Near-Optimal Multi-Version Codes

Majid Khabbazian
Department of Electrical and Computer Engineering
University of Alberta, Edmonton, Canada
mkhabbazian@ualberta.ca

***Abstract*—Motivated by applications to distributed storage and computing, the multi-version coding problem was formulated by Wang and Cadambe in [4]. In this problem, a client sequently over time stores v independent versions of a message in a storage system with n server nodes. It is assumed that, a message version may not reach some servers, and that each server is unaware of what has been stored in other servers. The problem requires that any c servers must be able to reconstruct their latest common version. An extended multi-version problem introduced in [5] relaxes the above requirement by requiring any c servers to be able to reconstruct their latest common version or any version later than that. The objective in both the original and extended multi-version problem is to minimize the worst case storage cost.**

In this work, we propose codes for both the multi-version problem and its extension. For the original multi-version coding problem, we show that the storage cost of our proposed codes are near-optimal. For the extended multi-version coding problem, we show that the storage cost of our first algorithm is optimal when $v|c-1$. Our second proposed extended multi-version code shows that storage cost of strictly less than one is achievable even when v is 50% larger than c. This is interesting, as the storage cost of existing codes becomes one as soon as v becomes larger than c.

I. Introduction

We study the multi-version coding problem formulated by Wang and Cadambe [4], and its extension introduced in [5]. The problem was motivated by applications in distributed computing and storage (see [4], for a list of such applications). In the original multi-version problem, there is a distributed storage system with n servers, and v independent message versions. The informal description of the problem is as follows. Every time, a client uploads one version (starting with version 1) by connecting to the n servers. Because of, say network failures, a version may not reach all the servers. However, when a version is reached/received by a server, the server stores some information about that message version (not necessarily the whole message), and perhaps modifies the information stored about previous versions. For example, in a *replication strategy*, when a version reaches a sever, the server stores the whole version and deletes any version stored before.

Let c, $1 \leq c \leq n$ be an integer. Parametrized by c, the multi-version coding problem requires any set of c servers to have collectively enough information to reconstruct their latest common version. The parameter c is similar to the locality parameter in locally recoverable codes designed for distributed storage systems (see [2] and [3] for instances). The objective of the problem is to minimize the worst-case storage cost, informally defined as the size of server's storage divided by the size of message (assuming that all versions have the same size).

By the above definition, the storage cost of the simple replication strategy is one. When $c < v$, a better strategy than the replication strategy, as stated in [4], is to use an (n, c) MDS code for each version. Using this approach, the worst-case storage cost becomes $\frac{v}{c}$. Interestingly, it was shown that the cost of $\frac{v}{c}$ can be slightly reduced for $v = 2$, and $v = 3$, to $\frac{2c-1}{c^2}$, and $\frac{3c-2}{c^2}$, respectively [4]. The authors of [4] also proved a lower bound of $1 - (1 - \frac{1}{c})^v$ for the worst-case storage cost, hence concluded that when the number of versions v approaches infinity, the *replication strategy* is close to optimal. Their lower bound also indicates that for small values of v, MDS codes are almost optimal.

The main contributions of this work are:

- We derive a new lower bound for multi-version codes. The lower bound in [4] holds for $n > c + v - 1$, while our lower bounds holds for any $n \geq c + 1$. Also, our lower bound shows that the storage cost of multi-version codes is at least one as soon as v becomes larger than c. From the lower bound in [4] we can only conclude that the storage cost approaches one as the number of versions v approaches infinity.

- We propose two different multi-version coding algorithms. Compared to the first algorithm, the second algorithm has slightly lower storage cost, which is near-optimal by our derived lower bound. This answers an open question raised in [4] about codes for moderate values of v. It also answers another question raised in [4] on whether coding across versions may help to improve the storage cost.
- For the extended multi-version codes, we also propose two coding algorithms. When $v|c-1$, the storage cost of the first algorithm[1] matches the lower bound shown in [1] and [6]. Our second algorithm shows that storage cost of strictly less than one is achievable even when v is 50% larger than c.

The organization of this paper is as follows. In Section II, the problem is formally defined. In Section III we derive a new lower bound on the storage cost of multi-version codes. We also propose two algorithms that nearly match the lower bound. Section IV proposes two extended multi-version algorithms, and Section V concludes the paper.

II. System Model and Definitions

For $i, j \in \mathbb{N}^+$, $j \geq i$, let $[i]$ and $[i,j]$ denote the sets $\{1,2,3,\ldots,i\}$, and $\{i, i+1, \ldots, j\}$, respecitvely. Also, for a set of integers $S = \{s_1, \ldots, s_m\}$, we use X_S to denote $X_{s_1}, \ldots, X_{s_m}$.

