\
=&I(X_{[c-1]}, X_{c+1}; W_1) - I(X_{c+1}; W_1|X_{[c-1]}) && (7)\\
=&H(W_1) - H(W_1|X_{[c-1]\cup\{c+1\}}) - I(X_{c+1}; W_1|X_{[c-1]}) \\
=&B - 0 - H(X_{c+1}|X_{[c-1]}) + H(X_{c+1}|X_{[c-1]}, W_1) && (8)\\
\geq&B - H(X_{c+1}) + 0 && (9)\\
\geq&B - \alpha. && (10)
\end{aligned}$$

Here (7) follows from chain rule, (8) follows from (3)(5), (9) follows from (4) and the fact that condition reduces entropy, and (10) follows from (2). ■

Notice that in (6) α can be interpreted as the amount of information stored at Server $c+1$, and B as the amount of information of Version 1. Intuitively, we can understand (6) like this: the information stored at servers in $[c-1]$ combined with the information at Server $c+1$ 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 when $\nu = 2$,*

$$\sum_{i=1}^{c} H(X_{[c]\backslash\{i\}}|W_1) \geq (c-1)H(X_{[c]}|W_1).$$

Proof: Rewrite the expression as follows,

$$\begin{aligned}
&\sum_{i=1}^{c} H(X_{[c]\backslash\{i\}}|W_1) \\
=&\sum_{i=1}^{c-1} H(X_{[c]\backslash\{i\}}|W_1) + H(X_{[c-1]}|W_1) \\
=&\sum_{i=1}^{c-1} H(X_{[c]\backslash\{i\}}|W_1) + \sum_{i=1}^{c-1} H(X_i|X_{[i-1]}, W_1) \\
\geq&\sum_{i=1}^{c-1} H(X_{[c]\backslash\{i\}}|W_1) + \sum_{i=1}^{c-1} H(X_i|X_{[c]\backslash\{i\}}, W_1) \\
=&(c-1)H(X_{[c]}|W_1).
\end{aligned}$$

And the proof is completed. ■

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

Theorem 3 *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 $X_{[c-1]}$.

$$\begin{aligned}
(c-1)\alpha \geq H(X_{[c-1]}) &= I(X_{[c-1]}; W_1, W_2) \\
&= I(X_{[c-1]}; W_1) + I(X_{[c-1]}; W_2|W_1). && (11)
\end{aligned}$$

And by (4),

$$\begin{aligned}
I(X_{[c-1]}; W_2|W_1) =&H(X_{[c-1]}|W_1) - H(X_{[c-1]}|W_1, W_2) \\
=&H(X_{[c-1]}|W_1).
\end{aligned}$$

Moreover, we show next that the following holds: without loss of generality

$$H(X_{[c-1]}|W_1) \geq \frac{c-1}{c}B. \tag{12}$$

If so, combined with (6)(11)(12) our proof will be complete because

$$(c-1)\alpha \geq B - \alpha + \frac{c-1}{c}B.$$

State	1	2	3
Version 1	α		$\alpha - B/c$
Version 2		α	B/c

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 $\alpha = \frac{2c-1}{c^2}B$.

The inequality (12) holds because

$$\sum_{i=1}^{c} H(X_{[c]\backslash\{i\}}|W_1) \geq (c-1)H(X_{[c]}|W_1) \quad (13)$$
$$=(c-1)(H(X_{[c]}, W_1) - H(W_1))$$
$$=(c-1)(H(X_{[c]}, W_1, W_2) - H(W_2|X_{[c]}, W_1) - H(W_1))$$
$$=(c-1)(H(W_1, W_2) - 0 - H(W_1)) \quad (14)$$
$$=(c-1)H(W_2) = (c-1)B. \quad (15)$$

Here (13) follows from Lemma 2, (14) comes from (4)(5), and (15) 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 $1, \dots, c$. Therefore the bound follows. ∎

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., $\frac{\log q}{\log M} \leq \min\{1, \nu/c\}$, for $c \geq 2$. Again we use $\alpha = \log_2 q, B = \log_2 M$.

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 $\alpha = \frac{\nu c - \nu + 1}{c^2}B$, $\nu = 2, 3$. 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 $\varphi_S^{(1)} = \varphi_S^{(2)} = \cdots = \varphi_S^{(n)}$. 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 $\alpha = 3B/4, \alpha - B/c = B/4, B/c = B/2$. 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.

State	1	2	3	4	5	6	7
Version 1	α			α_1	α_1		α_2
Version 2		α		B/c		α_1	B/c
Version 3			α		B/c	B/c	B/c

TABLE V
Storage assignment for a code with $\nu = 3$. Blank entries indicate that the state does not include the version, e.g., State 4 contains Versions $1, 2$. The storage cost is $\alpha = \frac{3c-2}{c^2}B$.

The above construction is indeed a multi-version code, as Theorem 2 claims. And the proof can be found in the full version of the paper [11]. In fact, the result can be extended to arbitrary ν versions, and the constructed code has storage cost $\alpha = \frac{\nu c - \nu + 1}{c^2}B$.

Acknowledgment

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

This work is partially supported by the Center for Science of Information (CSoI), an NSF Science and Technology Center, under grant agreement CCF-0939370. The second author thanks Prof. Muriel Médard for providing financial support for his research.

References

[1] N. A. Lynch, *Distributed Algorithms*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1996.

[2] M. Herlihy and N. Shavit, *The Art of Multiprocessor Programming, Revised Reprint*. Elsevier, 2012.

[3] H. Attiya, A. Bar-Noy, and D. Dolev, "Sharing memory robustly in message-passing systems," *J. ACM*, vol. 42, no. 1, pp. 124–142, Jan. 1995.

[4] G. DeCandia, D. Hastorun, M. Jampani, 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.

[5] J. Hendricks, G. R. Ganger, and M. K. Reiter, "Low-overhead byzantine fault-tolerant storage," *ACM SIGOPS Operating Systems Review*, vol. 41, no. 6, pp. 73–86, 2007.

[6] P. Dutta, R. Guerraoui, and R. R. Levy, "Optimistic erasure-coded distributed storage," in *Distributed Computing*. Springer, 2008, pp. 182–196.

[7] V. R. Cadambe, N. Lynch, M. Medard, and P. Musial, "Coded emulation of shared atomic memory for message passing architectures," 2013, http://hdl.handle.net/1721.1/79606.

[8] D. Dobre, G. Karame, W. Li, M. Majuntke, N. Suri, and M. Vukolić, "PoWerStore: proofs of writing for efficient and robust storage," in *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*. ACM, 2013, pp. 285–298.

[9] P. Gopalan, C. Huang, H. Simitci, and S. Yekhanin, "On the locality of codeword symbols," *IEEE Transactions on Information Theory*, vol. 58, no. 11, pp. 6925 –6934, nov. 2012.

[10] D. S. Papailiopoulos and A. G. Dimakis, "Locally repairable codes," in *Proceedings of 2012 IEEE International Symposium on Information Theory (ISIT)*. IEEE, 2012, pp. 2771–2775.

[11] Z. Wang and V. Cadambe, "Multi-version coding in distributed storage," available at the authors' website.