The Pennsylvania State University

The Graduate School

HARNESSING DATA CORRELATION AND NETWORK INFORMATION IN DISTRIBUTED KEY-VALUE STORES

A Dissertation in

Electrical Engineering

by

Ramy E. Ali

Submitted in Partial Fulfillment

of the Requirements

for the Degree of

Doctor of Philosophy

May 2020

The dissertation of Ramy E. Ali was reviewed and approved by the following:

Viveck R. Cadambe
Assistant Professor of Electrical Engineering
Dissertation Advisor, Chair of Committee

David J. Miller
Professor of Electrical Engineering

Jing Yang
Assistant Professor of Electrical Engineering

Bhuvan Urgaonkar
Associate Professor of Computer Science

Jan Reimann
Associate Professor of Mathematics

Kultegin Aydin
Professor of Electrical Engineering
Head of the Department of Electrical Engineering

Abstract

Distributed key-value stores are an essential component of the cloud computing infrastructure and are used by several applications including reservation systems and financial transactions. These systems commonly replicate the data over multiple nodes to provide fast data access and to ensure the data availability despite the possible failures. In such systems, the data stored is updated frequently by the write clients, and the time scales of data updates and access are often comparable to the time scales of propagating the data to the storage nodes. Ensuring that a read client gets the most recent version of the data is challenging in these settings as the data updates may not reach all nodes. This notion of ensuring that the latest version of the data must be accessible despite the frequent data updates is known as consistency in distributed systems.

In replication-based distributed key-value stores, each node stores the entire data which results in a significant communication cost and storage cost as key-value stores use expensive high-speed memory to provide fast access to the data. This has motivated the use of erasure coding in key-value stores, where each node just stores only a part of the data to reduce the storage and communication costs. However, erasure-coded distributed key-value stores face unique challenges that do not appear in replication-based systems as a result of each node storing only a part

of the data, in the form of coded elements, and a read client has to obtain enough coded elements corresponding to the same version of the data. Specifically, when an erasure code is used in a strongly consistent key-value store, a storage node that receives a new version of the data cannot delete the older versions of the data and it has to effectively wait for a sufficient number of nodes to receive the new version before deleting the older versions of the data. The multi-version coding framework was formulated by Wang et al. to study the storage cost of strongly consistent distributed key-value stores from an information-theoretic point of view. Specifically, this work showed that the storage nodes have to store several versions of the data in strongly consistent decentralized erasure-coded storage systems, thereby offsetting some of the storage cost benefits of erasure coding.

Given that storing multiple versions of the data is inevitable in strongly consistent decentralized erasure-coded systems, exploiting the possible correlations between the different data versions is an important opportunity to reduce the storage cost in such systems. While the multi-version coding framework treats the different data versions as being independent, our work obtains storage cost savings by developing code construction that combine the benefits of erasure coding and delta coding to compress the differences between subsequent versions of the data and shows that the correlation can be exploited even in asynchronous decentralized consistent storage systems.

Next, motivated by real-world considerations of geo-distributed key-value stores where the system is not completely decentralized as the nodes can exchange data depending on the network topology, we study a system where a node acquires side information of which versions propagated to some other nodes based on the network

topology. Our code constructions show that the side information can result in a better storage cost as compared with the case where the nodes do not exchange side information considered in the multi-version coding framework for some regimes at the expense of the additional latency and the negligible communication overhead of exchanging the side information. Our impossibility results identify surprising scenarios where exchanging tremendous amount of side information does not reduce the storage cost beyond the storage cost of the case where the nodes do not exchange side information. Hence, our work indicates that a careful study of the network topology is necessary to exploit the side information and reduce the storage cost in strongly consistent erasure-coded key-value stores.

Finally, we focus on improving the latency of distributed key-value stores. In replication-based strict quorum systems, the quorum sizes are chosen to ensure that the write and read quorums intersect and this guarantees strong consistency. In order to have fast access to the data, many key-values stores allow non-strict (partial, sloppy or probabilistic) quorums that may not intersect. Hence, such systems do not guarantee strong consistency and only provide eventual consistency. We study the latency-consistency trade-off of such systems and derive a closed-form expression for the inconsistency probability for 3-way replication assuming exponential delays. We also extend our study to eventually consistent erasure-coded key-value stores, show how to characterize the inconsistency probability of such systems and provide examples showing that erasure coding can provide more flexibility to the latency-consistency trade-off as compared with replication-based systems. Our approach allows tuning the latency and consistency guarantees based on the expected network delays.

Table of Contents

Chapter 1

Appendix F

List of Figures

List of Tables

Acknowledgments

First and foremost, all praises and gratitude are due to Allah. He provided me with strength, patience and confidence to follow the right paths when it was not clear that they were indeed the right paths. I am also deeply indebted to my family for continuously supporting me throughout my life.

I would like to thank Dr. Viveck Cadambe for his help and Professors David J. Miller, Jing Yang, Bhuvan Urgaonkar and Jan Reimann for their valuable feedback. I would like also to thank Prof. Antonio Blanca for introducing me to the elegant world of Randomized Algorithms in his class. I spent a considerable time in my PhD at Bell Labs, where I developed some of the significant ideas presented in this dissertation. I thank Carl Nuzman, Math of Communications Department, for hosting me during Fall 2017 and Summer 2018. I would like to thank Bob Jopson and Jairo Esteban for making my time at Crawford Hill enjoyable. I would like also to thank Bilgehan Erman, Ejder Baştuğ and Bruce Cilli, IP Networking Department, End-to-End Networks and Service Automation Group, for hosting me during Summer 2019. This was the best internship I had in my life, I had a great time working with them and they showed me the real value of working as a team. Bilgehan has been a great collaborator and friend who spent a lot of time discussing networking and distributed systems aspects with me and helping me preparing for my presentations. It was a great pleasure too participating in the Unix 50's celebration and meeting Brian Kernighan, Alfred Aho and Douglas McIlroy.

I would like to thank Prof. Mona M. Hella, Rensselaer Polytechnic Institute (RPI), and Dr. Hany Elgala for their support. I thank Prof. Agung Julius and Prof. Jeffrey W. Banks for their great classes and my friend at RPI, Yasin Hasekioglu for his support. Before starting my PhD, I got my M.Sc. from Nile University, Egypt and I had the pleasure to work with Prof. Karim G. Seddik, Prof. Mohammed Nafie, Dr. Fadel Digham, Dr. Amr El-Keyi, Prof. Mohamed Nafie and Prof. Zhu Han. Prof. Karim G. Seddik has been a great advisor and friend who continues to support me in all stages of my life. I am thankful to Prof. Zhu Han who supported me while applying for PhD. He asked me to help younger researchers when I get established and I hope I can do this in the future. I would like also to thank Prof. Tamer ElBatt and Prof. Ahmed H. Zahran for their continuous help even after graduating. The best and most challenging classes I took in my life were at Nile University. Before that, I got my B. Sc. in Electrical Engineering from Alexandria University, Egypt. I thank Prof. Sherif Rabia and my high school teacher, Ahmed Radwan, for showing me the elegance of Mathematics at an early stage.

Finally, I would like to thank my friends at Penn State, Mohammed Tarek, Mahmoud Ashour, Youssef Khazbak and Ahmed Essawy for their support during this journey. Special thanks to Abdulelah Altamimi and Zaid Alrawi for constantly keeping me caffeinated. My friends Alaa Olleak and Mohammad Ibrahim made my transition to New Jersey smooth and I sincerely thank them for that. I also thank my lifetime friends Ahmed Tayel, Karim Nawar, Ahmed Abozaid and Moetazbellah Medhat Samy for their continuous support.

Funding Acknowledgment and Disclaimer: Part of this material is based upon work supported by the National Science Foundation (NSF) under Award No. CCF 1553248. The author was also supported by a Graduate Fellowship from Penn State University. Any opinions, findings, and conclusions or recommendations expressed in this dissertation are those of the author and do not necessarily reflect the views of the National Science Foundation or Penn State University.

Dedication

To my family

Chapter 1
Introduction and Contributions

Key-value stores are databases, where the data is stored as a collection of key-value pairs and different clients can update or get the value of the stored data. Such systems play an important role in many applications such as reservation systems and financial transactions. Modern key-value stores store the data in a distributed fashion by replicating the data across multiple nodes to make the data available and accessible with low latency despite the possible failures and the nodes that are too slow to respond (stragglers). Distributed key-value stores form an integral part of modern cloud computing infrastructure and enable storing large amounts of data and serving millions of users simultaneously. Owing to their utility, there are numerous commercial and open-source cloud-based key-value store implementations such as Amazon Dynamo [1], Voldemort [2], Redis [3] and Apache Cassandra [4]. In distributed key-value stores, the data is frequently updated and it is desirable to make the latest version of the data accessible by the different users. This notion of ensuring that the latest version of the data must be accessible despite the frequent data updates is known as consistency in distributed systems [5].

Replicating the data across many nodes in the storage system however entails significant storage and communication costs as a result of each node storing the entire data object. Motivated by the goal of efficiently using the high-speed expensive memory used in modern key-value stores and providing fast access to the data, we propose storage schemes based on erasure coding that significantly reduce the storage costs of these systems. In addition, since many key-value stores offer faster access to the data at the expense of only providing probabilistic consistency guarantees [6], we study the latency-consistency trade-off in replication-based and erasure-coded distributed key-value stores.

In this chapter, we provide a brief background about distributed key-value stores and summarize the contributions of this dissertation. The rest of this chapter is organized as follows. In Section 1.1, we provide a brief overview about distributed key-value stores. Section 1.2 provides a brief overview about replication-based key-value stores. In Section 1.3, we explain the challenges associated with using erasure coding in distributed key-value stores. Section 1.4 discusses the multi-version coding framework [7] which proposes code constructions and studies the fundamental limits of the storage costs of distributed key-value stores. Finally, Section 1.5 summarizes the main contributions of this dissertation.

1.1 Quorum-based Algorithms for Key-Value Stores

The design principles of key-value stores are rooted in the distributed computing-theoretic abstraction known as *shared memory emulation* [5]. The goal of the *read-write* shared memory emulation problem is to implement a read-write variable over a distributed system. While there has been interest in archival data storage systems in coding theory, e.g. [8, 9], the shared memory emulation set up differs in the following aspects.

1. *Asynchrony:* a new version of the data may not arrive at all servers simultaneously and at any given point of time, different servers may have received

different subsets of the data versions.

2. *Decentralized nature:* there is no single node that is aware of all versions of the data simultaneously, and a server is not aware of which versions are received by other servers in the system.

3. *Consistency:* a client accessing the storage system must be able to retrieve the latest possible version of the data.

Shared memory emulation algorithms use *quorum-based* techniques to deal with the asynchrony and the possible failures [10]. In quorum-based techniques, each write operation selects a set from a set of sets of servers, known as a quorum system. This set of nodes is known as the write quorum. Similarly, each read operation selects a set of nodes known as the read quorum from the quorum system. Specifically, in a system of n servers that tolerates f crash-stop server failures [5], a write operation sends a write request to all servers and waits for the acknowledgments from any $c_W \leq n - f$ servers for the operation to be considered *complete*. Similarly, a read operation sends a read request to all servers and waits for the response of any $c_R \leq n - f$ servers. This strategy ensures that, for every complete version, there are at least $\max(c_W + c_R - n, 0)$ servers that received that version and responded to the read request. Shared memory emulation protocols order the different writes usually using Lamport clocks [11] and require that the latest version that has been propagated to at least c_W nodes - that is, the version corresponding to latest *complete* write operation - or a later incomplete version must be obtained by a read operation. This requirement is known as strong consistency, *atomicity*, or *linearizability* in distributed systems [5].

While certain key-value stores can offer more complex data structures such as lists, tables, and corresponding operations such as appends, row and column-scans, studying simple key-value stores which offer simple write and read primitives to the external clients can help in understanding the more complex key-value stores.

1.2 Replication-based Key-value Stores

In quorum-based protocols that use replication such as the well-known ABD algorithm [12], strict quorum systems are used to ensure strong consistency. In a strict quorum system, any two quorums in the quorum system must overlap. Specifically, the quorum sizes c_W and c_R are chosen such that $c_W + c_R > n$ to ensure that the write and read quorums intersect [1, 5].

In replication-based eventually consistent key-value stores however, the quorum sizes c_W and c_R can be chosen such that $c_W + c_R \leq n$. Hence, the write and read quorums may not overlap in such systems. Such quorum systems are known as non-strict, partial or probabilistic quorum systems [13, 14]. In fact, many key-values stores including Amazon Dynamo [15], Apache Cassandra [16] and Riak [17] use probabilistic quorum systems to provide a lower write latency and/or read latency. Probabilistic quorum systems however only guarantee that the users will eventually return consistent data if there are no new write operations [18, 19].

1.3 Challenges of Erasure-coded Key-value Stores

There has been significant interest recently in developing erasure-coded key-value stores to obtain storage cost savings over replication-based algorithms. In fact, there is enormous interest in the distributed systems research community with an emerging body of work on the design of erasure-coded key-value stores prototypes [20–24]. For e.g, Microsoft Giza [21] uses erasure coding across 11 data centers in 3 continents for typical cloud storage workloads to reduce the storage cost at the expense of inter-data center traffic and shows that erasure coding can in fact hold its own even for production workloads in comparison with replication.

Erasure coding has also several benefits beside reducing the storage cost as demonstrated in [25] for instance which showed that erasure coding can be superior to replication for in-memory data stores when designed carefully. In geo-distributed settings, the inter-data center bandwidths are very expensive and erasure coding can be also beneficial over replication by allowing write operations to send only the encoded values as opposed to complete copies of the data to remote data centers. Similarly, the read operations can also be less expensive by allowing data centers to send only the encoded data.

The use of erasure coding in asynchronous distributed systems where strong consistency is required however leads to interesting algorithmic and coding challenges as there is no single server that stores the data entirely. If a maximum distance separable (MDS) code of dimension $k > 1$ is used for instance, each server only stores a fraction of $1/k$ of the entire value. Therefore, for a read operation to get a version of the data, at least k servers must send the codeword symbols corresponding to this version. As a consequence, when a write operation updates the data, a server cannot delete the symbol corresponding to the old version before symbols corresponding to a new version propagate to a sufficient number of servers. That is, servers cannot simply store the latest version they receive; they have to store older versions [26, 27] as shown in Fig. 1.1.

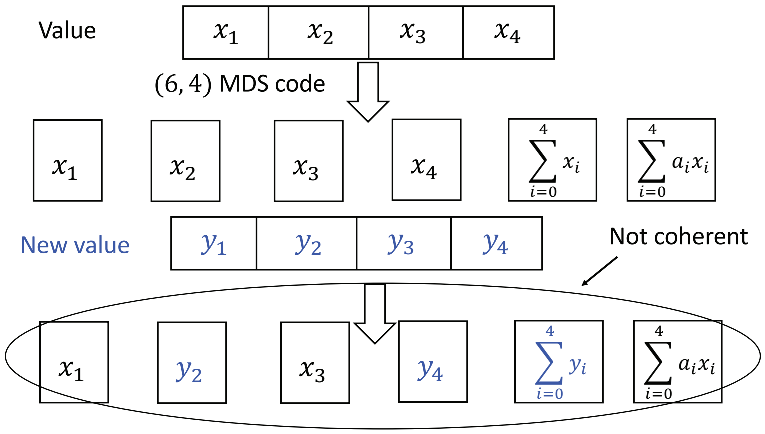

Value
x_1 x_2 x_3 x_4
(6, 4) MDS code
x_1 x_2 x_3 x_4 $\sum_{i=0}^{4} x_i$ $\sum_{i=0}^{4} a_i x_i$
New value
y_1 y_2 y_3 y_4
Not coherent
x_1 y_2 x_3 y_4 $\sum_{i=0}^{4} y_i$ $\sum_{i=0}^{4} a_i x_i$

Figure 1.1. An erasure-coded distributed system with $n = 6$, $c_W = 5$ and $c_R = 5$. The nodes only store the codeword symbols corresponding to the latest version. A value (x_1, x_2, x_3, x_4) is encoded using $(6, 4)$ maximum distance separable (MDS) code and each node stores a codeword symbol. The value is then updated to (y_1, y_2, y_3, y_4), but only three nodes have received the updated codeword symbols at this point in time due to the asynchrony in the system and the other three nodes store the old codeword symbols. At this point, the decoder cannot return neither the first version (x_1, x_2, x_3, x_4) nor the second version (y_1, y_2, y_3, y_4).

1.4 Multi-Version Coding

The *multi-version coding* problem [7] abstracts out algorithmic details of shared memory emulation while retaining the essence of strongly consistent storage systems. In fact, coding schemes for multi-version coding are used in protocols for key-value stores [28, 29]. Information-theoretic converses for multi-version coding have also led to algorithmic impossibility results that bound the storage costs [26].

The multi-version coding problem considers a decentralized storage system with n servers that tolerates f crash-stop failures, where the objective is storing a message (read-write variable or object) of length K bits with ν versions denoted by

$\mathbf{W}_1, \mathbf{W}_2, \cdots, \mathbf{W}_\nu$. The versions are totally ordered; versions with higher ordering are referred to as later versions, and lower ordering as earlier versions. Due to the inherent asynchrony in the system, the versions may not propagate to all the nodes. Specifically, each node receives an arbitrary subset of these ν versions referred to as the *state* of that node. Therefore, each server can be in any state out of 2^ν possible local states and the whole system can be in any of $2^{\nu n}$ possible systems states. The goal of the multi-version coding problem is to devise an encoding function with knowledge of only the local server state, such that, for each of the $2^{\nu n}$ system states where there may be a version that has propagated to at least c_W nodes, that version (or a later version) must be decodable from every set of c_R servers.