In the problem setting, there are n servers, and v versions (versions $1, 2, \ldots, v$) of a message. The value of the ith, $i \in [v]$, version is denoted by $W_i \in [M]$. Each server can store a value from the set $[q]$. The set of versions reached/received by server i, $i \in [n]$, is called the state of server i and is denoted by $\mathbf{S}(i) \in [v]$. For example $\mathbf{S}(2) = \{2,4,5,7\}$ indicates that the second server has received versions $2, 4, 5,$ and 7.

As defined in [4] and [5], the multi-version and the extended multi-version codes can be expressed formally as follows.

Definition 1. *(Multi-version code [4]) An (n, c, v, M, q) multi-version code consists of*

- *encoding functions*

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

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

- *decoding functions*

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

for every set $\mathbf{S} \in \mathcal{P}([v])^n$, and set $T \subseteq [n]$ with $|T| = c$, where $\mathcal{P}([v])$ denotes the power set of $[v]$.

where the encoding and decoding functions satisfy

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

for every $W_{[v]} \in [M]^v$, where

$$T = \{t_1, t_2, \ldots, t_c\}, \quad t_1 < t_2 < \ldots < t_c$$

Definition 2. *(Extended multi-version code [5]) Similar encoding and decoding functions definitions as those in Definition 1. This time, the functions must satisfy*

$$\psi_{\mathbf{S}}^{(T)}\left(\varphi_{\mathbf{S}(t_1)}^{(t_1)}(W_{\mathbf{S}(t_1)}), \ldots, \varphi_{\mathbf{S}(t_c)}^{(t_c)}(W_{\mathbf{S}(t_c)})\right) = \begin{cases} W_m & \text{if } \cap_{i\in T}\mathbf{S}(i) \neq \emptyset \\ \emptyset, & \text{otherwise}, \end{cases}$$

for some $m \geq \max \cap_{i\in T}\mathbf{S}(i)$, and every $W_{[v]} \in [M]^v$, where

$$T = \{t_1, t_2, \ldots, t_c\}, \quad t_1 < t_2 < \ldots < t_c$$

The storage cost of an (n, c, v, M, q) (extended) multi-version code is defined to be equal to $\frac{\log q}{\log M}$. Given parameters (n, c, v), the objective of the (extended) multi-version coding problem is to find a code with minimal storage cost.

III. The Multi-Version Coding Problem

We present a lower bound on the storage cost multi-version codes. The lower bound shows that the storage cost will become at least one when v becomes larger than c. Therefore, for $v > c$, the storage cost of multi-version codes will not be better than that of the replication strategy[2]. We also propose two multi-version coding algorithms. The second algorithm has slightly better storage cost, which can be shown to be near-optimal by the lower bound derived.

A. Lower Bound

Proposition 1. *For $c \geq 2$, $2 \leq v \leq c+1$, and $n \geq c+1$, the storage cost of any (n, c, v, M, q) multi-version*

[1]This algorithm was proposed by this author in [1], and independently by Wang and Cadambe in [6].

[2]Note that replication strategy is an extended multi-version code.

code is at least $\frac{v}{c+1}$.
In particular, when $v = c+1$*, the storage cost will be at least one.*

Proof. Suppose the state of the ith server, $i \in [n]$, is

$$\mathbf{S}(i) = \begin{cases} [v]\backslash\{i\} & \text{if } i \leq v \\ [v] & \text{if } i > v. \end{cases}$$

Consider the set of servers $[c+1]$. For $i \in [v]$, let $T_i = [c+1]\backslash\{i\}$. We have $T_i \subset [c+1]$, and $|T_i| = c$. Also,

$$\forall i \in [v], \quad \max \cap_{j \in T_i} \mathbf{S}(j) = i.$$

In other words, the latest common version of servers in T_i is i. Therefore, $\psi_{\mathbf{S}}^{(T_i)}$ returns W_i, that is the ith version is decodable by servers in T_i. Thus, any version $i \in [v]$ is decodable by the set of servers in $[c+1]$, as for each version i, there is a subset of $[c+1]$ (i.e., T_i) that can decode that version. Intuitively, this implies that the set of servers $[c+1]$ must collectively have enough information about all the v versions. The v versions have $v \log M$ bits of information, assuming that their values are taken independently and uniformly at random from $[M]$. The maximum information that can be stored in servers $[c+1]$ is $(c+1)\log q$. Thus, we must have $(c+1)\log q \geq v \log M$, hence $\frac{\log q}{\log M} \geq \frac{v}{c+1}$. □

B. Near-Optimal Multi-Version Coding Algorithms

Following we describe two multi-version coding algorithms. The proposed algorithms have the following properties:

- The algorithms are simple in the sense that they do not code across versions.
- The proposed algorithms assure that, at each step of the process, the storage cost per server does not exceed the maximum storage cost.
- The information stored for one version does not need to increase when later versions arrive. This is an important practical constraint which is not captured by Definition 1.

