On Multi-version Coding for Distributed Storage

Zhiying Wang
Center for Science of Information,
Stanford University
zhiyingw@stanford.edu

Viveck R. Cadambe
Department of Electrical Engineering,
Pennsylvania State University
viveck@engr.psu.edu

***Abstract*—The multi-version coding problem described previously by Wang and Cadambe is motivated by applications to distributed storage and computing. Here, we consider a modification to the previously described multi-version coding problem that retains the essence of the earlier definition, and show that our modification leads to a reduced storage cost. We consider a setting where there are n servers that aim to store ν versions of a message, where there is a total ordering on the versions from the earliest to the latest. We assume that each message version has size $\log_2 M$ bits. Each server can receive any subset of the ν versions and stores over an alphabet of size q a function of the message versions it receives. The (n, c, ν, M, q) multi-version code we consider ensures that, a decoder that connects to any c of the n servers can recover the message corresponding to the latest common version stored among those servers, or a message corresponding to a version that is later than the latest common version. Unlike our earlier paper, we allow for the message version that is decoded to be one that is later than the latest common version. Through an achievable scheme and a tight converse, we describe the optimal multi-version code for $\nu = 2$ versions from the perspective of the storage cost $\frac{\log_2 q}{\log_2 M}$. In particular, we show that for $\nu = 2$, the optimal multi-version code has a storage cost of $\frac{2}{c+1}$ when c is odd and $\frac{2(c+1)}{c(c+2)}$ when c is even. We also present achievable code constructions for arbitrary values of the parameter ν.**

I. Introduction

In this paper, we consider a modification to the *multi-version coding problem* presented in [1]. Consider storing a message of size $\log_2 M$ bits in a distributed storage system with n server nodes, where the storage capacity of each server is $\log_2 q$ bits. To accommodate for the possibility that the message changes with time, assume that it has two versions, the *old* version and the *new* version. We are interested in a coding scheme that allows a client to connect to c of the n servers and decode the *latest common version* among the c servers, or a version that is later than the latest common version. To begin with, assume that the old version is dispersed to all the servers, and every server contains at most $\log_2 q$ bits of an encoded form of the old version. We require that from any c servers, the old version is recoverable in this state (see Figure 1 (a)). Then the new version is dispersed to the servers, and due to network failures or delays, only a subset of the servers get the new version. Every server contains a total of at most $\log_2 q$ bits, which are obtained by encoding its received message versions. In this new state, we require that from any c servers that received the new version, the client should recover the new version which is the latest common version among the c servers (see Figure 1 (b)). If any of the c servers did not receive the new version, the latest common version among the c servers is the old version, and the client should decode either the old or the new version (see Figure 1 (c)). We measure the performance of the code by the *storage cost* defined as $\frac{\log_2 q}{\log_2 M}$.

A *multi-version code* is a generalization of the above setting to accommodate an arbitrary number $\nu \geq 2$ of versions. Multi-version coding promises to have many applications in distributed storage and computing. Specifically, it could be useful for shared memory emulation [2]–[8], which in turn has applications in multiprocessor programming [9] and as key-value stores in distributed storage systems [4]. For a more detailed description of the motivation behind our formulation, we refer the readers to [1].

Our description of the multi-version code here is a modification of the definition in reference [1], where the latest common version was forced to be recoverable. That is, reference [1] omitted the possibility that a message version which is later than the latest common version can be recovered by the client. For example, in Figure 1 (c) the code in [1] only recovers the old version. We note that our modified definition does not alter the spirit of the multi-version code definition of [1], and is relevant to all its potential applications. Interestingly, however, we show here that our modification leads to significant savings in terms of storage cost.

Two straight-forward solutions to our problem are as follows: (i) replication, where every server stores $\log_2 M$ bits of its latest version, and any c servers suffice to recover the latest version among the servers; and (ii) simple MDS code, where every server stores $\log_2 M/c$ bits of every version it has, and the latest common version is recoverable. Here we present coding schemes that outperform both these solutions. It is worth noting that replication does not necessarily recover the latest common version since it sometimes returns a version that is later than the latest common version. Therefore replication does not fit in the definition of [1], but is a multi-version code as per the definition we consider