In [7], it was shown that there is a storage cost inevitable price for maintaining strong consistency in asynchronous decentralized storage systems. In multi-version coding, for any complete version, for any decoder, there are at least $c := c_W + c_R - n$ servers that have received that version and responded to the decoder. In the classical erasure coding model, where $\nu = 1$, the Singleton bound implies that the storage cost per server is at least

$$K/c.$$

However for $\nu > 1$, a server cannot simply store the codeword symbol corresponding to one version. In the case where the versions are independent, it was shown in [7] that the storage cost per server is at least

$$\frac{\nu}{c + \nu - 1} K - \Theta(1).$$

Since, for $\nu < c$, we have

$$\frac{\nu}{c + \nu - 1} \geq \frac{\nu}{2c},$$

and since the per server cost of storing a version is K/c, we may interpret the result as follows: when the data versions are independent, to compensate for the asynchrony and maintain consistency, a server has to store an amount of data that is, from a cost perspective, tantamount to at least $\nu/2$ versions, each stored using an MDS code of dimension c. In addition, a code construction was developed with storage cost of $\frac{\nu}{c+\nu-1}K$, for the case where $\nu|(c-1)$.

1.5 Dissertation Outline

In this work, we study the fundamental limits of strongly consistent read-write key-value stores and propose code constructions that reduce the storage cost of such systems in Chapter 2 and Chapter 3. We also study eventually consistent key-value stores and analyze the latency-consistency trade-off of such systems in Chapter 4 and Chapter 5. Specifically, the contributions of this work are as follow.

1.5.1 Harnessing Correlations in Erasure Coded Key-Value Stores

In Chapter 2, we extend the scope of the multi-version coding framework to the case where the different data versions are correlated. Since storing multiple versions is inevitable [7, 30, 31] in strongly consistent decentralized erasure-coded systems and that offsets some of the storage cost benefits of erasure coding, exploiting the possible correlations between the various data versions is crucial to reduce the storage cost. We develop an information-theoretic approach that combines the benefits of erasure coding and delta coding to compress the differences between subsequent versions of the data and obtains storage cost savings over the approach of [7] which treats the data versions as being independent.

Our approach enables significant storage cost savings as compared to replication-based schemes and erasure coding approaches that do not exploit correlations. We model the correlation between successive versions in terms of their binary representation. Given a version, we assume that the subsequent version is uniformly distributed in the Hamming ball of radius $\delta_K K$ centered around that given version, where $0 \leq \delta_K \leq 1$ denotes the correlation coefficient. We study the case where δ_K is not known a priori and develop a coding scheme based on Reed-Solomon code with a storage cost of $\frac{K}{c} + (\nu - 1)\delta_K K(\log K + o(\log K))$. This code obtains the $1/c$ *erasure coding gain* for the first version and stores every subsequent version using delta coding.

Moreover, we study the case where δ_K is known a priori and show the existence of a linear code with a storage cost of $\frac{K}{c} + \frac{\nu-1}{c} \log Vol(\delta_K K, K) + o(\log K)$ using a random coding argument, where $Vol(\delta_K K, K)$ is the volume of the Hamming Ball of radius $\delta_K K$. Interestingly, this scheme simultaneously obtains the gains of both erasure coding and delta coding for subsequent versions despite the asynchrony and the decentralized nature. In addition, we provide a lower bound on the storage cost showing that this achievable scheme is within a factor of 2 from the information-theoretic optimum in certain interesting regimes.

1.5.2 Erasure-Coded Key-Value Stores with Side Information

In Chapter 3, we extend the scope of the multi-version coding framework along a different direction motivated by real-world considerations of geo-distributed key-value stores where the system is not completely decentralized as the nodes can exchange data depending on the network topology, latency requirements and communication costs. The multi-version coding framework [7, 32] considered a completely decentralized asynchronous system where the servers are not aware of which versions are received by the other servers. We relax this assumption and study a system where a server acquires side information of the versions propagated to some other servers based on the network topology.

Our code constructions show that the side information result in a better storage cost as compared with the case where the servers do not exchange side information for some regimes at the expense of the additional latency of exchanging this side information. We also provide an information-theoretic lower bound on the storage cost in such settings and identify scenarios where exchanging side information does not reduce the storage cost beyond the completely decentralized setting. Finally, we demonstrate the potential storage cost savings and the storage-latency trade-off of our constructions through a case study over Amazon web services (AWS).

1.5.3 Latency-Consistency Trade-off of Replication-Based Probabilistic Key-value Stores

We focus on improving the latency of replication-based key-value stores in Chapter 4. In replication-based strict quorum systems, c_W and c_R are chosen such that $c_W + c_R > n$ to ensure strong consistency. In order to have fast access to the data, many key-values stores allow probabilistic quorums where $c_W + c_R \leq n$ to provide a lower latency at the expense of only providing only eventual consistency [19]. Eventual consistency only guarantees that eventually the clients will retrieve consistent data if there are no new write operations, but it does not specify how soon or how fast this will happen. Specifically, eventual consistency does not specify the probability of returning the value of latest complete write operation t units of time after it completes.

The probabilistically bounded staleness framework (PBS) [33, 34] studied the latency-consistency trade-off of 3-way replication-based probabilistic quorum systems through Monte Carlo event-based simulations as studying this problem analytically is challenging. In order to tune the consistency and latency guarantees based on the network delays using a Monte Carlo approach, Monte Carlo simulations need to be carried out for all possible network delays. In Chapter 4, we study the latency-consistency trade-off for such systems analytically and derive a simple closed-form expression for the inconsistency probability for 3-way replication in terms of the quorum sizes and the mean write and read delays. Our approach is simple, more tractable than the Monte Carlo approach and can be used to tune the latency and consistency guarantees in distributed key-value stores based on the mean values of the write and read delays.

1.5.4 Latency-Consistency Trade-off of Erasure-Coded Probabilistic Key-value Stores

In Chapter 5, we study the latency-consistency trade-off of simple erasure-coded key-value stores. Specifically, we consider simple erasure-coded key-value stores where each version of the data is encoded using an MDS code of dimension k. In erasure-coded strict quorum systems using an MDS code of dimension k, c_W and c_R must be chosen such that $c_W + c_R - n \geq k$. In simple erasure-coded probabilistic quorum systems, the quorum sizes are chosen such that $c_W + c_R - n < k$ to provide fast access to the data. We illustrate how to characterize the inconsistency probability of such systems. Moreover, we provide examples showing that erasure coding can provide a more flexible latency-consistency as compared with replication through tuning the dimension of the underlying erasure code k.

Finally, Chapter 6 summarizes this dissertation and point out to the possible future research directions. We note that this work is organized such that each chapter after this chapter is self-contained and can be understood without reading the earlier chapters.

Chapter 2
Harnessing Correlations in Erasure-Coded Key-Value Stores

In strongly consistent decentralized erasure-coded systems, storing multiple versions is inevitable [7,30,31]. However, storing older versions of the data offsets some of the storage cost benefits of erasure coding. Exploiting the possible correlations between the various data versions can provide significant storage cost savings, but it is challenging in asynchronous and decentralized setting. In this chapter, we develop an information-theoretic approach that combines the benefits of erasure coding and delta coding which relies on the idea of compressing differences between subsequent versions of the data. Our approach exploits correlations between subsequent updates and enable significant storage cost savings as compared to replication-based schemes and erasure coding approaches that do not exploit the possible data correlation.

2.1 Introduction and Contributions

In this work, we extend the scope of the multi-version coding problem [7] to the case where the different versions are correlated. Specifically, we consider a decentralized storage system with n servers storing ν possibly correlated versions of a message (object or read-write variable). We assume that each message version is K bits long, and model the correlation between successive versions in terms of the bit-strings that represent them. Given a version, we assume that the subsequent version is uniformly distributed in the Hamming ball of radius $\delta_K K$ centered around that given version. Hence, this version can be represented using $\log Vol(\delta_K K, K)$ bits, where $Vol(\delta_K K, K)$ is the volume of the Hamming Ball of radius $\delta_K K$. We derive three main results for this system.

1. We first study the case where δ_K is not known and propose a coding scheme based on Reed-Solomon code with a per-server storage cost of

   $$\frac{K}{c} + (\nu - 1)\delta_K K(\log K + o(\log K)).$$

   This scheme obtains the $1/c$ *erasure coding gain* for the first version and stores every subsequent version via delta coding with a cost of $\delta_K K(\log K + o(\log K))$ bits. Thus, this scheme is unable to simultaneously obtain the gains of both erasure and delta coding.

2. We then study the case where δ_K is known and show the existence of a code with a per-server storage cost of

   $$\frac{K}{c} + \frac{\nu - 1}{c}\log Vol(\delta_K K, K) + o(\log K)$$

   using on a random binning-based argument.

   From a cost viewpoint, this scheme is tantamount to storing one version using erasure coding with a cost of K/c and performing delta and erasure coding for the subsequent versions leading to a cost of $\frac{\log Vol(\delta_K K, K)}{c}$ bits per version. This scheme outperforms our first scheme as it simultaneously obtains the gains of both erasure coding and delta coding for subsequent versions. We also show the existence of linear codes that obtain this storage cost.

Note that a cost of $\frac{K}{c} + \frac{\nu-1}{c}\log Vol(\delta_K K, K) + o(\log K)$ bits is readily achievable in a setting, where every server receives all the versions, and each server is aware that the other servers have indeed received all the versions. In such a setting, each server can store a fraction of $1/c$ of the first version it receives using an MDS code of dimension c. For a new version, each server can store $1/c$ of the compressed difference between this version and the old version using an MDS code of dimension c. However, this scheme would fail in our setting because of the decentralized and asynchronous nature of our model. For instance, a server which receives versions 1 and 3 needs to compress version 3 with respect to version 1 and then encodes it, but a different server that receives only versions 2 and 3 needs to compute the increment of version 3 with respect to version 2 and then encodes it; from a decoder's viewpoint, the erasure-coded symbols stored at the two servers are not compatible. Furthermore, the decentralized nature implies that the server that receives versions 1 and 3 must store some data that would enable successful decoding no matter what versions are received by the other servers. Handling the decentralized and asynchronous nature while achieving both the erasure and the delta coding gain is our main technical contribution.

3. Finally, we extend the lower bound of [7] to the case of correlated versions and show our random binning scheme is within a factor 2 of the information-theoretic optimum in certain interesting regimes.

2.2 Related Work

The idea of exploiting the correlation between the different versions to efficiently update, store or exchange data has a rich history of study in network information theory [35]. In their classic work, Slepian and Wolf [36] studied the problem of compressing correlated distributed sources, where the objective is decoding the data of all sources. Linear code constructions that approach the Slepian-Wolf limits have been proposed in [37–41] and references therein. The problem of exploiting correlated information to design efficient storage and communication systems has been studied for several modern and emerging applications in recent times. Reference [42] considers simultaneous and sequential interactive data exchange problems between two users. By leveraging Slepian-Wolf and belief propagation techniques, practical schemes have been proposed that exploit the correlation to minimize the number of exchanged bits between the users.

Encoding incremental updates efficiently is the motivation of the delta compression used commonly in data storage. The notion of delta compression was refined in [43, 44] by modeling the data updates using the edit distance; in particular, these references developed schemes that synchronize a small number of edits between a client and a server efficiently. While we note that the edit distance is relevant to some applications such as collaborative text editing, we focus on the classical Hamming metric used more widely in coding theory for the sake of fundamental understanding. Our metric may also be useful for certain applications, for e.g., those that view the data as a table as in Apache Cassandra [4], and the writes update only few entries of the table.

Exploiting correlations to improve efficiency in *distributed* storage and caching settings has been of significant interest [45–51]. In [45] and [46], coding schemes were developed that use as input, the old and the new version of the data, and output a code that can be used to store both versions efficiently. Capacity-achieving *update-efficient* codes for binary symmetric and erasure channels were studied in [47, 48], where a small change in the message leads to a codeword which is close to the original codeword in Hamming distance.

In [49], the problem of minimizing the communication cost of updating a "stale" server that did not get an updated message, by downloading data from already updated servers, was studied and constructions and tight bounds were developed. We note that the problem of [49] has some common modeling elements with our approach. Specifically, there is a limited degree of asynchrony - a single update does not a reach a particular server. A side information problem is presented in [50], where the goal is to send an updated version to a remote entity that has as side information, an arbitrary linear transform of an old version. The reference shows that the optimal encoding function is related to a maximally recoverable subcode of the linear transform associated with the side information. The problem of [50] is of peripheral interest to some of the solutions of our paper, since we aim to store use codeword symbols of some versions as side information to store other versions.

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

Organization of the chapter

The rest of this chapter is organized as follows. Section 2.3 presents the multi-version coding problem and the main results of this chapter. In Section 2.4, we provide our code constructions. Section 2.5 provides a lower bound on the storage cost. Finally, conclusions are discussed in Section 2.6.

2.3 System Model and Background

We start with some notation. We use boldface for vectors, capital letters for random vectors and small letters for the values or the realizations of the random vectors. In the n-dimensional space over a finite field $\mathbb{F}_p$, the standard basis column vectors are denoted by $\{\mathbf{e}_1, \mathbf{e}_2, \cdots, \mathbf{e}_n\}$. We denote the Hamming weight of a vector $\mathbf{x}$ by $w_H(\mathbf{x})$ and the Hamming distance between any two vectors $\mathbf{x_1}$ and $\mathbf{x_2}$ by $d_H(\mathbf{x_1}, \mathbf{x_2})$. For a positive integer i, we denote by $[i]$ the set $\{1, 2, \cdots, i\}$. For any set of ordered indices $S = \{s_1, s_2, \cdots, s_{|S|}\} \subseteq \mathbb{Z}$, where $s_1 < s_2 < \cdots < s_{|S|}$, and for any ensemble of variables $\{X_i : i \in S\}$, the tuple $(X_{s_1}, X_{s_2}, \cdots, X_{s_{|S|}})$ is denoted by X_S. We use $\log(.)$ to denote the logarithm to the base 2 and $H(.)$ to denote the binary entropy function. We use the notation $[2^K]$ to denote the set of K-length binary strings. A *code* of length n and dimension k over alphabet $\mathcal{A}$ consists of an injective mapping $\mathcal{C} : \mathcal{A}^k \to \mathcal{A}^n$. When $\mathcal{A}$ is a finite field and the mapping $\mathcal{C}$ is linear, then the code is referred to as a *linear code*. We refer to a linear code $\mathcal{C}$ of length n and dimension k as an (n, k) code. An (n, k) linear code is called MDS if the mapping projected to *any* k output co-ordinates is invertible.

2.3.1 Multi-version Codes (MVCs)

We now present a variant of the multi-version coding problem [7], discussed in Ch. 1.4, where we model the correlation between the data versions and allow probabilistic decoding. We consider a distributed storage system with n servers that can tolerate f crash-stop server failures. The system stores ν possibly correlated versions of a message where $\mathbf{W}_u \in [2^K]$ is the u-th version, $u \in [\nu]$, and K is the message length in bits. The versions are assumed to be totally ordered, i.e., if $u > l$, $\mathbf{W}_u$ is interpreted as a *later* version with respect to $\mathbf{W}_l$. We assume that $\mathbf{W}_1 \rightarrow \mathbf{W}_2 \rightarrow \ldots \rightarrow \mathbf{W}_\nu$ form a Markov chain. $\mathbf{W}_1$ is uniformly distributed over the set of K length binary vectors. Given $\mathbf{W}_m, \mathbf{W}_{m+1}$ is uniformly distributed in a Hamming ball of radius $\delta_K K$ that is denoted by

$$B(\mathbf{W}_m, \delta_K K) = \{\mathbf{W} : d_H(\mathbf{W}, \mathbf{W}_m) \leq \delta_K K\}, \tag{2.1}$$

and the volume of the Hamming ball is given by

$$Vol(\delta_K K, K) = |B(\mathbf{W}_m, \delta_K K)| = \sum_{j=0}^{\delta_K K} \binom{K}{j}. \tag{2.2}$$

Given a correlation coefficient δ_K, we denote the set of possible tuples $(\mathbf{w}_1, \mathbf{w}_2, \cdots, \mathbf{w}_\nu)$ under our correlation model by A_{δ_K}. We provide the formal definition next.

Definition 1 (δ_K-possible Set of Tuples)**.** *The set A_{δ_K} of δ_K-possible set of tuples $(\mathbf{w}_{u_1}, \mathbf{w}_{u_2}, \cdots, \mathbf{w}_{u_L})$ is defined as follows*

$$A_{\delta_K}(\mathbf{W}_{u_1}, \mathbf{W}_{u_2}, \cdots, \mathbf{W}_{u_L}) = \{(\mathbf{w}_{u_1}, \mathbf{w}_{u_2}, \cdots, \mathbf{w}_{u_L}) : \mathbf{w}_{u_1} \in [2^K],$$

$$\mathbf{w}_{u_2} \in B(\mathbf{w}_{u_1}, (u_2 - u_1)\delta_K K), \mathbf{w}_{u_3} \in B(\mathbf{w}_{u_2}, (u_3 - u_2)\delta_K K), \cdots,$$
$$\mathbf{w}_{u_L} \in B(\mathbf{w}_{u_{L-1}}, (u_L - u_{L-1})\delta_K K)\}, \tag{2.3}$$

where $\{u_1, u_2, \cdots, u_L\} \subseteq [\nu]$ *such that* $u_1 < u_2 < \cdots < u_L$.