Before delving into the descriptions of our algorithms, we would like to remark that multi-version codes that do not code across versions can be simply viewed as multi-version storage assignment algorithms. In other words, an algorithm that, upon reception of a new version $i \in [v]$, and given the current state of the server, determines how much storage should be used for version i, can be transferred to a multi-version code using MDS (maximum distance separable) codes such as Reed-Solomon codes. We briefly explain this for the first algorithm.

1) First Algorithm (Algorithm 1): Let us first explain the storage assignment. Later, we explain how that storage assignment can be transferred into a multi-version code.

In the first algorithm, upon receiving a version $i \in [v]$, a server node i) assigns $\frac{2B}{c+1}$ bits of storage to version i if $i < v$, and $\frac{B}{c}$ bits if $i = v$; ii) reduces the storage assignment of version $i-1$ (if version $i-1$ was received before) from $\frac{2B}{c+1}$ to $\frac{B}{c+1}$. Let $\mathcal{G}(i)$ denote the amount of storage in bits assigned to a received version $i \in [v]$ by the first algorithm. By the above description, we have

$$\mathcal{G}(i) = \begin{cases} \frac{B}{c} & \text{if } i = v; \\ \frac{B}{c+1} & \text{if } i < v, \text{ and version } i+1 \text{ was received;} \\ \frac{2B}{c+1} & \text{if } i < v, \text{ and version } i+1 \text{ was not received.} \end{cases}$$

Now to transfer the storage assignment by the first algorithm to a multi-version coding solution we can use a $(2n, c+1)$ MDS code for versions $i \in [v-1]$, and a (n, c) MDS code for version v. To do this, a version $i \in [v-1]$ is divided into $c+1$ blocks (the version v is divided into c blocks). Using MDS codes, the $c+1$ blocks are then coded into $2n$ blocks for versions $i \in [v-1]$ (the c blocks of version v are coded into n blocks). Upon reception of version $i \in [v-1]$ the server stores two coded blocks. If version $i+1$ is received later, one of the two coded blocks of version i is deleted. For version v, the server stores only a single block when the version is received. Now, by the property of MDS codes, a version is recoverable by a set of servers if the sum of the storages assigned to that version over the set of servers is at least equal to the size of the version (as this translates to having enough number of blocks, that is $c+1$ for version $i \in [v-1]$, and c for version $i = v$).

Proposition 2. *Algorithm 1 is a multi-version code with storage cost of* $\frac{v}{c+1} + \frac{1}{c(c+1)}$.

Proof. Consider any set $T \subseteq [n]$ of c servers. Let $t \in [v]$ be their latest common version. To prove that version v can be decoded by servers in T, it suffices to show that the sum of storages assigned to version t over the servers in T is at least B. If $t = v$, the the sum of storages will be $c \cdot \frac{B}{c} = B$, as each server stores a coded block of size $\frac{B}{c}$ of version v. Equivalently, we can say that there are c coded blocks of the (n, c) MDS code, hence version v is decodable. Now, suppose $t < v$. In this case, every server has stored at least one coded block (of size $\frac{B}{c+1}$) of version v. Also, since

t is the latest common version, there must be server $u \in S$ that has not received version $t+1$. By the description of Algorithm 1, server u has stored two coded blocks for version t. Therefore, the number of coded blocks of version t stored across servers in T is at least $2+(c-1) = c+1$, hence version t is decodable (as we have $c+1$ coded blocks of a $(2n, c+1)$ MDS code).