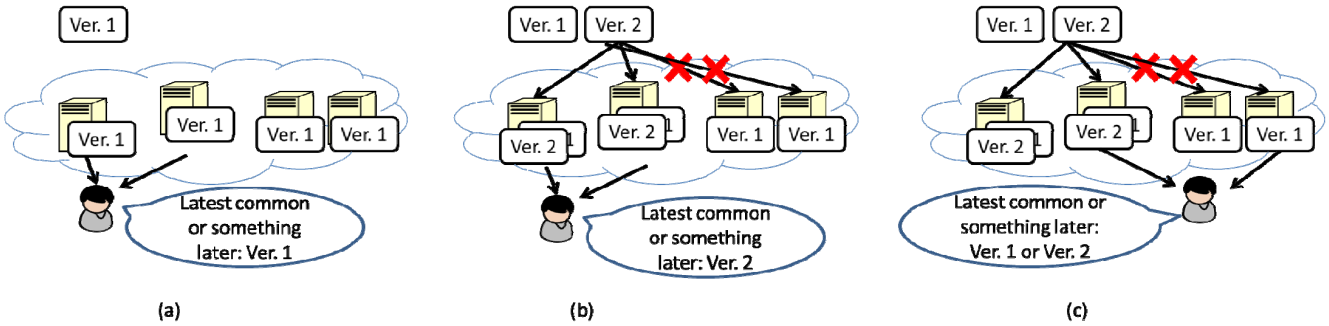

Fig. 1. Storing a file with 2 version in $n = 4$ nodes. From any $c = 2$ nodes, the code should recover the latest common version or something later. We denote the old and new versions as Version 1 and Version 2 respectively.

here.

The main contributions of the paper are: (i) an appropriate modification to the multi-version coding problem of [1], (ii) an information-theoretic lower bound for the storage cost of the multi-version code for all possible values of the parameters n, c, ν, q, M, and (iii) a code construction for arbitrary number of versions, which is optimal for $\nu = 2$ versions.

II. System Model and Main Results

A. The multi-version code

We begin with some notation. We write $[i] := \{1, 2, \ldots, i\}$, for integer $i \in \mathbb{N}^+$. 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.

We now define the multi-version coding problem, which is essentially a relaxation of the definition of [1]. We begin with an informal definition, and present the formal definition in Definition 1. 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. 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 an arbitrary subset of the 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. We refer to $\mathbf{S} = (\mathbf{S}(1), \ldots, \mathbf{S}(n)) \in \mathcal{P}([\nu])^n$ as the *state of the system*, where $\mathcal{P}([\nu])$ denotes the power set of $[\nu]$. 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, a message W_m should be decodable from the set T, where $m \geq \max \cap_{i \in T} \mathbf{S}(i)$ for every system state. 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 $\frac{\log_2 q}{\log_2 M}$ over all possible (n, c, ν, M, q) codes.

We present a formal definition next.

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

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

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

Definition 2 (Storage Cost of an (n, c, ν, M, q) multi-version code): The *storage cost* of an (n, c, ν, M, q) multi-version code is defined to be equal to $\frac{\log_q}{\log_2 M}$.

Note that replication, where the latest version is stored in every server, i.e., $\varphi_{\mathbf{S}_i}^{(i)}(W_{\mathbf{S}_i}) = W_{\max(\mathbf{S}(i))}$ incurs a storage cost of 1. An alternate strategy would be to separately encode every version using an MDS code of length n and dimension c, with each server storing an MDS codeword

symbol corresponding to every version that it has received. Such a coding scheme would achieve a storage cost of ν/c.

B. Main Results

Theorem 1: Given parameters (n, c, ν), there exists an (n, c, ν, M, q) multi-version code with a storage cost that is equal to

$$\max\left\{\frac{\nu t-\nu+1}{tc}, \frac{1}{t}\right\},$$

where

$$t=\begin{cases}\left\lceil\frac{c-1}{\nu}\right\rceil+1, & \text{if } c\geq(\nu-1)^2,\\ \left\lceil\frac{c}{\nu-1}\right\rceil, & \text{if } c<(\nu-1)^2.\end{cases}$$

Theorem 2: The storage cost of an (n, c, ν, M, q) multi-version code satisfies

$$\frac{\log_2 q}{\log_2 M}\geq\begin{cases}\frac{2}{c+1}, & \text{if } c \text{ is odd},\\ \frac{2(c+1)}{c(c+2)}, & \text{if } c \text{ is even}.\end{cases}$$

We make some remarks before proceeding.

Remark 1: Theorems 2 implies that the code construction that achieves the storage cost described in Theorem 1 is optimal from the perspective of storage cost if $\nu = 2$. The multi-version coding problem remains open for $\nu > 2$ version.

Remark 2: The achievable scheme of Theorem 1 strictly outperforms replication, simple MDS codes, and the scheme of [1].