We omit the dependency on the messages and simply write A_{δ_K}, when it is clear from the context. Similarly, we can also define the set of possible tuples $\mathbf{w}_{F_1}$ given a particular tuple $\mathbf{w}_{F_2}$, $A_{\delta_K}(\mathbf{W}_{F_1}|\mathbf{w}_{F_2})$, where F_1, F_2 are two subsets of $[\nu]$.

Remark 1. *Unlike the case of the twin binary symmetric source, in our model, the correlation coefficient* δ_K *is a function of* K *in general and is not necessarily a constant. The more familiar expressions that involve entropies can be obtained when* δ_K *is equal to a constant* δ *using Stirling's inequality [52]. Specifically, for* $\delta < 1/2$, *we have*

$$KH(\delta) - o(K) \leq \log Vol(\delta K, K) \leq KH(\delta).$$

The i-th server receives an arbitrary subset of versions $\mathbf{S}(i) \subseteq [\nu]$ that denotes the *state* of that server. We denote the system state by $\mathbf{S} = \{\mathbf{S}(1), \mathbf{S}(2), \cdots, \mathbf{S}(n)\} \in \mathcal{P}([\nu])^n$, where $\mathcal{P}([\nu])$ is the power set of $[\nu]$. For the i-th server with state $\mathbf{S}(i) = \{s_1, s_2, \cdots, s_{|\mathbf{S}(i)|}\}$, where $s_1 < s_2 < \cdots < s_{|\mathbf{S}(i)|}$, the server stores a codeword symbol generated by the encoding function $\varphi^{(i)}_{\mathbf{S}(i)}$ that takes an input $\mathbf{W}_{\mathbf{S}(i)}$ and outputs an element from the set $[q]$ that can be represented by $\log q$ bits. In state $\mathbf{S} \in \mathcal{P}([\nu])^n$, we denote the set of servers that have received $\mathbf{W}_u$ by

$$\mathcal{A}_{\mathbf{S}}(u) = \{j \in [n] : u \in \mathbf{S}(j)\}$$

and a version $u \in [\nu]$ is termed *complete* if

$$|\mathcal{A}_{\mathbf{S}}(u)| \geq c_W,$$

where $c_W \leq n - f$. The set of complete versions in state $\mathbf{S} \in \mathcal{P}([\nu])^n$ is given by

$$\mathcal{C}_{\mathbf{S}} = \{u \in [\nu] : |\mathcal{A}_{\mathbf{S}}(u)| \geq c_W\}$$

and the latest among them is denoted by

$$L_{\mathbf{S}} \coloneqq \max\ \mathcal{C}_{\mathbf{S}}.$$

The goal of the multi-version coding problem is to devise encoders such that for every decoder that connects to any arbitrary set of $c_R \leq n - f$ servers, the latest complete version or a later version is decodable with probability of error that is at most ϵ while minimizing the per-server worst-case storage cost. We express this formally next.

Definition 2 (ϵ-error $(n, c_W, c_R, \nu, 2^K, q, \delta_K)$ multi-version code (MVC))**.** *An ϵ-error $(n, c_W, c_R, \nu, 2^K, q, \delta_K)$ multi-version code (MVC) consists of the following*

- *encoding functions*

$$\varphi^{(i)}_{\mathbf{S}(i)} \colon [2^K]^{|\mathbf{S}(i)|} \to [q], \ \textit{for every } i \in [n]$$
$$\textit{and every state } \mathbf{S}(i) \subseteq [\nu],$$

- *decoding functions*

$$\psi_{\mathbf{S}}^{(T)}\colon [q]^{c_R} \to [2^K] \cup \{NULL\},$$

that satisfy the following

$$\Pr\left[\psi_{\mathbf{S}}^{(T)}\left(\varphi_{\mathbf{S}(t_1)}^{(t_1)}, \cdots, \varphi_{\mathbf{S}(t_{c_R})}^{(t_{c_R})}\right) = \mathbf{W}_m, \text{ for some } m \geq L_{\mathbf{S}}\right] \geq 1 - \epsilon, \tag{2.4}$$

for every possible system state $\mathbf{S} \in \mathcal{P}([\nu])^n$ *such that* $\mathcal{C}_{\mathbf{S}} \neq \emptyset$ *and every set of servers* $T = \{t_1, t_2, \cdots, t_{c_R}\} \subseteq [n]$, *where the probability is computed over all possible tuples of the message versions.*

We notice that set of possible tuples A_{δ_K} can be partitioned into disjoint sets as follows

$$A_{\delta_K} = A_{\delta_K,1} \cup A_{\delta_K,2}, \tag{2.5}$$

where $A_{\delta_K,1}$ is the set of tuples $(\mathbf{w}_1, \mathbf{w}_2, \cdots, \mathbf{w}_\nu) \in A_{\delta_K}$ for which we can decode successfully for all $\mathbf{S} \in \mathcal{P}([\nu])^n$ and $A_{\delta_K,2}$ is the set of tuples where we cannot decode successfully at least for one state $\mathbf{S} \in \mathcal{P}([\nu])^n$.

Definition 3 (Storage Cost of a Multi-version Code)**.** *The storage cost of an* ϵ*-error* $(n, c_W, c_R, \nu, 2^K, q, \delta_K)$ *MVC is equal to* $\alpha = \log q$ *bits.*

We next present an alternative decoding requirement that is shown in [7] to be equivalent to the multi-version coding problem defined above. For any set of servers $T \subseteq [n]$, note that $\max \cap_{i \in T} \mathbf{S}(i)$ denotes *the latest common version among* these servers. The alternate decoding requirement, which we refer to multi-version coding problem with Decoding Requirement A, replaces c_W, c_R by one parameter c.

The decoding requirement requires that the decoder connects to any c servers and decodes the *latest common version* amongst those c servers, or a later version.

Definition 4 (ϵ-error $(n, c, \nu, 2^K, q, \delta_K)$ Multi-version code (MVC) with Decoding Requirement A)**.** *An ϵ-error $(n, c, \nu, 2^K, q, \delta_K)$ multi-version code (MVC) consists of the following*

- *encoding functions $\varphi^{(i)}_{\mathbf{S}(i)} \colon [2^K]^{|\mathbf{S}(i)|} \to [q]$, for every $i \in [n]$ and every state $\mathbf{S}(i) \subseteq [\nu]$*
- *decoding functions $\psi^{(T)}_{\mathbf{S}} \colon [q]^c \to [2^K] \cup \{NULL\}$,*

that satisfy the following

$$\Pr\left[\psi^{(T)}_{\mathbf{S}}\left(\varphi^{(t_1)}_{\mathbf{S}(t_1)}, \cdots, \varphi^{(t_c)}_{\mathbf{S}(t_c)}\right) = \mathbf{W}_m, \textit{ for some } m \geq \max \cap_{i \in T} \mathbf{S}(i)\right] \geq 1 - \epsilon, \quad (2.6)$$

for every set of servers $T = \{t_1, t_2, \cdots, t_c\} \subseteq [n]$ and every possible system state $\mathbf{S} \in \mathcal{P}([\nu])^n$ such that $\cap_{i \in T} \mathbf{S}(i) \neq \emptyset$, where the probability is computed over all possible tuples of the message versions.

In this work, we present our achievability results for decoding requirement A and Lemma 1 establishes the connection between the two decoding requirements.

Lemma 1. *Consider any positive integers n, c_W, c_R, c such that $c = c_W + c_R - n$. An ϵ-error $(n, c, \nu, 2^K, q, \delta_K)$ MVC with decoding requirement A exists if and only if an ϵ-error $(n, c_W, c_R, \nu, 2^K, q, \delta_K)$ MVC exists.*

Lemma 1 can be shown along similar lines as [7] (e.g. see Remark 4 in this reference).

2.3.2 Background - Replication and Simple Erasure Coding

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

- *Replication-based MVCs:* In this scheme, each server only stores the latest version it receives. The encoding function is

  $$\varphi^{(i)}_{\mathbf{S}(i)}(\mathbf{W}_{\mathbf{S}(i)}) = \mathbf{W}_{s_{|\mathbf{S}(i)|}},$$

  hence the storage cost is K.

- *Simple MDS codes based MVC (MDS-MVC):* In this scheme, an (n, c) MDS code is used to encode each version separately. Specifically, suppose that $\mathcal{C} : [2^K] \rightarrow [2^{K/c}]^n$ is an (n, c) MDS code over alphabet $[2^{K/c}]$, and denote the i-th co-ordinate of the output of $\mathcal{C}$ by $\mathcal{C}^{(i)} : [2^K] \rightarrow [2^{K/c}]$. The encoding function is constructed as

  $$\varphi^{(i)}_{\mathbf{S}(i)}(\mathbf{W}_{\mathbf{S}(i)}) = (\mathcal{C}^{(i)}(\mathbf{W}_{s_1}), \mathcal{C}^{(i)}(\mathbf{W}_{s_2}), \ldots, \mathcal{C}^{(i)}(\mathbf{W}_{s_{|\mathbf{S}(i)|}})).$$

  That is, each server stores one codeword symbol for each version it receives and the storage cost is $\nu \frac{K}{c}$.

An important outcome of the study of [7] is that, when the different versions are independent, i.e., if $\delta_K = 1$, then the storage cost is at least

$$\frac{\nu K}{\nu + c - 1} - \Theta(1).$$

In particular, because $\frac{\nu}{\nu+c-1} \geq \frac{1}{2}\min(\frac{\nu}{c}, 1)$, the best possible MVC scheme is, for large K, at most twice as cost-efficient as the better among replication and simple erasure coding. In this work, we show that replication and simple erasure coding are significantly inefficient if the different versions are correlated. Our schemes resemble simple erasure codes in their construction; however, we exploit the correlation between the versions to store fewer bits per server.

Remark 2. *It is worth noting that for our problem statement, the decoding requirement for that the latest complete version* or a later version, *i.e., $m \geq L_{\mathbf{S}}$, is crucial. In fact replacing this with a requirement of $m = L_{\mathbf{S}}$ would be too strong as it would eliminate even simple replication from possible solutions. In order to see this, consider a replication-based distributed system with $n = 3, c_W = 2$ and $c_R = 2$ storing an object with two versions $\mathbf{W}_1$ and $\mathbf{W}_2$. Consider two states $\mathbf{S}_1$ and $\mathbf{S}_2$, where $\mathbf{W}_1$ is the latest complete version in $\mathbf{S}_1$ and $\mathbf{W}_2$ is the latest complete version in $\mathbf{S}_2$ as shown in Fig. 2.1. We assume that a decoder connects to the first two servers. Since the decoder cannot differentiate between the two scenarios, the read client has to return $\mathbf{W}_2$ in both scenarios, and therefore returns a version that is later than $L_{\mathbf{S_1}}$ in $\mathbf{S}_1$.*

2.3.3 Summary of Results

In order to explain the significance of our results, summarized in Table 2.1, we begin with a simple motivating scheme. Consider the MDS-MVC scheme of Section 2.3.2. Assume that we use a Reed-Solomon code over a field $\mathbb{F}_p$ of binary characteristic. The generator matrix of a Reed-Solomon code is usually expressed over $\mathbb{F}_p$. However, every element in $\mathbb{F}_p$ is a vector over $\mathbb{F}_2$, and a multiplication over the extension field $\mathbb{F}_p$ is a linear transformation over $\mathbb{F}_2$. Therefore, the generator matrix of the

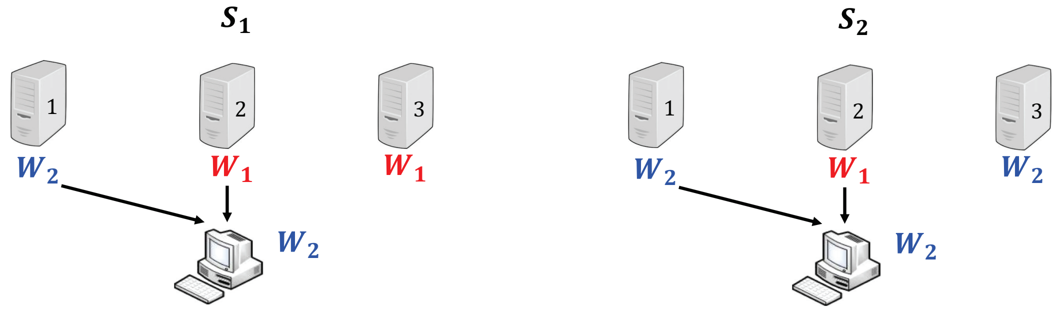


Figure 2.1. A distributed storage system with $n = 3$ and $c_W = c_R = 2$. In $\mathbf{S}_1$, $\mathbf{W}_1$ is the latest complete version while $\mathbf{W}_2$ is the latest complete version in $\mathbf{S}_2$. The read client cannot differentiate between the two states $\mathbf{S}_1$ and $\mathbf{S}_2$. Hence, the read client has to return $\mathbf{W}_2$ in both states, and therefore returns a version that is later than the latest complete version in state $\mathbf{S}_1$.

Reed-Solomon code can be equivalently expressed over $\mathbb{F}_2$ as follows

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

where G is a $K \times nK/c$ binary generator matrix, and $G^{(i)}$ has dimension $K \times K/c$. Because Reed-Solomon codes can tolerate $n - c$ erasures, every matrix of the form $(G^{(t_1)}, G^{(t_2)}, \ldots, G^{(t_c)})$, where $t_1, t_2, \ldots, t_c$ are distinct elements of $[n]$, has a full rank of K over $\mathbb{F}_2$.

We now describe a simple scheme that extends the MDS-MVC by exploiting the correlations and requires the knowledge of δ_K. Suppose that the i-th server receives the set of versions $\mathbf{S}(i) = \{s_1, s_2, \cdots, s_{|\mathbf{S}(i)|}\}$, where $s_1 < s_2 < \ldots < s_{|\mathbf{S}(i)|}$. The server encodes $\mathbf{W}_{s_1}$ using the binary code as $\mathbf{W}_{s_1}^{\mathrm{T}} G^{(i)}$. For $\mathbf{W}_{s_m}$, where $m > 1$, the server finds a difference vector $\mathbf{y}^{(i)}_{s_m, s_{m-1}}$ that satisfies the following

1. $\mathbf{y}^{(i)\,\mathrm{T}}_{s_m, s_{m-1}} G^{(i)} = (\mathbf{W}_{s_m} - \mathbf{W}_{s_{m-1}})^{\mathrm{T}} G^{(i)}$ and

2. $w_H(\mathbf{y}^{(i)}_{s_m, s_{m-1}}) \leq (s_m - s_{m-1})\delta_K K$.

Although it is not necessary that $\mathbf{y}^{(i)}_{s_m, s_{m-1}} = \mathbf{W}_{s_m} - \mathbf{W}_{s_{m-1}}$, the fact that $\mathbf{W}_{s_m} -$

$\mathbf{W}_{s_{m-1}}$ satisfies these two conditions implies that the encoder can find at least one vector $\mathbf{y}^{(i)}_{s_m,s_{m-1}}$ satisfying these conditions. Since $w_H(\mathbf{y}^{(i)}_{s_m,s_{m-1}}) \leq (s_m - s_{m-1})\delta_K K$, an encoder aware of δ_K can represent $\mathbf{y}^{(i)}_{s_m,s_{m-1}}$ by $\log Vol((s_m - s_{m-1})\delta_K K, K)$ bits. The first condition implies that a decoder that connects to the i-th server can obtain $\mathbf{W}^{\mathrm{T}}_{s_m} G^{(i)}$ by applying

$$\mathbf{W}^{\mathrm{T}}_{s_1} G^{(i)} + \sum_{\ell=2}^{m} \mathbf{y}^{(i)\,\mathrm{T}}_{s_\ell,s_{\ell-1}} G^{(i)} = \mathbf{W}^{\mathrm{T}}_{s_m} G^{(i)}.$$

Therefore, from any subset $\{t_1, t_2, \ldots, t_c\}$ of c servers, for any common version s_m among these servers, a decoder can recover

$$\left(\mathbf{W}^{\mathrm{T}}_{s_m} G^{(t_1)}, \mathbf{W}^{\mathrm{T}}_{s_m} G^{(t_2)}, \ldots, \mathbf{W}^{\mathrm{T}}_{s_m} G^{(t_c)}\right)$$

from these servers and can therefore recover $\mathbf{W}_{s_m}$.

The worst-case storage cost of this scheme is obtained when each server receives all the ν versions, which results in a storage cost of

$$\frac{K}{c} + (\nu - 1)\log Vol(\delta_K K, K).$$

This motivating scheme stores the first version using erasure coding - K/c bits - and the remaining $(\nu - 1)$ versions using delta coding, which adds a storage cost of $\log Vol(\delta_K K, K)$ bits per version. This scheme motivates the following questions.