Finally, the sum of storages assigned to different versions in a server is at most

$$\sum_{i\in[v]} \mathcal{G}_i \leq (v-1)\frac{B}{c+1} + \frac{B}{c}. \tag{1}$$

To see why (1) holds, one can change storage assignments as follows: For every $i \leq v-1$, if $\mathcal{G}_i = \frac{2B}{c+1}$ (which means $\mathcal{G}_{i+1} = 0$). change $\mathcal{G}_i$ to $\frac{B}{c+1}$, and $\mathcal{G}_{i+1}$ to $\frac{B}{c+1}$ if $i+1 < v$, and to $\frac{B}{c}$, otherwise. The following changes, do not reduce the sum of storage assignments. After applying the above change, the storage assignment for each version $i = [v-1]$ is either $\frac{B}{c+1}$ or zero. The storage assignment of version v does not change, hence it is either $\frac{B}{c}$ or zero.

By (1), the storage cost of the code is

$$\frac{(v-1)\frac{B}{c+1} + \frac{B}{c}}{B} = \frac{v-1}{c+1} + \frac{1}{c} = \frac{v}{c+1} + \frac{1}{c(c+1)}.$$

Recall that the minimum storage cost by Proposition 1 is $\frac{v}{c+1}$. □

2) Second Algorithm (Algorithm 2): The second algorithm slightly improves the storage cost of the first algorithm to

$$\frac{vc-(v-1)}{c^2} = \frac{v}{c+1} + \frac{c-v+1}{c^2(c+1)}.$$

For Algorithm 2, we just explain how storage is assigned to each version in a server. As before, using MDS codes, we can easily guarantee that a version is decodable by a set of servers as long as the sum of storages assigned to that version across the set of servers is at least B bits.

The storage assignment of the second algorithm works simply as follows. After receiving the first version, a server stores $\frac{vc-(v-1)}{c^2}B$ bits of information for the first version. When another version is received, the server deletes $\frac{B}{c}$ bits of information of the first version (if it was received), and stores $\frac{B}{c}$ bits of information of the version received.

Proposition 3. *Algorithm 2 is a multi-version code with storage cost of* $\frac{vc-(v-1)}{c^2}$.

Proof. By its description, the storage cost of Algorithm 2 is clearly $\frac{\frac{vc-(v-1)}{c^2}B}{B} = \frac{vc-(v-1)}{c^2}$.

Consider any set $T \subseteq [n]$ of c servers. Let $t \in [v]$ be their latest common version. If $t > 1$, then each server has $\frac{B}{c}$ bits of information of version t, hence the latest version can be decoded. If $t = 1$, the minimum storage cost assigned to the first version by a server u is

$$\frac{vc-(v-1)}{c^2}B - (v-1)\frac{B}{c} = \frac{c-(v-1)}{c^2}B,$$

which occurs when all the other versions are received by u. Note that, this cannot occur to all the c servers in T, as, otherwise, the first version will not be their latest common version. In fact, for each version $i > 1$, there must be server in T that has not received version i. By the description of Algorithm 2, that sever does not need to delete $\frac{B}{c}$ bits of information of the first version. Since this holds for $v-1$ versions $2, 3, \ldots, v$, the sum of storages assigned to the first version across servers in T is at least

$$c\left(\frac{c-(v-1)}{c^2}B\right) + (v-1)\frac{B}{c} = B,$$

hence the first version is decodable. □

IV. The Extended Multi-Version Coding Problem

In the original multi-version coding problem, the latest common version of any set of c servers should be decodable. This can be relaxed, as explained in [5], by requiring the latest common version or any later version to be decodable. In [5], it was shown that the storage cost of the extended multi-version coding problem is strictly less than that in the original problem. Recently, a lower bound of $\frac{c}{c+v-1}$ was proven in [6]. The same lower bound was attempted concurrently in [1]. In this work, we focus on constructing codes. Our first extended multi-version code (Algorithm 3) has optimal storage cost when $v|c-1$ by the lower bound proven in [6]. Our second code is proposed for the regime $v > c$, for which there is not any proposed code with storage cost less than one. Interestingly, using our second code (Algorithm 3), we show that storage costs of strictly less than one is possible for some values of $v > c$.

Our first multi-version code, Algorithm 3, works as follows. Each message version is divided into $\frac{1}{\lceil \frac{c}{v} \rceil}$ blocks. Then using an $(n, \frac{1}{\lceil \frac{c}{v} \rceil})$ MDS code, n coded blocks are generated. When receiving a version $i \in [v]$, a server deletes all the information stored about previous version, and stores a coded block of version i.

Proposition 4. *Algorithm 3 is an extended multi-version code with storage cost* $\frac{1}{\lceil \frac{c}{v} \rceil}$.

Proof. Algorithm 3 stores only one coded block of the latest version received, hence its storage cost is not more than $\frac{1}{\lceil \frac{c}{v} \rceil}$.