Remark 3: It is worth noting that, under the relaxed definition of multi-version coding presented here, the converse of [1] is not applicable. In fact, the achievable scheme of Theorem 1 achieves a storage cost that is lower than the storage cost lower bound of [1] by exploiting the fact that a version that is later than the latest common version can be recovered.

III. Proof of Theorem 1 - Code Construction

We first describe our construction. After describing our construction, we show that our construction is a multi-version code in Theorem 3.

In our construction, each server encodes different versions separately. So that the total number of bits stored at a server is the sum of the storage costs of each of the versions it has in that state. The encoding strategy at the servers satisfies the following property: Suppose that Server i stores $\alpha_{i,v}\log_2 M$ bits of Version v, then Version v can be recovered from the c servers $i_1, i_2, ...i_c$, so long as

$$\sum_{j=1}^{c}\alpha_{i_j,v}\geq 1.$$

Note that such an encoding function can be found for a sufficiently large value of q using standard coding techniques[1].

[1]In fact, using random linear coding suffices to satisfy this property with non-zero probability, implying that there exists a coding scheme with the property.

	State $\{1,2\}$	State $\{1\}$	State $\{2\}$
Ver 1	$\alpha-1/t$	α	
Ver 2	$1/t$		$1/t$

TABLE I

Storage allocations for code construction with $\nu = 2$ versions. Note that $t=\lceil\frac{c-1}{2}\rceil+1$, and the storage size is $\alpha=\frac{2t-1}{tc}$. More specifically, $\alpha = 1/t$ for odd values of c and $\alpha=\frac{2(c+1)}{c(c+2)}$ for even values of c.

We also note that, in our approach, the storage allocation $\alpha_{i,v}$ only depends on the server state but not on the server index. Therefore, we can write $\alpha_{i,v}=\alpha_v^{(\mathbf{S}(i))}$ for any Server i at a nonempty state $\mathbf{S}(i)\subseteq[\nu]$.

To describe our construction, we only need to specify the parameter $\alpha_v^{(\mathbf{S})}$ for every possible server state $\mathbf{S}\subseteq[\nu]$ and every $v\in\mathbf{S}$, that is, we only need to specify the amount of Version v stored at a server in state $\mathbf{S}$. We denote $\alpha=\frac{\log_2 q}{\log_2 M}$. Note that we have $\alpha=\max_{\mathbf{S}\subseteq[\nu]}\sum_{v\in\mathbf{S}}\alpha_v^{(\mathbf{S})}$. In our description, for every possible system state, we partition our servers into $\nu+1$ groups. For $i\in[\nu]$, Group i has the set of servers which have Version i as the latest version. So, for instance, if $\nu = 2$, Group 1 has the servers in state $\{1\}$, and Group 2 contains the servers in states $\{2\}$ and $\{1, 2\}$.

Before giving the general construction, we start by describing our construction for the case of $\nu = 2$ versions. Define $t=\lceil\frac{c-1}{2}\rceil+1$, and construct code as in Table I. The storage size is $\alpha=\frac{2t-1}{tc}$. It is easy to check that the code works for odd values of c. In this case $\alpha = 1/t$, $t = (c+1)/2$, and a server in Group i stores $\frac{2}{c+1}$ of Version i. For any subset of c servers, note that we have at least $(c+1)/2$ servers in Group 1, or at least $(c+1)/2$ servers in Group 2. Because $\alpha=\frac{2}{c+1}$, in the former case Version 1 is recoverable, and in the latter case Version 2 is recoverable. Finally, we note that the latest common version or a version that is later than the latest common version is recovered using this strategy.

For even values of c, the validity of the construction can be verified as follows.

- If the latest common version is Version 2, then all the c servers are in Group 2. Since we have $c\geq t$ servers, and each server contains $1/t$ amount of Version 2, Version 2 is recoverable.
- If the latest common version is Version 1, then the c servers may be in state $\{1\}$ or state $\{1, 2\}$.
  If there are at least t servers in state $\{1, 2\}$, then we can recover Version 2. Otherwise, there are at most $t-1$ servers in state $\{1, 2\}$, and at least $c-t+1$ servers in state $\{1\}$. Thus the total amount of Version 1 in these servers is at least
  $$(c-t+1)\alpha+(t-1)(\alpha-1/t)=1,$$
  so we can recover Version 1.

Table II is an example with $c = 7, \nu = 3, \alpha = 1/3$. Notice in this case, each server only stores (a function of) the

	State $\{1\}$	State $\{2\}$	State $\{1,2\}$	State $\{3\}$	State $\{1,3\}$	State $\{2,3\}$	State $\{1,2,3\}$
Version 1	$1/3$		0		0		0
Version 2		$1/3$	$1/3$			0	0
Version 3				$1/3$	$1/3$	$1/3$	$1/3$

TABLE II
Storage allocations for code with $c=7, \nu=3, \alpha=1/3$.

latest version it receives, and does not store any information about any of the older versions. It is easy to see that when connected to $c=7$ servers with a common version, at least one version, say Version i, can be decoded from 3 servers in Group i.