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

Q2*: Can we use erasure coding to achieve a storage cost of $\frac{K}{c} + \frac{\nu-1}{c}\log Vol(\delta_K K, K)$?*

In Section 2.4.1, we provide Theorem 1 that answers $Q1$ by developing a 0-error Reed-Solomon based scheme that does not require the knowledge of δ_K and obtains

Scheme	**Worst-case Storage Cost**	**Regime**
Replication	K	oblivious to δ_K, $\epsilon = 0$
Simple erasure codes	$\nu\frac{K}{c}$	outperforms replication if $\nu < c$, oblivious to δ_K, $\epsilon = 0$
Theorem 1 [Reed-Solomon update-efficient code]	$\frac{K}{c} + (\nu - 1)\delta_K K(\log K + o(\log K))$	asymptotically outperforms the above schemes for $\nu < c$ and $\delta_K = o(1/\log K)$, oblivious to δ_K, $\epsilon = 0$
Motivating Scheme of Section 2.3.3	$\frac{K}{c} + (\nu - 1)\log Vol(\delta_K K, K)$	not oblivious to δ_K, $\epsilon = 0$
Theorem 2 [Random binning]	$\frac{K}{c} + \frac{\nu - 1}{c}\log Vol(\delta_K K, K) + o(\log K)$	asymptotically outperforms the above schemes for $\nu < c$, not oblivious to δ_K, $\epsilon = 1/\log K$
Theorem 3 [Lower bound]	$\frac{K}{c + \nu - 1} + \frac{\nu - 1}{c + \nu - 1}\log Vol(K, \delta_K K) - \Theta(1)$	applicable for all δ_K

Table 2.1. Storage cost.

the erasure coding gain of $1/c$ for the first version available at a server and stores the subsequent versions via delta coding.

In Section 2.4.2, we provide Theorem 2 that gives a positive answer to $Q2$ by showing the existence of an ϵ-error storage efficient scheme that obtains the erasure coding factor of $1/c$, not only for the first version, but also for the subsequent versions. Moreover, the scheme is able to harness the delta compression gain.

Finally, in Section 2.5, Theorem 3 provides a lower bound on the per-server storage cost which implies that for $\nu < c$, constant $\delta_K = \delta$ and $\epsilon = 2^{-o(K)}$, the achievable scheme of Theorem 2 is asymptotically at most twice the lower bound. We notice that the regime where $\nu < c$ is interesting as the degree of asynchrony is typically limited as pointed out in [21].

2.4 Code Constructions (Theorem 1 and Theorem 2)

In this section, we provide our code constructions. We study the case where δ_K is not known and present a MVC based on Reed-Solomon code in Section 2.4.1. Later on in this section, we study the case where δ_K is known and develop a random binning argument in Section 2.4.2.

2.4.1 Update-efficient Multi-version Codes

We develop a simple multi-version coding scheme that exploits the correlation between the different versions and have smaller storage cost as compared with [7]. In this scheme, the servers do not know the correlation degree δ_K in advance. We begin by recalling the definition of the update efficiency of a code from [47].

Definition 5 (Update efficiency)**.** *For a code $\mathcal{C}$ of length N and dimension K with encoder $\mathcal{C}: \mathbb{F}^K \to \mathbb{F}^N$, the update efficiency of the code is the maximum number of codeword symbols that must be updated when a single message symbol is changed and is expressed as follows*

$$t = \max_{\substack{\mathbf{W}, \mathbf{W}' \in \mathbb{F}^K: \\ d_H(\mathbf{W}, \mathbf{W}')=1}} d_H(\mathcal{C}(\mathbf{W}), \mathcal{C}(\mathbf{W}')). \tag{2.7}$$

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

Definition 6 (Update efficiency of a server)**.** *Suppose that $\mathcal{C}^{(i)} : \mathbb{F}^K \to \mathbb{F}^{N/n}$ denotes the i-th co-ordinate of the output of $\mathcal{C}$ stored by the i-th server. The update efficiency of the i-th server is the maximum number of codeword symbols that must be updated in this server when a single message symbol is changed and is expressed as follows*

$$t^{(i)} = \max_{\substack{\mathbf{W},\mathbf{W}' \in \mathbb{F}^K: \\ d_H(\mathbf{W},\mathbf{W}')=1}} d_H(\mathcal{C}^{(i)}(\mathbf{W}), \mathcal{C}^{(i)}(\mathbf{W}')). \tag{2.8}$$

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

Definition 7 (The per-server maximum update efficiency)**.** *The per-server maximum update efficiency is the maximum number of codeword symbols that must be updated in any server when a single message symbol is changed and is given by*

$$t_s = \max_{i \in [n]} t^{(i)}. \tag{2.9}$$

We next present an update-efficient MVC construction, illustrated in Fig. 2.2, that is based on Reed-Solomon code and has a maximum update efficiency per-server $t_s = 1$.

Construction 1 (Reed-Solomon Update-Efficient MVC)**.** *Suppose that the i-th server receives the versions $\mathbf{S}(i) = \{s_1, s_2, \ldots, s_{|\mathbf{S}(i)|}\}$, where $s_1 < s_2 < \cdots < s_{|\mathbf{S}(i)|}$. A version $\mathbf{W}_{s_j}$ is divided into $\frac{K}{c \log n_p}$ blocks of length $c \log n_p$, where $n_p = 2^{\lceil \log_2 n \rceil}$.*

In each block, every consecutive string of $\log n_p$ *bits is represented by a symbol in* $\mathbb{F}_{n_p}$*. The representation of* $\mathbf{W}_{s_j}$ *over* $\mathbb{F}_{n_p}$ *is denoted by* $\overline{\mathbf{W}}_{s_j}$*. Each block is then encoded by an* (n, c) *Reed-Solomon code with a generator matrix* $\tilde{G}$ *that is given by*

$$\tilde{G} = \begin{pmatrix} 1 & 1 & \cdots & 1 \\ \lambda_1 & \lambda_2 & \cdots & \lambda_n \\ \lambda_1^2 & \lambda_2^2 & \cdots & \lambda_n^2 \\ \vdots & \vdots & \vdots & \vdots \\ \lambda_1^{c-1} & \lambda_2^{c-1} & \cdots & \lambda_n^{c-1} \end{pmatrix}, \tag{2.10}$$

where $\Lambda = \{\lambda_1, \lambda_2, \cdots, \lambda_n\} \subset \mathbb{F}_{n_p}$ *is a set of distinct elements. For* $\mathbf{W}_{s_1}$*, the i-th server stores* $\overline{\mathbf{W}}_{s_1}^{\mathrm{T}} G^{(i)}$*, where* $G^{(i)}$ *is a* $\frac{K}{\log n_p} \times \frac{K}{c \log n_p}$ *matrix that is given by*

$$G^{(i)} = \begin{pmatrix} \tilde{G}\mathbf{e}_i & 0 & \cdots & 0 & 0 \\ 0 & \tilde{G}\mathbf{e}_i & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & 0 & \tilde{G}\mathbf{e}_i \end{pmatrix}, \tag{2.11}$$

where $\mathbf{e}_i$ *is i-th standard basis vector over* $\mathbb{F}_{n_p}$*. For* $\mathbf{W}_{s_m}$*, where* $m > 1$*, the server may only store the updated symbols from the old version* $\overline{\mathbf{W}}_{s_{m-1}}$ *or store* $\overline{\mathbf{W}}_{s_m}^{\mathrm{T}} G^{(i)}$*.*

Theorem 1. *[Reed-Solomon Update-Efficient MVC] Construction 1 is a* 0*-error* $(n, c, \nu, 2^K, q, \delta_K)$ *multi-version code with a worst-case storage that is at most*

$$\frac{K}{c} + (\nu - 1) \min(\delta_K K \log\left(\frac{K n_p}{c \log n_p}\right), K/c), \tag{2.12}$$

where $n_p = 2^{\lceil \log_2 n \rceil}$*.*

Proof of Theorem 1. We observe that Construction 1 is a valid multi-version code

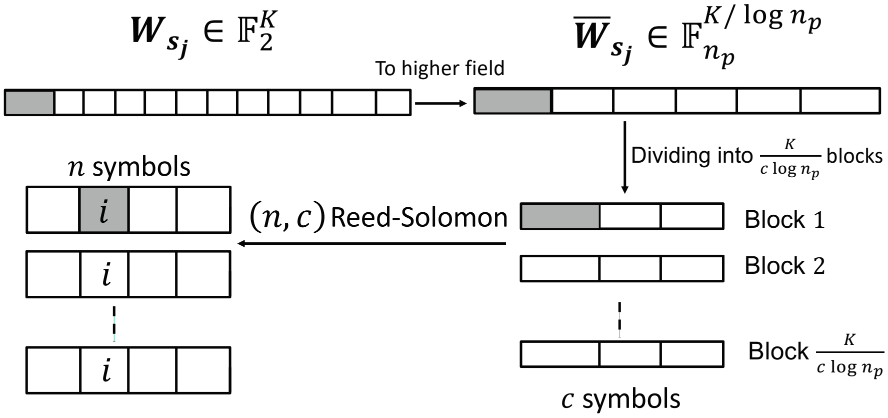

$W_{s_j} \in \mathbb{F}_2^K$
To higher field
$\overline{W}_{s_j} \in \mathbb{F}_{n_p}^{K/\log n_p}$
Dividing into $\frac{K}{c \log n_p}$ blocks
n symbols
i
i
i
(n, c) Reed-Solomon
Block 1
Block 2
Block $\frac{K}{c \log n_p}$
c symbols

Figure 2.2. Illustration of Construction 1. A bit that changes in the message leads to at most one updated codeword symbol in the i-th server, $\forall i \in [n]$, hence $t_s = 1$.

as the latest common version is recoverable from any c servers by the MDS property of Reed-Solomon code. In order to characterize the worst-case storage cost, we observe that the update efficiency of the i-th server is equal to the maximum row weight of $G^{(i)}$ which is equal to $1, \forall i \in [n]$. Thus, the per-server maximum update efficiency t_s is also equal to 1.

The worst-case storage cost corresponds to the case where a server receives all versions. In this case, the server stores $\overline{\mathbf{W}}_1^{\mathrm{T}} G^{(i)}$ for the first version. For $\mathbf{W}_u$, where $u \geq 2$, the server may only store the updated symbols from the old version $\overline{\mathbf{W}}_{u-1}$ or store $\overline{\mathbf{W}}_u^{\mathrm{T}} G^{(i)}$. Storing the index of an updated symbol requires $\log(\frac{K}{c \log n_p})$ bits and storing the value requires $\log n_p$. Therefore, the per-server storage cost is upper-bounded as follows

$$\begin{aligned}
\alpha &\leq \frac{K}{c} + \sum_{u=2}^{\nu} \min(d_H(\mathbf{W}_u, \mathbf{W}_{u-1}) \log\left(\frac{K n_p}{c \log n_p}\right), K/c) \\
&\leq \frac{K}{c} + (\nu - 1) \min(\delta_K K \log\left(\frac{K n_p}{c \log n_p}\right), K/c).
\end{aligned}$$

□

2.4.2 Random Binning Based Multi-version Codes

We next introduce a random binning argument showing the existence of a multi-version code that can harness both of the erasure and the delta coding gains for all versions for the case where δ_K is known. Recall that Slepian-Wolf coding [36, 53] is a distributed data compression technique for correlated sources that are drawn in independent and identical manner according to a given distribution. In the Slepian-Wolf setting, the decoder is interested in decoding the data of all sources. In the multi-version coding problem, the decoder is interested in decoding the latest common version, or a later version, among any set of c servers.

We notice that our model differs from the Slepian-Wolf setting as we do not aspire to decode all the versions. The lossless source coding problem with a helper [54–56] may also seem related to our approach, since the side information of the older versions may be interpreted as helpers. In the optimal strategy for the helper setting, the helper side information is encoded via a joint typicality encoding scheme, whereas the random binning is used for the message. However, in the multi-version coding setting, a version that may be a side information for one state may be required to be decoded in another state. For this reason, a random binning scheme for all versions leads to schemes with a near-optimal storage cost. We now present our argument that is inspired by Cover's random binning proof of the Slepian-Wolf problem [56].

Construction 2 (Random binning multi-version code)**.** *Suppose that the i-th server receives the versions $\mathbf{S}(i) = \{s_1, s_2, \cdots, s_{|\mathbf{S}(i)|}\} \subseteq [\nu]$, where $s_1 < s_2 < \cdots < s_{|\mathbf{S}(i)|}$.*

- Random code generation: *At the i-th server, for a version s_j the encoder assigns an index at random from $\{1, 2, \cdots, 2^{R_{s_j}^{(i)}/c}\}$ uniformly and independently* *to each vector of length K bits, where $R_{s_j}^{(i)}/c$ is the rate assigned by the i-th server to version s_j.*

- Encoding*: The server stores the corresponding index to each version that it receives and the decoder is also aware of this mapping. The encoding function of the i-th server is given by*

$$\varphi_{\mathbf{S}(i)}^{(i)} = (\varphi_{s_1}^{(i)}, \varphi_{s_2}^{(i)}, \cdots, \varphi_{s_{|\mathbf{S}(i)|}}^{(i)}), \tag{2.13}$$

where $\varphi_{s_j}^{(i)} \colon [2^K] \to \{1, 2 \cdots, 2^{KR_{s_j}^{(i)}/c}\}$ and we choose the rates as follows

$$KR_{s_1}^{(i)} = K + (s_1 - 1)\log Vol(\delta_K K, K) + (s_1 - 1) - \log \epsilon 2^{-\nu n}, \tag{2.14}$$

$$KR_{s_j}^{(i)} = (s_j - s_{j-1})\log Vol(\delta_K K, K) + (s_j - 1) - \log \epsilon 2^{-\nu n},$$
$$j \in \{2, 3, \cdots, |\mathbf{S}(i)|\}. \tag{2.15}$$

Consider a state $\mathbf{S} \in \mathcal{P}([\nu])^n$ and suppose that the decoder connects to the servers $T = \{t_1, t_2, \cdots, t_c\} \subseteq [n]$. Suppose that a version s_j is received by a set of servers $\{i_1, i_2, \cdots, i_r\} \subseteq T$, then the bin index corresponding to this version is given by

$$\varphi_{s_j} = (\varphi_{s_j}^{(i_1)}, \varphi_{s_j}^{(i_2)}, \cdots, \varphi_{s_j}^{(i_r)}). \tag{2.16}$$

In this case, the rate of version s_j is given by

$$R_{s_j} = \frac{1}{c} \sum_{i \in \{i_1, i_2, \cdots, i_r\}} R_{s_j}^{(i)}. \tag{2.17}$$

Decoding*: The decoder employs the* possible set decoding *strategy as follows. Assume that $\mathbf{W}_{u_L}$ is the latest common version in $\mathbf{S}$ and that versions*

$\mathbf{W}_{u_1}, \mathbf{W}_{u_2}, \cdots, \mathbf{W}_{u_{L-1}}$ *are the older versions such that each of them is received by at least one server out of those c servers. We denote this set of versions by $\mathbf{S}_{\mathrm{T}}$ and define it formally as follows*

$$\begin{aligned} \mathbf{S}_{\mathrm{T}} &= \{u_1, u_2, \cdots, u_L\} \\ &= \left(\bigcup_{t \in T} \mathbf{S}(t) \right) \setminus \{u_L + 1, u_L + 2, \cdots, \nu\}, \end{aligned} \tag{2.18}$$

where $u_1 < u_2 < \cdots < u_L$. Given the bin indices $(b_{u_1}, b_{u_2}, \cdots, b_{u_L})$, the decoder finds all tuples $(\mathbf{w}_{u_1}, \mathbf{w}_{u_2}, \cdots, \mathbf{w}_{u_L}) \in A_{\delta_K}$ such that

$$(\varphi_{u_1}(\mathbf{w}_{u_1}) = b_{u_1}, \varphi_{u_2}(\mathbf{w}_{u_2}) = b_{u_2}, \cdots, \varphi_{u_L}(\mathbf{w}_{u_L}) = b_{u_L}).$$

If all of these tuples have the same latest common version $\mathbf{w}_{u_L}$, the decoder declares $\mathbf{w}_{u_L}$ to be the estimate of the latest common version $\hat{\mathbf{W}}_{\mathbf{u_L}}$. Otherwise, the decoder declares an error.