Now consider a set $T \in [n]$ of c servers, with some common latest version $t \geq 1$. Since every server in T has received at least one version in $[t, v]$, and each server has exactly one latest version received, by the extended pigeonhole principle, there must be version $t' \in [t, v]$ that is the latest received version in at least $\lceil \frac{c}{v-t+1} \rceil$ servers. Since we only need $\lceil \frac{c}{v} \rceil$ coded blocks to decode a version, version t' is decodable, which completes the proof as $t' \geq t$. □

Remark 1. *For* $v|c-1$*, we get* $c = vq + 1$ *for some non-negative integer* q*. In this case, the storage cost of the proposed code (Algorithm 3) is*

$$\frac{1}{\lceil \frac{c}{v} \rceil} = \frac{1}{q+1} = \frac{1}{\frac{c-1}{v}+1} = \frac{v}{c+v-1},$$

with matches the lower bound proven in [6].

Our second extended multi-version code (Algorithm 4) uses a $(2, 1)$ MDS code for versions $i \geq v-c$, and a (n, c) MDS code for versions $i \in [v-c-1]$. If the received version is i, $1 \leq i \leq v-c-1$, a server simply stores a coded block of version i. However, if the received version is i, $i \geq v-c$, in addition to storing a coded block of version i, the server deletes the coded blocks of any received version j, $v-c \leq j < i$. In other words, a server keeps only the coded block of the latest version among versions j, $v-c \leq j \leq v$. The coded block of any earlier version than $v-c$ is never deleted.

Proposition 5. *Algorithm 4 is an extended multi-version code with storage cost*

$$0.5 + \frac{v-c-1}{c}.$$

Proof. Each server, in the worst case, stores a single coded block of the $(2, 1)$ MDS code, and $v-c-1$ coded blocks of the (n, c) MDS code. Therefore, the total number of bits stored in a server is at most

$$\frac{B}{2} + (v-c-1)\frac{B}{c},$$

hence the code's storage cost is $0.5 + \frac{v-c-1}{c}$.

Let $T \in [n]$ be a set of c servers, with latest common version $t \in [v]$. If $t \leq v-c-1$, then each server in T must have a code block of version t, hence version t is decodable. Now suppose $t \geq v-c$. In this case, the latest received version of each server is in the set $[v-c, v]$, which consists of $c+1$ versions. Since there are c servers in T, by the pigeonhole principle, there must be two servers with the same latest common version t', $t' \geq t \geq v-c$. Version t' is threfore decoded. This completes the proof as $t' \geq t$. □

Remark 2. *Suppose* $v \leq 1.5c$*. Then the storage cost of Algorithm 4 is*

$$0.5 + \frac{v-c-1}{c} \leq 0.5 + \frac{1.5c-c-1}{c},$$

which is strictly less than one.

V. Conclusion

We studied the multi-version coding problem and its extension. For the multi-version coding problem, we proved a lower bound, and constructed two near-optimal multi-version codes. For the multi-version problem, we proposed to extended multi-version codes. We showed that, the storage cost of our first code is optimal when $v|c-1$. Using our second code, we showed that storage cost of strictly less than one is possible if v is at most $1.5c$. An interesting question is whether extended multi-version codes with storage cost of less than one exist for large values of v (for example, $v > 2c$).

Acknowledgements

The author would like to thank Dr. Zhiying Wan, Dr. Viveck Cadambe, Dr. Nancy Lynch, and Mohsen Ghaffari for insightful discussions. This work was partially supported by the Natural Sciences and Engineering Research Council of Canada.

References

[1] M. Khabbazian. Multi-version coding. In *arXiv:1503.06479 [cs.DC]*, March, 2015.

[2] D. S. Papailiopoulos and A. G. Dimakis. Locally repairable codes. In *IEEE International Symposium on Information Theory (ISIT)*, pages 2771–2775, 2012.

[3] I. Tamo and A. Barg. A family of optimal locally recoverable codes. *IEEE Transactions on Information Theory*, 60(8):4661–4676, 2014.

[4] Zhiying Wang and Viveck R. Cadambe. Multi-version coding in distributed storage. In *IEEE International Symposium on Information Theory (ISIT)*, pages 871–875, 2014.

[5] Zhiying Wang and Viveck R. Cadambe. On multi-version coding for distributed storage. In *Allerton Conference on Communication, Control, and Computing (Allerton)*, 2014.

[6] V. R. Cadambe Z. Wang. Multi-version coding in distributed storage. In *arXiv:1506.00684 [cs.IT]*, June, 2015.