To describe our general construction, we use an auxiliary parameter t defined as follows.

$$t=\begin{cases}\lceil\frac{c-1}{\nu}\rceil+1, & c>(\nu-1)^2,\\ \lceil\frac{c}{\nu-1}\rceil, & c\le(\nu-1)^2.\end{cases} \tag{1}$$

It is useful to note that if $c>(\nu-1)^2$, then $t\ge\nu$ and if $c\le(\nu-1)^2$, then $t<\nu$.

Construction 1: Given the above parameters, we construct the (n,c,ν,M,q) code with storage cost

$$\alpha=\frac{\log_2 q}{\log_2 M}=\max\left\{\frac{\nu t-\nu+1}{tc},\frac{1}{t}\right\}.$$

For state $\mathbf{S}$, the parameter $\alpha_v^{(\mathbf{S})}$ is set as follows:

- If Version j, $j\ge 2$, is the latest version in state $\mathbf{S}$, then $\alpha_j^{(\mathbf{S})}=\frac{1}{t}$, that is, store $\frac{1}{t}\log_2 M$ bits of Version j.
- If Version 1 is the latest version, namely $\mathbf{S}=\{1\}$, then $\alpha_1^{(\mathbf{S})}=\alpha$. That is, store $\alpha\log_2 M$ bits of Version 1.
- If Version 1 is not the latest version in state $\mathbf{S}$, and $\{1\}\in\mathbf{S}$, then, $\alpha_1^{(\mathbf{S})}=\alpha-\frac{1}{t}$, that is, store $\left(\alpha-\frac{1}{t}\right)\log_2 M$ bits of Version 1.

Note that in our construction, a server in Group j only stores encoded symbols of Version j and Version 1.

Remark 4: It can be easily verified that the storage cost of Construction 1 can be expressed more explicitly as follows:

$$\alpha=\begin{cases}\frac{1}{t}, & \text{if } c\in[(t-1)\nu+1,(\nu-1)t], t<\nu,\\ \frac{\nu t-\nu+1}{tc}, & \text{otherwise.}\end{cases}$$

where t is defined as in (1).

Remark 5: We note that when $\nu|(c-1)$, $t=\frac{c-1}{\nu}+1$ irrespective of whether c is bigger then $\nu-1$ or not. As a result, we have $\alpha=\frac{\nu}{c+\nu-1}$ when $\nu|(c-1)$.

In Figure 2 we show the storage size of the construction with $\nu=5$ versions, we can see the advantage of the proposed code compared to previous results.

Table III is an example for $t=3, c=5, \nu=3, \alpha=7/15$. In this example, the storage size of the states are not equal, but one can simply treat the worst-case size as α. One can check that the above code recovers the latest common version, or a version that is later than the latest common version. For example, suppose the latest common version is Version 1.

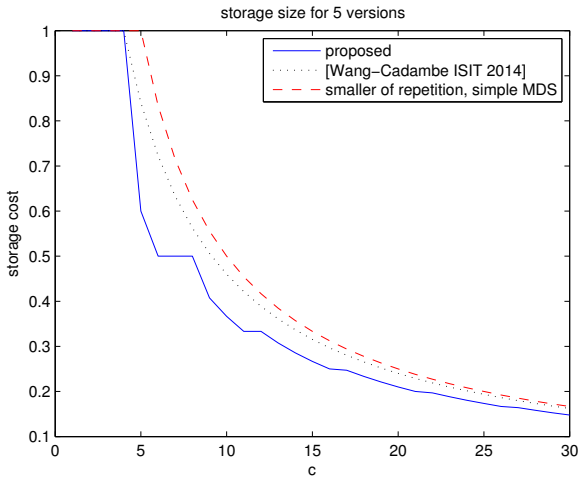

Fig. 2. Comparison between the construction for the code in Construction 1, the code in [1], and the smaller of replication and the simple MDS code. We fix $\nu=5$ versions and plot results for different number of connected servers, c.

- If at least three of the c servers are in Group 2, then Version 2 is recoverable.
- If at least three of the c servers are in Group 3, then Version 3 is decodable.
- Otherwise, among the c servers, at most two servers are in Group 2, at most two servers are in Group 3, and at least one server is in state $\{1\}$. The amount of information of Version 1 in these c servers is at least $7/15+2/15\times 4=1$, which implies that Version 1 is recoverable.