Theorem 2. *[Random Binning MVC] There exists an ϵ-error $(n, c, \nu, 2^K, q, \delta_K)$ multi-version code whose worst-case storage cost is at most*

$$\frac{K}{c} + \frac{(\nu - 1) \log Vol(\delta_K K, K)}{c} + \frac{\nu(\nu - 1)/2 - \nu \log \epsilon 2^{-\nu n}}{c}. \tag{2.19}$$

Proof of Theorem 2. We show that Construction 2 is an ϵ-error multi-version code. We denote the error event by E and express it as follows

$$E = \{\exists (\mathbf{w}'_{u_1}, \mathbf{w}'_{u_2}, \ldots, \mathbf{w}'_{u_L}) \in A_{\delta_K} : \mathbf{w}'_{u_L} \neq \mathbf{W}_{u_L} \text{ and } \varphi_u(\mathbf{w}'_u) = \varphi_u(\mathbf{W}_u), \forall u \in \mathbf{S}_T\}. \tag{2.20}$$

The error event in decoding can be equivalently expressed as follows

$$E = \bigcup_{\mathcal{I} \subseteq \mathbf{S}_T : u_L \in \mathcal{I}} E_{\mathcal{I}}, \tag{2.21}$$

where

$$E_{\mathcal{I}} = \{\exists \mathbf{w}'_u \neq \mathbf{W}_u, \forall u \in \mathcal{I} : \varphi_u(\mathbf{w}'_u) = \varphi_u(\mathbf{W}_u), \forall u \in \mathcal{I} \text{ and } (\mathbf{w}'_{\mathcal{I}}, \mathbf{W}_{\mathbf{S}_T \setminus \mathcal{I}}) \in A_{\delta_K}\}, \tag{2.22}$$

for $\mathcal{I} \subseteq \mathbf{S}_T$ such that $u_L \in \mathcal{I}$. By the union bound, we have

$$\begin{aligned} P_e(\mathbf{S}, T) &:= P(E) \\ &= P\left(\bigcup_{\mathcal{I} \subseteq \mathbf{S}_T : u_L \in \mathcal{I}} E_{\mathcal{I}} \right) \\ &\leq \sum_{\mathcal{I} \subseteq \mathbf{S}_T : u_L \in \mathcal{I}} P(E_{\mathcal{I}}), \end{aligned}$$

and we require that

$$P_e(\mathbf{S}, T) \leq \epsilon 2^{-\nu n}.$$

Thus, for every $\mathcal{I} \subseteq \mathbf{S}_T$ such that $u_L \in \mathcal{I}$, it suffices to show that

$$P(E_{\mathcal{I}}) \leq \epsilon 2^{-(L-1)} 2^{-\nu n}.$$

We now proceed in a case by case manner as shown in Fig. 2.3. We first consider the case where $u_{L-1} \notin \mathcal{I}$, later we consider the case where $u_{L-1} \in \mathcal{I}$. For the case

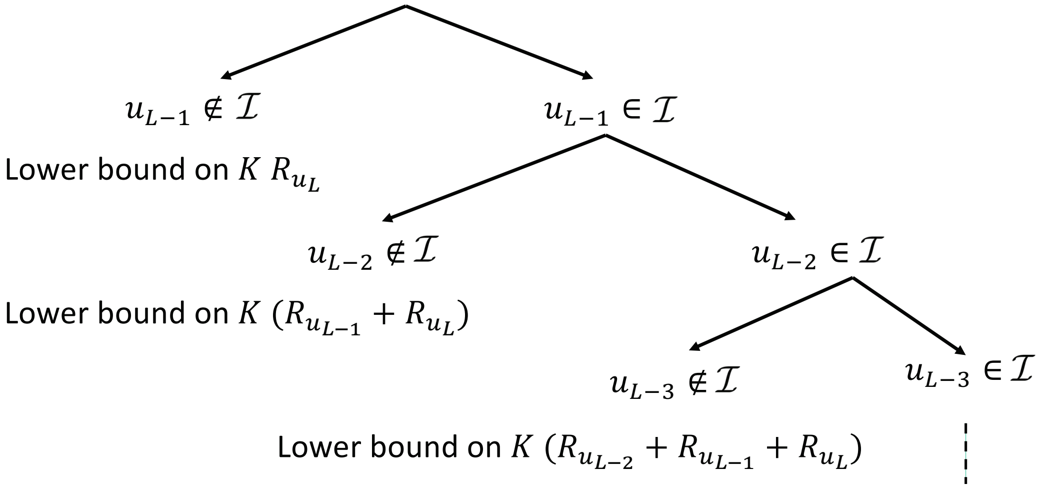

$u_{L-1} \notin \mathcal{I}$
$u_{L-1} \in \mathcal{I}$
Lower bound on $K\ R_{u_L}$
$u_{L-2} \notin \mathcal{I}$
$u_{L-2} \in \mathcal{I}$
Lower bound on $K\ (R_{u_{L-1}} + R_{u_L})$
$u_{L-3} \notin \mathcal{I}$
$u_{L-3} \in \mathcal{I}$
Lower bound on $K\ (R_{u_{L-2}} + R_{u_{L-1}} + R_{u_L})$

Figure 2.3. Illustration of the error analysis of Construction 2.

where $u_{L-1} \notin \mathcal{I}$, we have

$$E_{\mathcal{I}} \subset \tilde{E}_{u_{L-1}} := \{\exists \mathbf{w}'_{u_L} \neq \mathbf{W}_{u_L} : \varphi_{u_L}(\mathbf{w}'_{u_L}) = \varphi_{u_L}(\mathbf{W}_{u_L}) \text{ and } (\mathbf{W}_{u_{L-1}}, \mathbf{w}'_{u_L}) \in A_{\delta_K}\}. \tag{2.23}$$

Consequently, we have

$$\begin{aligned}
P(E_{\mathcal{I}}) &\leq P(\tilde{E}_{u_{L-1}}) \\
&= \sum_{(\mathbf{w}_{u_{L-1}}, \mathbf{w}_{u_L})} p(\mathbf{w}_{u_{L-1}}, \mathbf{w}_{u_L}) \\
&\qquad\qquad P(\exists \mathbf{w}'_{u_L} \neq \mathbf{w}_{u_L} : \varphi_{u_L}(\mathbf{w}'_{u_L}) = \varphi_{u_L}(\mathbf{w}_{u_L}) \text{ and } (\mathbf{w}_{u_{L-1}}, \mathbf{w}'_{u_L}) \in A_{\delta_K}) \\
&\overset{(a)}{\leq} \sum_{(\mathbf{w}_{u_{L-1}}, \mathbf{w}_{u_L})} p(\mathbf{w}_{u_{L-1}}, \mathbf{w}_{u_L}) \sum_{\substack{\mathbf{w}'_{u_L} \neq \mathbf{w}_{u_L} \\ (\mathbf{w}_{u_{L-1}}, \mathbf{w}'_{u_L}) \in A_{\delta_K}}} P(\varphi_{u_L}(\mathbf{w}'_{u_L}) = \varphi_{u_L}(\mathbf{w}_{u_L})) \\
&\overset{(b)}{=} \sum_{(\mathbf{w}_{u_{L-1}}, \mathbf{w}_{u_L})} p(\mathbf{w}_{u_{L-1}}, \mathbf{w}_{u_L}) \sum_{\substack{\mathbf{w}'_{u_L} \neq \mathbf{w}_{u_L} \\ (\mathbf{w}_{u_{L-1}}, \mathbf{w}'_{u_L}) \in A_{\delta_K}}} \prod_{i=1}^{c} P(\varphi^{(t_i)}_{u_L}(\mathbf{w}'_{u_L}) = \varphi^{(t_i)}_{u_L}(\mathbf{w}_{u_L})) \\
&\leq \sum_{(\mathbf{w}_{u_{L-1}}, \mathbf{w}_{u_L})} p(\mathbf{w}_{u_{L-1}}, \mathbf{w}_{u_L}) |A_{\delta_K}(\mathbf{W}_{u_L} | \mathbf{w}_{u_{L-1}})| \ \prod_{i=1}^{c} 2^{-KR^{(t_i)}_{u_L}/c}
\end{aligned}$$

$$\begin{aligned}&\overset{(c)}{=} \sum_{(\mathbf{w}_{u_{L-1}},\mathbf{w}_{u_L})} p(\mathbf{w}_{u_{L-1}},\mathbf{w}_{u_L})|A_{\delta_K}(\mathbf{W}_{u_L}|\mathbf{w}_{u_{L-1}})|\ 2^{-KR_{u_L}}\\&= \sum_{(\mathbf{w}_{u_{L-1}},\mathbf{w}_{u_L})} p(\mathbf{w}_{u_{L-1}},\mathbf{w}_{u_L})2^{-(KR_{u_L}-\log Vol((u_L-u_{L-1})\delta_K K,K))}\\&= 2^{-(KR_{u_L}-\log Vol((u_L-u_{L-1})\delta_K K,K))}, \qquad (2.24)\end{aligned}$$

where (a) follows by the union bound, (b) follows since each server assigns an index uniformly and independently from the other servers and (c) follows from (2.17). Choosing R_{u_L} to satisfy

$$KR_{u_L} \geq \log Vol((u_L - u_{L-1})\delta_K K, K) + (L-1) - \log \epsilon 2^{-\nu n}$$

ensures that

$$P(E_{\mathcal{I}}) \leq \epsilon 2^{-(L-1)} 2^{-\nu n}.$$

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

$$\begin{aligned}E_{\mathcal{I}} \subseteq \tilde{E}_{u_{L-2}} := \{\exists \mathbf{w}'_{u_{L-1}} \neq \mathbf{W}_{u_{L-1}}, \mathbf{w}'_{u_L} \neq \mathbf{W}_{u_L} : \varphi_{u_{L-1}}(\mathbf{w}'_{u_{L-1}}) = \varphi_{u_{L-1}}(\mathbf{W}_{u_{L-1}}),\\ \varphi_{u_L}(\mathbf{w}'_{u_L}) = \varphi_{u_L}(\mathbf{W}_{u_L}) \text{ and } (\mathbf{W}_{u_{L-2}}, \mathbf{w}'_{u_{L-1}}, \mathbf{w}'_{u_L}) \in A_{\delta_K}\}. \quad (2.25)\end{aligned}$$

Therefore, we have

$$\begin{aligned}P(E_{\mathcal{I}}) &\leq P(\tilde{E}_{u_{L-2}}) \qquad (2.26)\\&= \sum_{(\mathbf{w}_{u_{L-2}},\mathbf{w}_{u_{L-1}},\mathbf{w}_{u_L})} p(\mathbf{w}_{u_{L-2}},\mathbf{w}_{u_{L-1}},\mathbf{w}_{u_L})\end{aligned}$$

$$
\begin{aligned}
&\sum_{\substack{\mathbf{w}'_{u_{L-1}} \neq \mathbf{w}_{u_{L-1}} \\ \mathbf{w}'_{u_L} \neq \mathbf{w}_{u_L} \\ (\mathbf{w}_{u_{L-2}}, \mathbf{w}_{u'_{L-1}}, \mathbf{w}'_{u_L}) \in A_{\delta_K}}} P(\varphi(\mathbf{w}'_{u_{L-1}}) = \varphi(\mathbf{w}_{u_{L-1}})) P(\varphi_{u_L}(\mathbf{w}'_{u_L}) = \varphi(\mathbf{w}_{u_L})) \\
&\leq \sum_{(\mathbf{w}_{u_{L-2}}, \mathbf{w}_{u_{L-1}}, \mathbf{w}_{u_L})} p(\mathbf{w}_{u_{L-2}}, \mathbf{w}_{u_{L-1}}, \mathbf{w}_{u_L}) 2^{-K(R_{u_{L-1}} + R_{u_L})} \qquad (2.27) \\
&\qquad |A_{\delta_K}(\mathbf{W}_{u_{L-1}}, \mathbf{W}_{u_L} | \mathbf{w}_{u_{L-2}})| \\
&= \sum_{(\mathbf{w}_{u_{L-2}}, \mathbf{w}_{u_{L-1}}, \mathbf{w}_{u_L})} p(\mathbf{w}_{u_{L-2}}, \mathbf{w}_{u_{L-1}}, \mathbf{w}_{u_L}) 2^{-K(R_{u_{L-1}} + R_{u_L})} \\
&\quad 2^{\log Vol((u_L - u_{L-1})\delta_K K, K) + \log Vol((u_{L-1} - u_{L-2})\delta_K K, K)}. \qquad (2.28)
\end{aligned}
$$

In this case, we choose the rates as follows

$$
K(R_{u_{L-1}} + R_{u_L}) \geq \sum_{j=L-1}^{L} \log Vol((u_j - u_{j-1})\delta_K K, K) + (L-1) - \log \epsilon 2^{-\nu n}.
$$

We next consider the other case where $u_{L-2} \in \mathcal{I}$. In this case, we also have two cases based on whether u_{L-3} is in $\mathcal{I}$ or not. By applying the above argument repeatedly, we obtain the following conditions for the overall probability of error to be upper bounded by $\epsilon 2^{-\nu n}$.

$$
\begin{aligned}
&K \sum_{j=i}^{L} R_{u_j} \geq \sum_{j=i}^{L} \log Vol((u_j - u_{j-1})\delta_K K, K) + (L-1) - \log \epsilon 2^{-\nu n}, \\
&\qquad \forall i \in \{2, 3, \cdots, L\}, \\
&K \sum_{j=1}^{L} R_{u_j} \geq K + \sum_{j=2}^{L} \log Vol((u_j - u_{j-1})\delta_K K, K) + (L-1) - \log \epsilon 2^{-\nu n}. \qquad (2.29)
\end{aligned}
$$

Since

$$
\log Vol(m\delta_K K, K) \leq m \log Vol(\delta_K K, K), \forall m \in \mathbb{Z}^+, \qquad (2.30)
$$

it suffices if the rates satisfy

$$K\sum_{j=i}^{L} R_{u_j} \geq \sum_{j=i}^{L}(u_j - u_{j-1}) \log Vol(\delta_K K, K) + (L-1) - \log \epsilon 2^{-\nu n},$$
$$\forall i \in \{2, 3, \cdots, L\},$$
$$K\sum_{j=1}^{L} R_{u_j} \geq K + \sum_{j=2}^{L}(u_j - u_{j-1}) \log Vol(\delta_K K, K) + (L-1) - \log \epsilon 2^{-\nu n}. \quad (2.31)$$

The rates chosen in (2.14), (2.15) satisfy the above inequalities, therefore our construction has a probability of error bounded by $\epsilon 2^{-\nu n}$.

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

$$\frac{K - \log \epsilon 2^{-\nu n}}{c} + \frac{(\nu - 1)(\log Vol(\delta_K K, K) - \log \epsilon 2^{-\nu n} + \nu/2)}{c}.$$

It remains to show that there exists a deterministic multi-version code that has a probability of error that is at most ϵ for all possible $2^{\nu n}$ states, where the j-th state is denoted by $\mathbf{S}_j, j \in [2^{\nu n}]$. Suppose that the ensemble of the random code has m codes, where the i-th code is denoted by $\mathcal{C}_i, i \in [m]$. We denote the probability of error of the i-th code in state $\mathbf{S}_j$ by $\epsilon_i^{(j)}$. Since we have shown that the average probability of error is at most $\epsilon 2^{-\nu n}$, then we have

$$\frac{1}{m}\sum_{i=1}^{m} \epsilon_i^{(j)} \leq \epsilon 2^{-\nu n}, \quad (2.32)$$

for every $j \in [2^{\nu n}]$. Therefore, we have

$$\sum_{j=1}^{2^{\nu n}}\sum_{i=1}^{m} \epsilon_i^{(j)} \leq m\epsilon. \quad (2.33)$$

We now show that there exists a deterministic code $\mathcal{C}_l$, $l \in [m]$, where

$$\epsilon_l^{(j)} \leq \epsilon, \forall j \in [2^{\nu n}] \tag{2.34}$$

by contradiction. Suppose that for every code $\mathcal{C}_i$, $i \in [m]$, there exists a state $\mathbf{S}_j \in \mathcal{P}([\nu])^n$, where

$$\epsilon_i^{(j)} > \epsilon. \tag{2.35}$$

In this case, we have

$$\sum_{j=1}^{2^{\nu n}} \sum_{i=1}^{m} \epsilon_i^{(j)} > m\epsilon, \tag{2.36}$$

and we have a contradiction. □

Motivated by the fact that linear codes have lower complexity, in Appendix A, we show that linear codes exist that achieve the storage cost of Theorem 2. Our proof is inspired by [37].

Remark 3. *The proof of Theorem 2 uses simultaneous non-unique decoding ideas [57] used in several multi-user scenarios. In particular, with our non-unique decoding approach to decode $\mathbf{W}_{u_L}$, the decoder picks the unique $\mathbf{w}_{u_L}$ such that $(\mathbf{w}_{u_1}, \mathbf{w}_{u_2}, \ldots, \mathbf{w}_{u_L}) \in A_{\delta_K}$ for* some *$\mathbf{w}_{u_1}, \mathbf{w}_{u_2}, \ldots, \mathbf{w}_{u_{L-1}}$, which are consistent with the bin indices. We use this strategy since unlike the Slepian-Wolf problem where all the data of all sources are to be decoded, we are only required to decode the latest common version. In contrast, the* unique decoding *approach employed by Slepian-Wolf coding would require the decoder to obtain for some subset $S \subseteq \{u_1, u_2, \ldots, u_L\}$ such that $u_L \in S$, the unique $\mathbf{w}_S$ in the possible set that is consistent with the* *bin-indices; unique decoding, for instance, would not allow for correct decoding if there are multiple possible tuples even if they happen to have the same latest common version* $\mathbf{w}_{u_L}$. *The discussion in [58], which examined the necessity of non-unique decoding, motivates the following question: Can we use the decoding ideas of Slepian-Wolf - where all the messages are decoded - however, for an appropriately chosen subset of versions and have the same rates? In other words, if we take the union of the unique decoding rate regions over all possible subsets of* $\{\mathbf{W}_{u_1}, \mathbf{W}_{u_2}, \ldots, \mathbf{W}_{u_L}\}$, *does the rate allocation of (2.14), (2.15) lie in this region? The answer of this question is that non-unique decoding provides better rates than unique decoding in our case. This can be verified for Example 1 that we provide next, where* $c = 2, \nu = 3$ *and the first server receives all three versions and the second server only receives versions* 2 *and* 3.

Example 1. *We consider the case where* $\delta_K = \delta$, $c = 2$ *and* $\nu = 3$. *Consider the state where server* 2 *does not receive* $\mathbf{W}_1$. *Then, the storage allocation of our scheme is given by Table 2.2.*

Version	**Server 1**	**Server 2**
$\mathbf{W}_1$	$\frac{K + o(K)}{2}$	$-$
$\mathbf{W}_2$	$\frac{KH(\delta) + o(K)}{2}$	$\frac{K + KH(\delta) + o(K)}{2}$
$\mathbf{W}_3$	$\frac{KH(\delta) + o(K)}{2}$	$\frac{KH(\delta) + o(K)}{2}$

Table 2.2. Storage Allocation of Example 1.

We now examine unique decoding decoders that aim to recover $\mathbf{W}_3$. *It is clear* *that the decoder cannot recover the K bits of $\mathbf{W}_3$ without using side information, since the total number of bits of $\mathbf{W}_3$ stored is only $KH(\delta) + o(K)$.*

Now consider the case where a unique decoding based decoder uses the subset $\{\mathbf{W}_1, \mathbf{W}_3\}$. The rates $(R_1^{(1,3)}, R_3^{(1,3)})$ for a vanishing probability of error must satisfy

$$KR_3^{(1,3)} \geq KH(\delta * \delta) + o(K), \tag{2.37}$$

$$K(R_1^{(1,3)} + R_3^{(1,3)}) \geq K + KH(\delta * \delta) + o(K), \tag{2.38}$$

where $\delta * \delta = 2\delta(1-\delta)$. Note however that we store, in total,

$$KR_3 = KH(\delta) + o(K)$$

bits for $\mathbf{W}_3$, which is fewer than $KR_3^{(1,3)}$ for all δ.

Now consider the case where a unique decoding based decoder uses $\{\mathbf{W}_2, \mathbf{W}_3\}$ for decoding. In this case, the decoder requires

$$KR_3^{(2,3)} \geq KH(\delta) + o(K), \tag{2.39}$$

$$K(R_2^{(2,3)} + R_3^{(2,3)}) \geq K + KH(\delta) + o(K), \tag{2.40}$$

and we notice that

$$\begin{aligned} K(R_2 + R_3) &= K/2 + 2KH(\delta) + o(K) \\ &< K(R_2^{(2,3)} + R_3^{(2,3)}), \end{aligned}$$

for $\delta < H^{-1}(0.5)$.

Finally, consider the case where a unique decoding based decoder uses all three *versions* $\{\mathbf{W}_1, \mathbf{W}_2, \mathbf{W}_3\}$. *In this case, the rate tuples* $(R_1^{(1,2,3)}, R_3^{(1,2,3)})$ *have to satisfy seven inequalities, including the following inequalities*

$$K(R_1^{(1,2,3)} + R_3^{(1,2,3)}) \geq K + KH(\delta * \delta) + o(K) \tag{2.41}$$

Clearly,

$$K(R_1 + R_3) < K(R_1^{(1,2,3)} + R_3^{(1,2,3)}).$$

Thus, the union of the unique decoding rate regions for vanishing error probabilities, taken over all possible subsets of $\{\mathbf{W}_1, \mathbf{W}_2, \mathbf{W}_3\}$, *does not include the rate tuple of Table 2.2.*

Remark 4. *The deterministic code of Theorem 2 allows for erroneous decoding with probability at most* ϵ *and does not constitute a* 0*-error code; it can make an error for some subset of message tuples. For such a deterministic code construction, the probability of error is not computed over the randomization of the code, it is in fact computed over the distribution of the tuples of the message versions. Indeed, the main use of the probability measure over the tuple of the message versions corresponding to the* ν *versions in Section 2.3 is to bound the probability measure of the subset of erroneous message tuples. Specifically, under the (conditional) uniform measure in our system model, the deterministic code of Theorem 2 ensures* 0*-error decoding for at least* $1 - \epsilon$ *of the possible tuples. For this deterministic construction, there is a subset of message tuples which can be erroneous, but the probability measure of this subset under the conditional uniform measure of Section 2.3 is at most* ϵ.

A natural question is to ask whether a similar storage cost can be obtained if we want the probability of error to be 0, *i.e., if we want our decoder to correctly* *decode for* every *possible message tuple. The answer to this question in general is connected to the question of whether* 0*-error rate region and* ϵ*-error rate region are identical for our setting. There are several instances in network information theory where, even though there is no noise in the network the* ϵ*-error capacity is still larger than the* 0*-error capacity, (e.g., multiple access [52]). For some networks, the answer to this question is unknown and involves deep connections to other related questions [59]. Note also for distributed source coding setups such as our problem, the determination of the* 0*-error capacity is more complicated and involves the use of graph entropy [60]. In this work, we leave the question of whether the* 0*-error and* ϵ*-error rate regions are the same, open.*

2.5 Lower Bound on The Storage Cost (Theorem 3)

In this section, we extend the lower bound on the per-server storage cost of [7] for the case where we have correlated versions, and we require the probability of error to be at most ϵ.

Theorem 3. *[Storage Cost Lower Bound] An* ϵ*-error* $(n, c, \nu, 2^K, q, \delta_K)$ *multi-version code with correlated versions such that* $\mathbf{W}_1 \to \mathbf{W}_2 \to \ldots \to \mathbf{W}_\nu$ *form a Markov chain,* $\mathbf{W}_m \in [2^K]$ *and given* $\mathbf{W}_m$*,* $\mathbf{W}_{m+1}$ *is uniformly distributed in a Hamming ball of radius* $\delta_K K$ *centered around* $\mathbf{W}_m$ *must satisfy*

$$\log q \geq \frac{K + (\nu - 1)\log Vol(\delta_K K, K)}{c + \nu - 1} + \frac{\log(1 - \epsilon 2^{\nu n}) - \log \binom{c+\nu-1}{\nu}\nu!}{c + \nu - 1}, \qquad (2.42)$$

where $\epsilon < 1/2^{\nu n}$*.*

Proof of Theorem 3 for $\nu = 2$*.* Consider any ϵ-error $(n, c, 2, 2^K, q, \delta_K)$ multi-version code, and consider the first c servers, $T = [c]$, for decoding. We recall that the set of possible tuples A_{δ_K} is partitioned into disjoint sets as follows

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

where $A_{\delta_K,1}$ is the set of tuples $(\mathbf{w}_1, \mathbf{w}_2) \in A_{\delta_K}$ for which we can decode successfully for all $\mathbf{S} \in \mathcal{P}([\nu])^n$ and $A_{\delta_K,2}$ is the set of tuples where we cannot decode successfully at least for one state $\mathbf{S} \in \mathcal{P}([\nu])^n$, which can be expressed as follows

$$A_{\delta_K,2} = \bigcup_{\mathbf{S} \in \mathcal{P}([\nu])^n} A^{(\mathbf{S})}_{\delta_K,2}, \tag{2.43}$$

where $A^{(\mathbf{S})}_{\delta_K,2}$ is the set of tuples for which we cannot decode successfully given a particular state $\mathbf{S} \in \mathcal{P}([\nu])^n$. Consequently, we have

$$|A_{\delta_K,2}| \leq \sum_{\mathbf{S} \in \mathcal{P}([\nu])^n} |A^{(\mathbf{S})}_{\delta_K,2}|. \tag{2.44}$$

For any state $\mathbf{S} \in \mathcal{P}([\nu])^n$, we require the probability of error, P_e, to be at most ϵ. Since all tuples in the set A_{δ_K} are equiprobable, we have

$$P_e = \frac{|A^{(\mathbf{S})}_{\delta_K,2}|}{|A_{\delta_K}|} \leq \epsilon. \tag{2.45}$$

Therefore, we have

$$\begin{aligned} |A_{\delta_K,1}| &= |A_{\delta_K}| - |A_{\delta_K,2}| \\ &\geq |A_{\delta_K}| - \sum_{\mathbf{S} \in \mathcal{P}([\nu])^n} |A^{(\mathbf{S})}_{\delta_K,2}| \\ &\geq |A_{\delta_K}| - \sum_{\mathbf{S} \in \mathcal{P}([\nu])^n} \epsilon |A_{\delta_K}| \\ &\geq |A_{\delta_K}|(1 - \epsilon 2^{\nu n}). \end{aligned} \tag{2.46}$$

Suppose that $(\mathbf{W}_1, \mathbf{W}_2) \in A_{\delta_K,1}$. Because of the decoding requirements, if $\mathbf{W}_1$ is available at all servers, the decoder must be able to obtain $\mathbf{W}_1$ and if $\mathbf{W}_2$ is available at all servers, the decoder must return $\mathbf{W}_2$. Hence, as shown in [7], there exist $\mathbf{S}_1, \mathbf{S}_2 \in \mathcal{P}([\nu])^n$ such that

- $\mathbf{S}_1$ and $\mathbf{S}_2$ differ only in the state of one server indexed by $B \in [c]$, and
- $\mathbf{W}_1$ can be recovered from the first c servers in state $\mathbf{S}_1$ and $\mathbf{W}_2$ can be recovered from the first c servers in $\mathbf{S}_2$.

Therefore both $\mathbf{W}_1$ and $\mathbf{W}_2$ are decodable from the c codeword symbols of the first c servers in state $\mathbf{S}_1$, and the codeword symbol of the B-th server in state $\mathbf{S}_2$. Thus, we require the following

$$\begin{aligned} c\, q^{c+1} &\geq |A_{\delta_K,1}| \\ &\geq |A_{\delta_K}|(1 - \epsilon 2^{\nu n}). \end{aligned} \tag{2.47}$$

We also have

$$|A_{\delta_K}| = 2^K Vol(\delta_K K, K). \tag{2.48}$$

Therefore, the storage cost is lower-bounded as follows

$$\log q \geq \frac{K + \log Vol(\delta_K K, K)}{c+1} + \frac{\log(1 - \epsilon 2^{\nu n}) - \log c}{c+1}. \tag{2.49}$$

□

We now provide a proof sketch for the case where $\nu \geq 3$.

Proof sketch of Theorem 3 for $\nu \geq 3$. Consider any ϵ-error $(n, c, \nu, 2^K, q, \delta_K)$ multi-version code, and consider the first $c \leq n$ servers, $T = [c]$, for decoding. Suppose we have ν versions $\mathbf{W}_{[\nu]} = (\mathbf{W}_1, \mathbf{W}_2, \cdots, \mathbf{W}_\nu)$. Suppose that $\mathbf{W}_{[\nu]} \in A_{\delta_K,1}$. We construct auxiliary variables $Y_{[c-1]}, Z_{[\nu]}, B_{[\nu]}$, where $Y_i, Z_j \in [q], i \in [c-1], j \in [\nu]$, $1 \leq B_1 \leq \cdots \leq B_\nu \leq c$ and a permutation $\Pi : [\nu] \to [\nu]$, such that there is a bijection mapping from these variables to $A_{\delta_K,1}$. In order to construct these auxiliary variables, we use the algorithm of [7]. Therefore, we have

$$\begin{aligned} q^{c+\nu-1}\binom{c+\nu-1}{\nu}\nu! &\geq |A_{\delta_K,1}| \\ &> |A_{\delta_K}|(1-\epsilon 2^{\nu n}), \end{aligned} \tag{2.50}$$

where the first inequality follows since $Y_i, Z_j \in [q]$, there are at most $\binom{c+\nu-1}{\nu}$ possibilities of $B_{[\nu]}$ and at most $\nu!$ possible permutations. We also have

$$|A_{\delta_K}| = 2^K Vol(\delta_K K, K)^{(\nu-1)}. \tag{2.51}$$

Therefore, the storage cost is lower-bounded as follows

$$\log q \geq \frac{K + (\nu-1)\log Vol(\delta_K K, K)}{c+\nu-1} + \frac{\log(1-\epsilon 2^{\nu n}) - \log\binom{c+\nu-1}{\nu}\nu!}{c+\nu-1}. \tag{2.52}$$

□

2.6 Conclusion

In this chapter, we have proposed multi-version codes to efficiently store correlated updates of data in a decentralized asynchronous storage system. These constructions are based on Reed-Solomon codes and random binning. An outcome of our results is that the correlation between versions can be used to reduce storage costs in asynchronous decentralized systems, even if there is no single server or client node who is aware of all data versions, in applications where consistency is important. In addition, our converse result shows that these constructions are within a factor of 2 from the information-theoretic optimum in certain interesting regimes. The development of practical coding schemes for the case where δ_K is known a priori is an open research question, which would require non-trivial generalization of previous code constructions for the Slepian-Wolf problem [38, 39]. Extending our study beyond the simple Hamming-based correlation model studied in this work is also an interesting future research direction.

Chapter 3
Fundamental Limits of Erasure-Coded Key-Value Stores with Side Information

Previous work on multi-version coding considered a completely decentralized asynchronous system where the nodes (servers) are not aware of which updates (versions) of the data are received by the other nodes. In this chapter, we relax this assumption and study a system where a node acquires side information of the versions propagated to some other nodes based on the network topology. Specifically, we study a storage system with n nodes over a graph that store ν totally ordered versions of an object (message). Each node receives a subset of these ν versions. A node is aware of which versions received by its neighbors in the network graph. Our code constructions show that the side information result in a better storage cost as compared with the case where the nodes do not exchange side information for some regimes at the expense of the additional latency and the negligible communication overhead of exchanging the side information. Through an information-theoretic converse, we identify scenarios where exchanging tremendous amount of side information does not reduce the storage cost. Finally, we present a case study over Amazon web services (AWS) that demonstrates the potential storage cost reductions of our code constructions.

3.1 Introduction

The use of erasure codes in key-value stores faces unique challenges that do not appear in archival storage due to the fact that in key-value stores, *write* operations update the data frequently, and *read* operations aspire to get recently updated version of the data. Recall that an (n, k) code - where n is the code length and k is the dimension of the code - partitions the value to be stored into k fragments and encodes them into n coded fragments and each node stores one coded fragment. Using maximum distance separable (MDS) codes ensures that the value can be recovered from any k of the n nodes. Notice that for $k > 1$, there is no single node that stores the data entirely. When such a code is used in a strongly consistent key-value store, a node that receives an updated version of the data cannot immediately delete the older versions of the data. The node has to effectively wait for a sufficient number of nodes to receive the update before deleting the old version of the data. There are two challenges associated with this problem,

- the storage cost benefit of erasure coding diminishes with the need to store older versions, or
- a node has to discover the versions received by the other nodes.

There are two approaches to overcome these challenges in distributed algorithms literature. A common approach is to allow nodes to store a fixed number of versions of the data, so that in the duration of the propagation of the new version, there is indeed a coherent version that can be decoded. This approach obviates the need to discover what versions are received by other nodes. In such algorithms [31, 61–64], it is shown that each node has to store a number of versions that grows with the degree of concurrency - the maximum number of concurrent writes to the same object. However, storing older versions offsets the storage benefits of erasure coding. The second approach employed by protocols [65, 66] is the use of message broadcast primitives where the nodes exchange the data versions, find out the versions received by the other nodes and delete the older versions; in this manner, each node stores exactly one version of the data[1]. However, these message broadcast primitives that exchange the data versions among every pair of nodes incur significant costs, specifically: (1) latency cost of waiting to receive the data version before completing the write operation, and (2) the communication cost of exchanging these data versions. These costs can be especially exaggerated in geo-distributed settings where the communication cost of exchanging large data is expensive [67]. In addition, exchanging data between two very distant nodes - say one in Seoul and one in Paris- can lead to significant latency increase due to the distance [68]. Our work is as such motivated to address the following question: if a node exchanges meta-data[2] information of which versions it receives with nearby nodes, to what extent can the storage cost be reduced? We assume a known network topology graph among the nodes and that the nodes exchange meta-data “side information” of the versions received with their neighbors in the graph, and conduct an information-theoretic study on the achievable storage cost[3].

[1] Even in [65, 66], the nodes have to temporarily store older versions while the broadcast is being performed. Note that this temporary storage growth is shown to be inevitable [26, 27]; however the point is that at the cost of latency and communication, [65, 66] achieve the minimum possible storage cost, whereas the stable state storage cost of [31, 61–64] is larger.

[2] Throughout this chapter, we use the terms *side* information and *meta-data* information interchangeably.

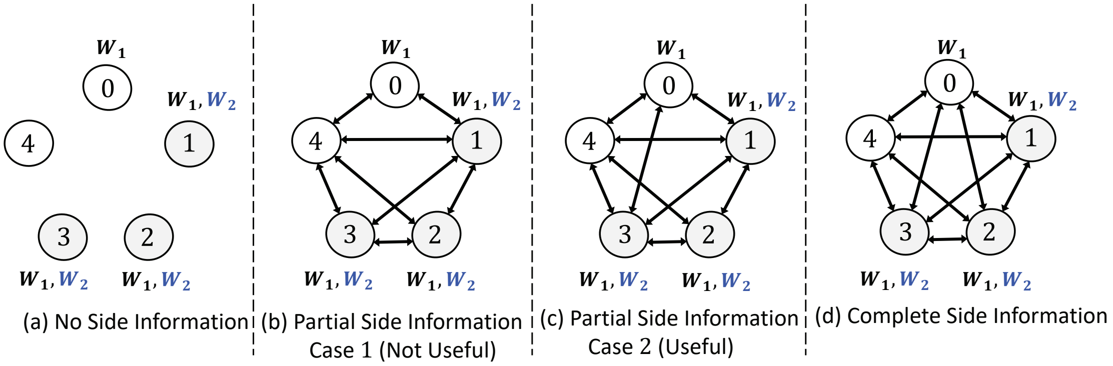
[3] This side information is in the order of bytes, whereas the data itself can be in the order of Megabytes. Hence, the side information communication cost of our approach is negligible as compared with the approaches where the nodes share the data versions or the coded data versions as in [66] which is quite expensive [67].