Theorem 3: The code in Construction 1 is a multi-version code.

Proof: To show a version is recoverable, it suffices to show that the total storage allocation for that version in the connected servers is at least 1. Let j be the latest common version among c servers. Note that there are at most $\nu-j+1$ groups, since Group 1, Group 2, ..., Group $j-1$ are empty.

- When $j\ge 2$, there exists a group, say Group k, with at least $\lceil\frac{c}{\nu-j+1}\rceil\ge\lceil\frac{c}{\nu-1}\rceil$ servers. In our construction, each server in Group k stores $\frac{1}{t}$ of Version k. To prove the theorem, it suffices to show that $\lceil\frac{c}{\nu-1}\rceil\ge t$, since this implies that Version k is recoverable from the servers in Group k. When $c\le(\nu-1)^2$, then $\lceil\frac{c}{\nu-1}\rceil=t$.

	State $\{1\}$	State $\{2\}$	State $\{1,2\}$	State $\{3\}$	State $\{1,3\}$	State $\{2,3\}$	State $\{1,2,3\}$
Version 1	$7/15$		$2/15$		$2/15$		$2/15$
Version 2		$1/3$	$1/3$			0	0
Version 3				$1/3$	$1/3$	$1/3$	$1/3$

TABLE III
Server storage allocations for $c = 5, \nu = 3, \alpha = 7/15$.

Therefore, we need to show this for $c > (\nu-1)^2$. When $c > (\nu-1)^2$, we have $t = \lceil \frac{c-1}{\nu} \rceil + 1$. Therefore, we have $c \in [\nu(t-2)+2, \nu(t-1)+1]$. Notice also that $c \geq \nu$. These imply the following.

$$\begin{aligned}
&\lceil \frac{c}{\nu-1} \rceil \\
\geq &\lceil \frac{\nu(t-2)+2}{\nu-1} \rceil \\
= &\lceil t-1+\frac{t-\nu+1}{\nu-1} \rceil \\
\geq &\lceil t-1+\frac{1}{\nu-1} \rceil \\
= &t.
\end{aligned}$$

Therefore, the theorem is proved for the case where $j \geq 2$.

- When $j = 1$ is the latest common version, if Group i has at least t servers for any $2 \leq i \leq \nu$, then Version i is recoverable and therefore the theorem is proved. Otherwise, there are at most $t-1$ servers in Group i, for all $2 \leq i \leq \nu$, each of which stores $\alpha - \frac{1}{t}$ size of Version 1; and thus at least $c-(\nu-1)(t-1)$ servers in Group 1, each storing α size of Version 1. The total storage size for Version 1 in these servers is at least

$$(\alpha - \frac{1}{t})(\nu-1)(t-1) + \alpha(c-(\nu-1)(t-1)).$$

And by the choice of α, we know the above amount is at least 1. Therefore, Version 1 can be recovered.

We have thus completed the proof. ∎

We would like to mention here that in fact, the construction in this section is inspired by computer search for $\nu = 3$ and small values of c using integer linear programming.

IV. Converse: Proof of Theorem 2

From the definitions, note that an (n, c, ν, M, q) multi-version code is also an $(n, c, 2, M, q)$ multi-version code. Therefore, it suffices to prove the theorem for the special case where $\nu = 2$. Consider any $(n, c, 2, M, q)$ multi-version code, and consider the first c servers[2]. For any arbitrary state $\mathbf{S} \in \mathcal{P}([\nu])^n$, let $\mathbf{S}^{[c]}$ denote the projection of the state onto the first c servers. To prove the converse, we first make the following claim.

[2]We note that an arbitrary set of c servers can be considered for the converse. We consider the first c servers without loss of generality.

A. Proof of Theorem 2 when c is odd

We first consider the case where c is odd, whose converse proof is simpler. We begin with the following claim.

Claim 1: For any achievable $(n, c, 2, M, q)$ code, there are two states $\mathbf{S}_1, \mathbf{S}_2 \in \mathcal{P}([\nu])^n$ such that

- The c-length tuples $\mathbf{S}_1^{[c]}$ and $\mathbf{S}_2^{[c]}$ differ only in one element, that is, they differ only with respect to the state of one of the first c servers
- The decoding function $\psi_{\mathbf{S}_1}^{([c])}$ outputs the first version W_1, whereas the decoding function $\psi_{\mathbf{S}_2}^{([c])}$ outputs the second version W_2.