3.2 Background and Summary of Contributions

We describe our contribution through the *multi-version coding* framework [7] described in Ch. 1.4. We begin by high-level description of the multi-version coding framework is provided in Sec. 3.2.1. Sec. 3.2.2 then summarizes our contributions.

3.2.1 Multi-Version Coding

We recall that multi-version coding framework [7], discussed in Ch. 1.4, considers a distributed storage system of n nodes storing ν totally ordered versions $\mathbf{W}_1, \mathbf{W}_2, \cdots, \mathbf{W}_\nu$ of an object of size K bits. The higher ordered versions are interpreted as later versions, and lower ordered versions as earlier versions. Due to the inherent asynchrony in the system, each node may receive an arbitrary subset of these ν versions referred to as the state of that node. Any version that has been propagated to at least c_W nodes is referred to as a complete version, and the goal at the decoder is to connect to an arbitrary subset of c_R nodes and decode the latest complete version - the complete version with the highest order - or a later version. For any complete version and for any set of c_R nodes, there are at least $c \coloneqq c_W + c_R - n$ nodes that have received that version. In the classical erasure coding model, where $\nu = 1$, the Singleton bound implies that the storage cost per node is at least K/c. However, for $\nu > 1$, a node cannot simply store the codeword symbol corresponding to one version, since other nodes may not have received that version. In fact, the lower bound of [7] implies that the amount of data to be stored per node is at least $\frac{\nu K}{c+\nu-1} - \theta(1) = (\frac{\nu}{c} - \frac{\nu(\nu-1)}{c^2} + o(\frac{1}{c^2}))K$. That is, there is a cost to be paid for the decentralized nature and the asynchrony in the system, and this storage cost translates approximately to storing $\nu/2$ versions of the data, each with an MDS code of dimension c. We next provide an example as depicted in Fig. 3.1 to illustrate the code construction developed in [7].



(a) No Side Information

(b) Partial Side Information Case 1 (Not Useful)

(c) Partial Side Information Case 2 (Useful)

(d) Complete Side Information

Figure 3.1. A key-value store with $n = 5$ nodes and $c_W = c_R = 4$ is considered. Nodes $1, 2, 3$ have received both versions, node 0 has received only version 1 and node 4 has not received any version. In this state, version 1 is the latest complete version as it has been received by $c_W = 4$ nodes. Without side information, all nodes do not know that version 2 is incomplete. With complete side information, all nodes know that version 2 is incomplete. In a partial side information case 2, node 2 does not know that version 2 is incomplete as it knows that version 2 is received by at least 3 nodes and does not know whether node 0 has received version 2 or not. In a partial side information case 1, nodes 2 and 3 do not know that version 2 is incomplete.

Motivating Example of Fig. 3.1: Consider the setting of Fig. 3.1 with $n = 5$ nodes, $\nu = 2$ versions and $c_W = c_R = 4$. Each server may receive any subset of these two versions, so each server may be in $2^2 = 4$ states, and the system can be in one of $2^{\nu n} = 1024$ states; one such system (global) state is depicted in Fig. 3.1. The goal is for each server is to devise an encoding function with knowledge of only its local state, such that, for each of the 1024 system states, the latest version that has propagated to at least $c_W = 4$ nodes (or a later version) must be decodable from every set of $c_R = 4$ servers. In the state shown in Fig. 3.1-(a), $\mathbf{W}_1$ has propagated to 4 servers and is the latest complete version. So, from each of the $\binom{5}{4} = 5$ possible sets of 4 nodes in the system, $\mathbf{W}_1$ or $\mathbf{W}_2$ must be recoverable.

The storage scheme of replication involves each server storing the latest version it receives and is naturally a feasible strategy that incurs a storage cost of K bits. A simple erasure coding scheme involves each server storing $K/c = K/3$ bits of each version it receives suffices, as every subset of $c_R = 4$ servers has at least $c = 3$ servers that have the latest complete version. Since there are two versions, this leads to a worst-case storage cost of $2K/3$ bits per server. However, the multi-version code of [7] has a storage cost of only $K/2$ bits. In this strategy, each server stores $K/2$ bits of the *latest* version it receives, instead of storing the entire object as in replication or both versions as in the simple erasure coding approach. For instance, in the state shown in Fig. 3.1-(a), servers $1, 2, 3$ store $K/2$ bits of $\mathbf{W}_2$ via a linear code of dimension 2, and server 0 stores $K/2$ bits of $\mathbf{W}_1$. It can be seen that from every set of $c_R = 4$ servers, $\mathbf{W}_2$ can be decoded as each set of 4 servers includes at least 2 servers which store $\mathbf{W}_2$. Through pigeon-hole based arguments, [7] showed that this coding strategy works in all states and showed that this code is approximately optimal via network-information-theoretic converse arguments. We will revisit this example in the context of our contributions.

3.2.2 Contributions

While the multi-version coding problem [7] opens the door to information-theoretic analysis of key-value stores, its assumptions are arguably too conservative. Specifically, the multi-version coding problem assumes independent versions of the data and that each node is completely unaware of the versions received by any other nodes (i.e., completely decentralized). However, in real-world key-value stores, these assumptions can be relaxed. In fact, reference [32] has studied the case where the data versions are correlated, and developed code constructions based on Reed

Solomon codes and random binning techniques that obtained significant storage gains. In this work, we extend the scope of the multi-version coding framework along a different direction motivated by real-world considerations of geo-distributed key-value stores where the system is not completely decentralized as the nodes can exchange data depending on the network topology, latency requirements and communication costs.

Specifically, we allow nodes to receive side information of the states of some other nodes based on the topology and study the impact of this side information on the storage cost. We represent the side information by a directed graph $\mathcal{G} = (\mathcal{N}, \mathcal{E})$, where an edge $e_{ij} \in \mathcal{E}$ from vertex i to vertex j indicates that node i is aware of the state of node j as shown in Fig. 3.1. The multi-version coding setting discussed in Sec. 3.2.1 refers to a completely decentralized setting, where $\mathcal{G} = (\mathcal{N}, \mathcal{E})$ has only self-edges. In the completely centralized case where each node is aware of the states of all nodes, that is $\mathcal{G} = (\mathcal{N}, \mathcal{E})$ is a complete digraph as shown in Fig. 3.1-(d), each node is aware of the latest complete version. In this case, a node that receives the latest complete version stores it with an MDS code of dimension c and the storage cost is $\frac{1}{c}K$. We provide results that depart from the two extreme points - the completely decentralized setting and the completely centralized setting - and bridge the gap between them. Specifically, our contributions are as follows.

1. In Section 3.4, we develop a graph-theoretic approach for this side information problem and provide code constructions that show that this side information can reduce the worst-case storage cost significantly as compared with the case where nodes do not share their states. In particular, for a given side information graph $\mathcal{G}$, we identify a graph functional, that we refer to as the size of the *complement of the smallest c_W-maximally externally connected*

subset - or simply, the size of the c_W-CSMECS - denoted by $\overline{m}_{\mathcal{G}}$, which dictates the storage cost of our code constructions. Specifically, our construction has a storage cost of

$$\left(\frac{1}{c} + \frac{(\nu-1)\overline{m}_{\mathcal{G}}(c_W)}{c^2} + o\left(\frac{\overline{m}_{\mathcal{G}}(c_W)}{c^2}\right)\right) K.$$

In the completely decentralized setting, $\overline{m}_{\mathcal{G}}(c_W) = c_W - 1$, whereas for a complete digraph network topology, $\overline{m}_{\mathcal{G}}(c_W) = 0$. For a regular side information graph $\mathcal{G}$ with degree H, we show that

$$\overline{m}_{\mathcal{G}}(c_W) \leq \min((n - c_W + 1)(n - H), c_W - 1),$$

which leads to an achievable scheme on the storage cost. The development of achievable scheme overcomes two challenges (i) we require achievable schemes to return consistent data in all $2^{n\nu}$ states, and (ii) the potential asymmetry in the encoding strategies due to asymmetry in the side information which leads to much larger state space as compared with the completely decentralized setting. For instance, consider the graph of Fig. 3.1-(c), where our results show that using the side information, the storage cost can be reduced to 4/9 K, as compared with 1/2 K in the completely decentralized setting. Interestingly, for the state considered in Fig. 3.1-(c), server 2 has a different storage strategy as compared with servers 1 and 3, even though they are in the same state as they all have received both versions, due to the differences in the side information they observe; this is discussed in Example 5 and Table 3.1 in Section 3.4.

2. We provide information-theoretic lower bounds on the storage cost for the

case of $\nu = 2$ and identify a curious outcome of these results. Specifically, we identify a scenario where each node is aware of the versions received by $(n - 3)$ other nodes, and yet the side information does not help in improving the worst-case storage cost. This surprising result can be noted in Fig. 3.1-(b) which is obtained from just removing one edge as compared with Fig. 3.1-(c). Our converse implies that there is no possible improvement in storage cost as compared with the completely decentralized case shown in Fig 3.1-(a). That is, the optimal storage cost of Fig. 3.1-(b) is in fact $K/2$ bits despite 8 pairs of servers exchanging side information among themselves. These results indicate that a careful understanding of the topology is required to completely exploit the side information in distributed key value stores.

3.2.3 Case Study

Several services implement their own key-value stores using public cloud services, for instance, Overleaf [69] uses Amazon S3 [70]. The development of a full-fledged protocol that use our code constructions is outside the scope of our work. However, we conduct a case study in Section 3.6.2 illustrating the potential storage cost savings of using our code constructions assuming a hypothetical key-value store implementation over Amazon web services (AWS) public cloud. Although our constructions would also be relevant for private/commercial key-value stores such as DynamoDB, the fact that the pricing information of public clouds is readily available enables us to realistically understand the potential utility of our contributions. While the side information can offer significant storage cost savings, it also incurs a negligible communication cost of exchanging that tags of the versions (version numbers) and additional latency required for the nodes to exchange these tags.

In the case where the nodes do not exchange their states (completely decentralized system), the nodes do not wait before deciding what to store. In the case where the nodes exchange side information, the nodes wait for the side information of the other nodes to be received before deciding what to store. Therefore, there is an additional latency cost associated with exchanging the side information. Allowing higher latency, may allow more distant nodes to exchange side information and that can reduce the storage cost. Based on the maximum additional latency that the system can tolerate and the latencies between data centers, we construct a side information graph $\mathcal{G}$ such that an edge exists from a data center $i \in \mathcal{N}$ to a data center $j \in \mathcal{N}$, if the latency from the j-th data center to the i-th data center is below the maximum additional latency allowed in the system. We calculate the size of the c_W-CSMECS, $\overline{m}_{\mathcal{G}}(c_W)$, for the generated graph and show that using side information and our code constructions lead to significant storage cost savings. We illustrate this latency-storage trade-off in our case study in Section 3.6.2.

3.3 System Model

We start with the notation. For a positive integer i, we denote by $[i]$ the set $\{1, 2, \cdots, i\}$. For any set of ordered indices $S = \{s_1, s_2, \cdots, s_{|S|}\} \subseteq \mathbb{Z}$, where $s_1 < s_2 < \cdots < s_{|S|}$, and for any ensemble of variables $\{X_i : i \in S\}$, the tuple $(X_{s_1}, X_{s_2}, \cdots, X_{s_{|S|}})$ is denoted by X_S. We use $\log(.)$ to denote the logarithm to the base 2 and $H(.)$ to denote the binary entropy function. We use the notation $[2^K]$ to denote the set of K-length binary strings. A *code* of length n and dimension k over alphabet $\mathcal{A}$ consists of an injective mapping $\mathcal{C} : \mathcal{A}^k \rightarrow \mathcal{A}^n$. When $\mathcal{A}$ is a finite field and the mapping $\mathcal{C}$ is linear, then the code is referred to as a *linear code*. A linear code $\mathcal{C}$ of length n and dimension k is referred to as (n, k) code. An (n, k)

linear code is referred to as maximum distance separable (MDS) if the mapping projected to *any* k co-ordinates is invertible. In a directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, the in-degree and the out-degree of of a vertex $v \in \mathcal{V}$ are denoted by $\deg^{-}_{\mathcal{G}}(v)$ and $\deg^{+}_{\mathcal{G}}(v)$ respectively.

We now present a variant of the multi-version coding problem [7], discussed in Ch. 1.4, where we model exchanging the side information between the nodes. We study a storage system of n nodes, denoted by $\mathcal{N}$, that can tolerate f failures[4]. The objective of the system is to store ν independent totally ordered versions of a K bits message. The j-th version of the message is denoted by $\mathbf{W}_j \in [2^K]$, where $j \in [\nu]$. If $i < j$, we interpret $\mathbf{W}_j$ as a later version with respect to $\mathbf{W}_i$. The i-th node receives an arbitrary subset of versions $\mathbf{S}(i) \subseteq [\nu]$ that denotes its *state*. We denote the system state by $\mathbf{S} \in \mathcal{P}([\nu])^n$, where $\mathcal{P}([\nu])$ denotes the power set of $[\nu]$. In state $\mathbf{S}$, we denote the set of nodes that have received version $u \in [\nu]$ by $\mathcal{A}_{\mathbf{S}}(u)$. A version that is received by at least $c_W \leq (n - f)$ nodes is referred to as a *complete* version. The set of complete versions in state $\mathbf{S} \in \mathcal{P}([\nu])^n$ is given by

$$\mathcal{C}_{\mathbf{S}} \coloneqq \{u \in [\nu] : |\mathcal{A}_{\mathbf{S}}(u)| \geq c_W\} \tag{3.1}$$

and the latest complete version is denoted by $L_{\mathbf{S}} \coloneqq \max\ \mathcal{C}_{\mathbf{S}}$. The decoder connects to any $c_R \leq (n - f)$ nodes and must decode a version $u \in [\nu]$ such that $u \geq L_{\mathbf{S}}$. We notice that among these c_R nodes, any complete version is present at least at $c \coloneqq c_W + c_R - n$ nodes.

A node is aware of the states of some other nodes in the network based on the topology. Sharing the states among the nodes is specified by the *side information graph*. The side information graph $\mathcal{G} = (\mathcal{N}, \mathcal{E})$ is a directed graph, where the set

[4] By failures, we refer to nodes that halt and do not respond.

of the vertices represent the nodes and an edge $e_{ij} \in \mathcal{E}$ from vertex i to vertex j indicates that node i is aware of the state of node j. Based on the side information graph, node i obtains the states of some nodes in the system. This set of nodes is referred to as the neighborhood of node i that is given by

$$\mathcal{H}_i = \{j \in \mathcal{N} : e_{ij} \in \mathcal{E}\}. \tag{3.2}$$

We denote the states of the nodes in $\mathcal{H}_i$ by $\mathbf{S}(\mathcal{H}_i)$. Node i stores a symbol from $[q]$ based on the versions that it receives $\mathbf{W}_{\mathbf{S}(i)}$ and the local side information $\mathbf{S}(\mathcal{H}_i)$. We next define the multi-version code with side information formally.

Definition 8 (Multi-version code with side information)**.**
A $(\mathcal{G} = (\mathcal{N}, \mathcal{E}), c_W, c_R, \nu, 2^K, q)$ multi-version code with side information consists of the following

- *encoding functions*

$$\begin{aligned} &\varphi^{(i)}_{\mathbf{S}(\mathcal{H}_i)} \colon [2^K]^{|\mathbf{S}(i)|} \to [q], \\ &\textit{for every } i \in \mathcal{N} \textit{ and every } \mathbf{S}(\mathcal{H}_i) \subseteq \mathcal{P}([\nu])^{|\mathcal{H}_i|}, \end{aligned}$$

- *decoding functions*

$$\psi^{(\mathcal{R})}_{\mathbf{S}} \colon [q]^{c_R} \to [2^K] \cup \{NULL\},$$

that satisfy the following

$$\psi_{\mathbf{S}}^{(\mathcal{R})}\left(\varphi_{\mathbf{S}(\mathcal{H}_{t_1})}^{(t_1)},\cdots,\varphi_{\mathbf{S}(\mathcal{H}_{t_{c_R}})}^{(t_{c_R})}\right)=\begin{cases}\mathbf{W}_m & \textit{for some } m \geq L_{\mathbf{S}}, \\ & \textit{if } \mathcal{C}_{\mathbf{S}} \neq \emptyset, \\ NULL & \textit{otherwise,}\end{cases} \tag{3.3}$$

for every possible system state $\mathbf{S} \in \mathcal{P}([\nu])^n$, every $\mathbf{W}_{[\nu]} \in [2^K]^\nu$ and every set of nodes $\mathcal{R} \subseteq \mathcal{N}$, where $\mathcal{R} = \{t_1, t_2, \cdots, t_{c_R}\}$, such that $t_1 < t_2 < \cdots < t_{c_R}$.

The objective of multi-version coding side information problem is minimizing the storage cost that we define next.