In other words, from the perspective of the first c servers, states $\mathbf{S}_1$ and $\mathbf{S}_2$ differ only in the state of one server. Furthermore, W_1 is decodable from the symbols stored among the first c servers in state $\mathbf{S}_1$, and W_2 is decodable from the symbols stored in the first c symbols in state $\mathbf{S}_2$..

Proof of Claim 1: To prove Claim 1, consider the set of all states where every server has version 1. That is, consider the set of states $(\{\{1\},\{1,2\}\})^n$. Notice that for this set, every subset of c servers has Version 1 or Version 2 as the latest common version. This means that, for every state in this set, the corresponding decoding function must return Version 1 or Version 2. We partition this set of states into two partitions. The first partition $\mathcal{P}_1 \subset (\{\{1\},\{1,2\}\})^n$ is one where the decoding function returns the first version W_1 from the first c servers. The second partition $\mathcal{P}_2 \subset (\{\{1\},\{1,2\}\})^n$ is one where the decoding function returns the second version W_2 from the first c servers. Consider a state with the smallest number of occurrences of $\{1,2\}$ in partition $\mathcal{P}_2$ and denote this state as $\mathbf{S}_2$. In other words,

$$\mathbf{S}_2 = \arg\min_{\mathbf{S} \in \mathcal{P}_2} |\{i : \mathbf{S}(i) = \{1,2\}\}|.$$

Let $\mathbf{S}_1$ be a state obtained by replacing one of the occurrences of $\{1,2\}$ of $\mathbf{S}_2$ by $\{1\}$. Notice that, since the number of occurrences of $\{1,2\}$ in the state tuple $\mathbf{S}_1$ is smaller than the number occurrences of $\mathbf{S}_2$, the state $\mathbf{S}_1$ does not lie in $\mathcal{P}_2$. Furthermore state $\mathbf{S}_1$ lies in $(\{\{1\},\{1,2\}\})^n$. Therefore $\mathbf{S}_1$ lies in $\mathcal{P}_1$. It is easy to verify that states $\mathbf{S}_1$ and $\mathbf{S}_2$ satisfy the conditions of the claim.

Informal proof of Theorem 2 for odd c: Based on the conditions of Claim 1, Version 1 is decodable from the c servers in state $\mathbf{S}_1$, and Version 2 is decodable from the first c servers in state $\mathbf{S}_2$. But, notice that the encoded symbols of the servers $2, 3, \ldots, c$ are the same in both states $\mathbf{S}_1$ and $\mathbf{S}_2$. This implies that both Version 1 and Version 2 are decodable

from the following $c+1$ symbols: the c codeword symbols of first c servers in state $\mathbf{S}_1$, and the codeword symbol of the *first* server in state $\mathbf{S}_2$. Since 2 versions, each of alphabet size M are decodable from $c+1$ symbols from an alphabet of size q, we need that $(c+1)\log_2 q \geq 2\log_2 M$. This gives the bound for odd c.

Formal proof of Theorem 2 for odd c: Let $\mathbf{S}_1$ and $\mathbf{S}_2$ be two states that satisfy Claim 1. Notice that among the first c servers, $\mathbf{S}_1$ and $\mathbf{S}_2$ differ with respect to the state of only 1 server. Without loss of generality, assume that $\mathbf{S}_1(1) \neq \mathbf{S}_2(1)$. Therefore $\mathbf{S}_1(j) = \mathbf{S}_2(j), j = 2, 3, \ldots, c$. For W_1, W_2, let $X_i = \varphi^{(i)}_{\mathbf{S}_1}(W_{\mathbf{S}_2(i)})$ and $Y = \varphi^{(1)}_{\mathbf{S}_2}(W_{\mathbf{S}_2(1)})$, where φ_S represents the encoding functions of the $(n, c, 2, M, q)$ multi-version code. By the system model, W_1, W_2 are independent. Since the code should work for any distribution for each version W_i over $[M]$, we consider the case when W_1, W_2 are each uniformly distributed over $[M]$. Denoting the binary entropy function by $H(.)$, note that

$$H(W_1|X_1, X_2, \ldots, X_c) = 0 = H(W_2|Y, X_2, \ldots, X_c).$$

The above equality implies that

$$\begin{aligned}
& H(W_1, W_2|Y, X_{[c]}) && = 0. \\
\Rightarrow \quad & H(Y, X_{[c]}) && \overset{(a)}{=} H(W_1, W_2) \\
& && = 2\log_2 M. \\
\Rightarrow \quad & H(Y) + \textstyle\sum_{i=1}^{c} H(X_{[i]}) && \geq 2\log_2 M. \\
\Rightarrow \quad & \max(H(Y), H(X_1), \ldots, H(X_c)) && \geq \tfrac{2\log_2 M}{c+1}. \\
\Rightarrow \quad & \log_2 q && \geq \tfrac{2\log_2 M}{c+1}.
\end{aligned}$$

Here (a) follows from the fact that $H(Y, X_{[c]}|W_1, W_2) = 0$, and the final inequality follows because $\log_2 q \geq H(X_i)$ for all $i \in [c]$ and $\log_2 q \geq H(Y)$ for any distribution on the random variables $X_{[c]}, Y$. The above argument completes the proof for odd c.

B. Proof of Theorem 2 when c is even

Notice that the proof of the lower bound for the case where c is odd did not use the fact that c is odd. The argument is valid for the case where c is even as well. Therefore, we automatically have

$$\frac{\log_2 q}{\log_2 M} \geq \frac{2}{c+1}.$$

We aim to prove the stronger bound stated in the theorem for the case where c is even. We begin by strengthening Claim 1. As in the proof of Claim 1, we partition $(\{\{1\}, \{1, 2\}\})^n$ into two partitions $\mathcal{P}_1$ and $\mathcal{P}_2$. The partition $\mathcal{P}_1$ contains the set of states where the decoding function $\psi^{[c]}_{\mathbf{S}}$ returns the first version W_1 from the first c servers. Partition $\mathcal{P}_2$ contains the set of states where the decoding function on the first c servers returns the second version. Now, we make the following claim.

Claim 2: For any (n, c, ν, M, q) multi-version code if $(\{\{1\}, \{1, 2\}\})^n$ is partitioned into two partitions $\mathcal{P}_1, \mathcal{P}_2$ as described above, and

$$a = \min_{\mathbf{S} \in \mathcal{P}_2} |\{i : \mathbf{S}(i) = \{1, 2\}, i \in [c]\}|,$$

we have

(i) $\log_2 q \geq \left(2 - \frac{1}{a}\right) \frac{\log_2 M}{c}$,

(ii) $a \geq \frac{\log_2 M}{\log_2 q}$.

The desired storage bound simply follows from the above claim. In particular, suppose as a contradiction, we have

$$\frac{\log_2 q}{\log_2 M} < \frac{2(c+1)}{c(c+2)}. \tag{2}$$

Then, combining (2) with (i) of Claim 2, we have $a < (c+2)/2$. Now, since a is an integer, this implies that

$$a \leq c/2. \tag{3}$$

However, combining (2) with (ii) in Claim 2, we have

$$a \geq \frac{\log_2 M}{\log_2 q} > \frac{c(c+2)}{2(c+1)} > c/2,$$

which contradicts with (3). Therefore, we cannot have $\frac{\log_2 q}{\log_2 M} < \frac{2(c+1)}{c(c+2)}$.

We now aim to prove Claim 2.

Let

$$\mathbf{S}_2 = \arg\min_{\mathbf{S} \in \mathcal{P}_2} |\{i : \mathbf{S}(i) = \{1, 2\}\}|.$$

Now, note that in state $\mathbf{S}_2$, a of the first c servers are in states $\{1, 2\}$, and the remaining $c - a$ servers are in state $\{1\}$. Without loss of generality, assume that the first a servers are in state $\{1, 2\}$, that is, the codeword symbols of the first a servers are functions of both W_1, W_2, and the codeword symbols of the remaining $c-a$ servers are functions of W_1 alone. Denote X_i to be the ith codeword symbol in state $\mathbf{S}_2$. Claim 2 follows from the following three lemmas.

Lemma 1: For any subset $P \subset [a]$, such that $|P| = a - 1$

$$I(X_P; W_1) \geq \log_2 M - (c - a + 1)\log_2 q.$$

Lemma 2 (Han [10]):

$$\sum_{P \subset [a], |P| = a-1} H(X_P|W_1) \geq (a-1)H(X_{[a]}|W_1).$$

The above lemma, also known as the subset entropy inequality, has been shown in [10].

Lemma 3:

$$H(X_{[a]}|W_1) = \log_2 M.$$

Proof of Lemma 1: For a given set P, define state $\mathbf{S}_1$ as follows:

$$\mathbf{S}_1(i) = \begin{cases} \mathbf{S}_2(i), & i \in P \text{ or } i \in \{a+1, a+2 \ldots, c\}, \\ \{1\}, & \text{otherwise.} \end{cases}$$

Note that with respect to the first c servers, states $\mathbf{S}_2$ and $\mathbf{S}_1$ differ only in the state of one server - the server in the

singleton set $[a]-P$. Now, note that among the first c servers, only $a-1$ servers are in states $\{1,2\}$ which means that $\mathbf{S}_1$ lies in partition $\mathcal{P}_1$. This implies that the first version W_1 is decodable from $X_P, X_{\{a+1,\ldots,c\}}$ and $\varphi^{(k)}_{\mathbf{S}_1(k)}(W_1)$, where $[a]-P=\{k\}$ and φ represents the encoding function of the multi-version code. This implies that

$$I(X_P, X_{\{a+1,a+2,\ldots,c\}}, \varphi^{(k)}_{\mathbf{S}_1(k)}(W_1); W_1) = \log_2 M.$$

$$\begin{aligned}
\Rightarrow \quad & I(X_P; W_1) \\
\overset{(b)}{=} \quad & \log_2 M - H(X_{\{a+1,a+2,\ldots,c\}}, \varphi^{(k)}_{\mathbf{S}_1(k)}(W_1)|X_P) \\
\geq \quad & \log_2 M - H(X_{\{a+1,a+2,\ldots,c\}}, \varphi^{(k)}_{\mathbf{S}_1(k)}(W_1)) \\
\geq \quad & \log_2 M - (c-a+1)\log_2 q,
\end{aligned}$$

where (b) follows because the servers in $\{a+1, a+2, \ldots, c\} \cup ([a]-P)$ are in state $\{1\}$, meaning that their codeword symbols are functions of W_1, therefore, $H(X_{\{a+1,a+2,\ldots,c\}}, X_{[a]-P}|W_1, X_P) = 0$.

Proof of Lemma 3: In state $\mathbf{S}_2$, the codeword symbols of the first a servers depend on both W_1 and W_2. The remaining $c-a$ codeword symbol of the first c servers depends only on W_1. Furthermore, note that W_2 is recoverable from the first c symbols because $\mathbf{S}_2$ is in partition $\mathcal{P}_2$. This implies that, the message W_2 must be decodable from $X_{[a]}$ and W_1, since W_1 can be used to reconstruct $X_{\{a+1,a+2,\ldots,c\}}$. Therefore, we have

$$\begin{aligned}
& H(W_2|X_{[a]}, W_1) &=& \ 0. \\
\Rightarrow \quad & H(W_2, W_1, X_{[a]}) &=& \ H(X_{[a]}, W_1). \\
\Rightarrow \quad & H(W_1, W_2) &\overset{(c)}{=}& \ H(W_1) + H(X_{[a]}|W_1). \\
\Rightarrow \quad & \log_2 M &=& \ H(X_{[a]}|W_1).
\end{aligned}$$

Here (c) follows from the fact that $H(X_{[a]}|W_1, W_2) = 0$. The last equality uses the fact that the two versions are independent.

Proof of Claim 2: Lemmas 1, 2 and 3 together imply that

$$\begin{aligned}
a(a-1)\log_2 q \quad \geq \quad & \sum_{P\subset[a],|P|=a-1} H(X_P) \\
= \quad & \sum_{P\subset[a],|P|=a-1} (I(X_P; W_1) + H(X_P|W_1)) \\
\geq \quad & a(\log_2 M - (c-a+1)\log_2 q) \\
& + (a-1)H(X_{[a]}|W_1) \\
= \quad & a(\log_2 M - (c-a+1)\log_2 q) \\
& + (a-1)\log_2 M. \\
\Rightarrow \log_2 q \quad \geq \quad & \left(2 - \frac{1}{a}\right)\frac{\log_2 M}{c}.
\end{aligned}$$

which proves (i) in Claim 2. The proof of (ii) follows simply from Lemma 3 as follows:

$$a\log_2 q \geq H(X_{[a]}|W_1) = \log_2 M,$$

which implies that $a \geq \frac{\log_2 M}{\log_2 q}$.

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. Viveck R. Cadambe acknowledges his startup fund from the Department of Electrical Engineering at the Pennsylvania State University

REFERENCES

[1] Z. Wang and V. Cadambe, "Multi-version coding in distributed storage," in *Information Theory (ISIT), 2014 IEEE International Symposium on*, pp. 871–875, IEEE, 2014.

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

[3] H. Attiya, A. Bar-Noy, and D. Dolev, "Sharing memory robustly in message-passing systems," *J. ACM*, vol. 42, 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, pp. 205–220, 2007.

[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*, pp. 182–196, Springer, 2008.

[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*, pp. 285–298, ACM, 2013.

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

[10] T. S. Han, "Nonnegative entropy measures of multivariate symmetric correlations," *Information and Control*, vol. 36, no. 2, pp. 133–156, 1978.