Definition 9 (Storage cost)**.** *The storage cost of a $(\mathcal{G} = (\mathcal{N}, \mathcal{E}), c_W, c_R, \nu, 2^K, q)$ multi-version code is equal to $\alpha = \log q$ bits.*

As we explained in Chapter 1, it was shown in [7] that the storage cost for the case where the versions are independent and nodes do not share their states, that is $\mathcal{H}_i = \{i\}, i \in \mathcal{N}$, is lower-bounded as follows

$$\log q \geq \frac{\nu}{c+\nu-1}K - \frac{\log\left(\nu^\nu\binom{c+\nu-1}{\nu}\right)}{(c+\nu-1)}. \tag{3.4}$$

In addition, a code construction was developed with storage cost

$$\alpha = \frac{\nu}{c+\nu-1}K, \tag{3.5}$$

for the case where $\nu|(c-1)$.

3.4 Coding With Side Information

In this section, we describe achievable schemes showing that the side information can reduce the worst-case storage cost. We start with explaining the intuition behind our code construction in Section 3.4.1. In Section 3.4.2, we provide a graph-theoretic interpretation of our code constructions through developing the notion of the maximally externally connected subset of a graph. Based on this interpretation, we develop our code constructions in Section 3.4.3.

3.4.1 Code Constructions Intuition

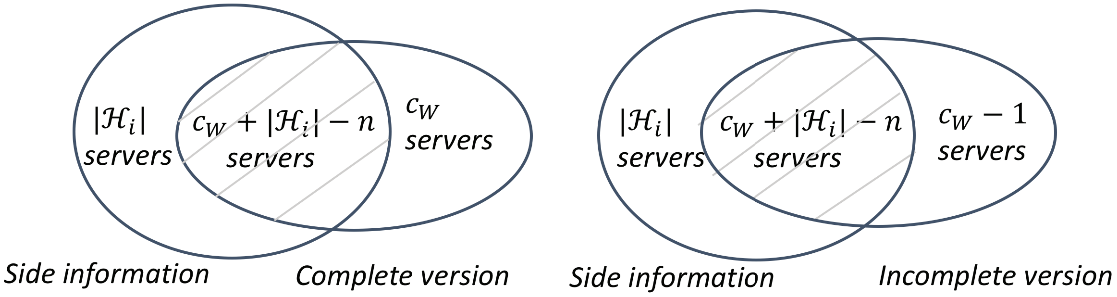

$|\mathcal{H}_i|$ servers
$c_W + |\mathcal{H}_i| - n$ servers
c_W servers
Side information
Complete version
$|\mathcal{H}_i|$ servers
$c_W + |\mathcal{H}_i| - n$ servers
$c_W - 1$ servers
Side information
Incomplete version

Figure 3.2. The intuition behind the code constructions.

We begin by explaining the intuition behind the construction as depicted in Fig. 3.2. We intend to provide each node with methodology to distinguish between complete and incomplete versions. Note that the greater the connectivity of the graph, the more reliably a node can “guess” the set of complete versions and improve the storage cost. Consider a state $\mathbf{S} \in \mathcal{P}([\nu])^n$, where version $u \in [\nu]$ is complete. Suppose that the i-th node has received version u, i.e., $i \in \mathcal{A}_{\mathbf{S}}(u)$, then this node observes through its local side information at least $c_W + |\mathcal{H}_i| - n$ nodes having u. However, the converse may not be true. That is, there may exist an *incomplete* version u' such that a node that receives u' observes $c_W + |\mathcal{H}_i| - n$ or more nodes having u' through its side information. In particular, this can occur when $|\mathcal{H}_i| < n$, that is, when the graph is incomplete[5].

In our strategy, a node that observes at least $c_W + |\mathcal{H}_i| - n$ nodes having version u assumes that u is a complete version. For any given incomplete version u, the maximum number of nodes that mistakenly assume that u is complete is given by

$$\overline{m}_{\mathcal{G}}(c_W) \coloneqq \max_{\mathbf{S} \in \mathcal{P}([\nu])^n : \mathcal{A}_{\mathbf{S}}(u) \leq (c_W - 1)} |\{i \in \mathcal{A}_{\mathbf{S}}(u) : |\mathcal{A}_{\mathbf{S}}(u) \cap \mathcal{H}_i| \geq c_W + |\mathcal{H}_i| - n\}|. \tag{3.6}$$

Intuitively, propagating an incomplete version u to more nodes such that it remains incomplete cannot decrease the number of nodes that mistakenly assume that u is complete. In fact, the optimization problem in (3.6) can be performed over $\mathbf{S} \in \mathcal{P}([\nu])^n$ such that $\mathcal{A}_{\mathbf{S}}(u) = (c_W - 1)$. In order to see this, consider two states $\mathbf{S}', \mathbf{S} \in \mathcal{P}([\nu])^n$ such that $\mathcal{A}_{\mathbf{S}'}(u) \subset \mathcal{A}_{\mathbf{S}}(u)$ and u is incomplete in both states. Consider a node $i \in \mathcal{A}_{\mathbf{S}'}(u)$ such that $|\mathcal{A}_{\mathbf{S}'}(u) \cap \mathcal{H}_i| \geq c_W + |\mathcal{H}_i| - n$. Since $\mathcal{A}_{\mathbf{S}'}(u) \subset \mathcal{A}_{\mathbf{S}}(u)$, then $|\mathcal{A}_{\mathbf{S}}(u) \cap \mathcal{H}_i| \geq c_W + |\mathcal{H}_i| - n$. In other words, if node i assumes that version u is complete in $\mathbf{S}'$, it will also assume that u is complete in $\mathbf{S}$. Thus, we have

$$\begin{aligned} &\{i \in \mathcal{A}_{\mathbf{S}'}(u) : |\mathcal{A}_{\mathbf{S}'}(u) \cap \mathcal{H}_i| \geq c_W + |\mathcal{H}_i| - n\} \\ &\subseteq \{i \in \mathcal{A}_{\mathbf{S}}(u) : |\mathcal{A}_{\mathbf{S}}(u) \cap \mathcal{H}_i| \geq c_W + |\mathcal{H}_i| - n\}. \end{aligned}$$

Hence, the maximum corresponds to the case where $u \notin \mathcal{C}_{\mathbf{S}}$ is received by exactly

[5] Throughout this chapter, we assume that each vertex in the side information graph has a self-edge.

$c_W - 1$ nodes and $\overline{m}_{\mathcal{G}}(c_W)$ can be simplified as follows

$$\overline{m}_{\mathcal{G}}(c_W) = \max_{\mathbf{S} \in \mathcal{P}([\nu])^n : \mathcal{A}_{\mathbf{S}}(u) = (c_W - 1)} |\{i \in \mathcal{A}_{\mathbf{S}}(u) : |\mathcal{A}_{\mathbf{S}}(u) \cap \mathcal{H}_i| \geq c_W + |\mathcal{H}_i| - n\}|. \tag{3.7}$$

In fact, the quantity $\overline{m}_{\mathcal{G}}(c_W)$ motivates a graph functional that we term the size of *the complement of the smallest c_W-maximally externally connected subset of a graph (c_W-CSMECS)*, that we define in Section 3.4.2 in the context of an abstract graph $\tilde{\mathcal{G}}$. In Section 3.4.3, we return to the multi-version coding with side information problem and describe code constructions whose storage cost can be derived based on the size of the c_W-CSMECS of the side information graph.

3.4.2 The CSMECS of a Graph

In this subsection, we develop the concept of maximally externally connected subset of a graph $\tilde{\mathcal{G}} = (\tilde{\mathcal{V}}, \tilde{\mathcal{E}})$. We use the tilde notation here to distinguish an arbitrary graph used in this subsection from the side information graph $\mathcal{G}$ that arises in the multi-version coding problem.

Consider a directed graph $\tilde{\mathcal{G}} = (\tilde{\mathcal{V}}, \tilde{\mathcal{E}})$. A vertex $u \in \tilde{\mathcal{V}}$ is an in-neighbor of a vertex $v \in \tilde{\mathcal{V}}$ if $(u, v) \in \tilde{\mathcal{E}}$ and an out-neighbor of v if $(v, u) \in \tilde{\mathcal{E}}$. We use $\mathcal{N}^+_{\tilde{\mathcal{G}}}(v)$ to denote the set of out-neighbors and $\mathcal{N}^-_{\tilde{\mathcal{G}}}(v)$ to denote the set of in-neighbors of $v \in \tilde{\mathcal{V}}$. An induced subgraph $\tilde{\mathcal{G}}[\tilde{\mathcal{V}}'] \subseteq \tilde{\mathcal{G}}$ is a subgraph of $\tilde{\mathcal{G}}$ formed by a subset of the vertices $\tilde{\mathcal{V}}' \subseteq \tilde{\mathcal{V}}$ and all edges connecting pairs of vertices in $\tilde{\mathcal{V}}'$. Consider a graph $\tilde{\mathcal{G}} = (\tilde{\mathcal{V}}, \tilde{\mathcal{E}})$ and any induced subgraph $\tilde{\mathcal{G}}[\tilde{\mathcal{U}}]$, where $\tilde{\mathcal{U}} \subseteq \tilde{\mathcal{V}}$. The out-degree of any vertex $u \in \tilde{\mathcal{U}}$ is lower-bounded as follows

$$\deg^+_{\tilde{\mathcal{G}}[\tilde{\mathcal{U}}]}(u) := |\mathcal{N}^+_{\tilde{\mathcal{G}}}(u) \cap \tilde{\mathcal{U}}|$$

$$\geq \deg^{+}_{\tilde{\mathcal{G}}}(u) + |\tilde{\mathcal{U}}| - |\tilde{\mathcal{V}}|. \tag{3.8}$$

Given a subgraph $\tilde{\mathcal{G}}' = (\tilde{\mathcal{V}}', \tilde{\mathcal{E}}') \subset \tilde{\mathcal{G}}$, where $|\tilde{\mathcal{V}}'| = (s-1)$, an s-maximally externally connected subset (s-MECS) is a subset of $\tilde{\mathcal{V}}'$ such that each vertex is connected to more than $|\tilde{\mathcal{V}}| - s$ vertices outside $\tilde{\mathcal{V}}'$, that is in $\tilde{\mathcal{V}} - \tilde{\mathcal{V}}'$. That is, an s-MECS is the set of all vertices such that each vertex is connected to every vertex outside $\tilde{\mathcal{V}}'$. The *s-SMECS of a graph $\tilde{\mathcal{G}}$* is the *smallest* s-MECS, where the minimization is over all possible subgraphs $\tilde{\mathcal{G}}' = (\tilde{\mathcal{V}}', \tilde{\mathcal{E}}') \subset \tilde{\mathcal{G}}$, where $|\tilde{\mathcal{V}}'| = (s-1)$. We now define *the size of the s-SMECS of a graph* formally.

Definition 10 (Size of the Smallest s-Maximally Externally Connected Subset (s-SMECS) of a Graph)**.** *In a graph $\tilde{\mathcal{G}} = (\tilde{\mathcal{V}}, \tilde{\mathcal{E}})$, for every integer $1 \leq s \leq |\tilde{\mathcal{V}}|$, the size of the smallest s-maximally externally connected subset (s-SMECS) of $\tilde{\mathcal{G}}$, $m_{\tilde{\mathcal{G}}}(s)$, is given by*

$$m_{\tilde{\mathcal{G}}}(s) = \min_{\tilde{\mathcal{G}}' = (\tilde{\mathcal{V}}', \tilde{\mathcal{E}}') \subset \tilde{\mathcal{G}}:\ |\tilde{\mathcal{V}}'| = (s-1)} |\{u \in \tilde{\mathcal{V}}' : \deg^{+}_{\tilde{\mathcal{G}}}(u) - \deg^{+}_{\tilde{\mathcal{G}}'}(u) > |\tilde{\mathcal{V}}| - s\}|. \tag{3.9}$$

We notice that the size of the s-SMECS of a graph does not change if the minimization in its definition is performed over all sub-graphs $\tilde{\mathcal{G}}' = (\tilde{\mathcal{V}}', \tilde{\mathcal{E}}')$, where $|\tilde{\mathcal{V}}'| \leq s-1$. We denote the size of the complement of the s-SMECS set by $\overline{m}_{\tilde{\mathcal{G}}}(s)$ and refer to it as the size of the s-CSMECS of a graph. The s-CSMECS is an important graph-theoretic quantity and dictates the storage costs of our constructions. The size of the s-CSMECS of the graph can be expressed as follows

$$\overline{m}_{\tilde{\mathcal{G}}}(s) = \max_{\tilde{\mathcal{G}}' = (\tilde{\mathcal{V}}', \tilde{\mathcal{E}}') \subset \tilde{\mathcal{G}}:\ |\tilde{\mathcal{V}}'| = (s-1)} |\{u \in \tilde{\mathcal{V}}' : \deg^{+}_{\tilde{\mathcal{G}}}(u) - \deg^{+}_{\tilde{\mathcal{G}}'}(u) \leq |\tilde{\mathcal{V}}| - s\}| \tag{3.10}$$

$$= (s-1) - m_{\tilde{\mathcal{G}}}(s). \tag{3.11}$$

We next give examples to illustrate the calculation of the s-CSMECS.

Example 2. *Consider the graph $\tilde{\mathcal{G}} = (\tilde{\mathcal{V}}, \tilde{\mathcal{E}})$ shown in Fig. 3.3, where $\tilde{\mathcal{V}} = \{0, 1, 2, 3, 4\}$. Consider the case where $s = 5$ and consider a subgraph $\tilde{\mathcal{G}}' = (\tilde{\mathcal{V}}', \tilde{\mathcal{E}}')$, where $\tilde{\mathcal{V}}' = \tilde{\mathcal{V}} \setminus \{4\}$. In this case, we have*

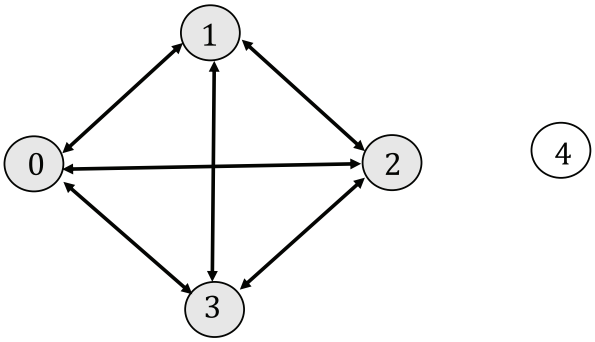


Figure 3.3. A graph $\tilde{\mathcal{G}} = (\tilde{\mathcal{V}}, \tilde{\mathcal{E}})$, where $|\tilde{\mathcal{V}}| = \{0, 1, 2, 3, 4\}$. For the subgraph $\tilde{\mathcal{G}}' = (\tilde{\mathcal{V}}', \tilde{\mathcal{E}}')$, where $\tilde{\mathcal{V}}' = \tilde{\mathcal{V}} \setminus \{4\}$, the vertices $0, 1, 2, 3$ still have an out-degree of 4 and hence $m_{\tilde{\mathcal{G}}}(s) = 0$.

$$m_{\tilde{\mathcal{G}}}(5) = |\{u \in \tilde{\mathcal{V}}' : \deg^{+}_{\tilde{\mathcal{G}}}(u) - \deg^{+}_{\tilde{\mathcal{G}}'}(u) > 0\}| = 0,$$

and hence $\overline{m}_{\tilde{\mathcal{G}}}(5) = 4$.

In general, determining $\overline{m}_{\tilde{\mathcal{G}}}(s)$ is a discrete optimization that is computationally intractable for large graphs. We derive an upper bound on $\overline{m}_{\tilde{\mathcal{G}}}(s)$ for regular directed graphs[6] in Lemma 3. For such a graph, we have

$$\overline{m}_{\tilde{\mathcal{G}}}(s) = \max_{\tilde{\mathcal{G}}' = (\tilde{\mathcal{V}}', \tilde{\mathcal{E}}') \subset \tilde{\mathcal{G}}:\ |\tilde{\mathcal{V}}'| = (s-1)} |\{u \in \tilde{\mathcal{V}}' : \deg^{+}_{\tilde{\mathcal{G}}'}(u) \geq k + s - |\tilde{\mathcal{V}}|\}|. \tag{3.12}$$

Before providing our upper bound, we first consider the following useful lemma.

[6] We recall that in k-regular directed graph $\tilde{\mathcal{G}}$ every vertex has in-degree as well as out-degree equal to k.

Lemma 2. *Consider any subgraph $\tilde{\mathcal{G}}' = (\tilde{\mathcal{V}}', \tilde{\mathcal{E}}')$ of a k-regular graph $\tilde{\mathcal{G}} = (\tilde{\mathcal{V}}, \tilde{\mathcal{E}})$, where $|\tilde{\mathcal{V}}'| = s-1$. For any vertex $u \in \tilde{\mathcal{V}}'$, $\deg^{+}_{\tilde{\mathcal{G}}'}(u) \geq k + s - |\tilde{\mathcal{V}}|$ if and only if there exists at least one vertex $i \in \tilde{\mathcal{V}}$ such that $i \notin \mathcal{N}^{+}_{\tilde{\mathcal{G}}}(u) \cup \tilde{\mathcal{V}}'$.*

Proof. Consider a vertex $u \in \tilde{\mathcal{V}}'$. Suppose there exists at least one vertex $i \notin \mathcal{N}^{+}_{\tilde{\mathcal{G}}}(u) \cup \tilde{\mathcal{V}}'$. In this case, we have

$$\begin{aligned}\deg^{+}_{\tilde{\mathcal{G}}'}(u) &= |\mathcal{N}^{+}_{\tilde{\mathcal{G}}}(u) \cap \tilde{\mathcal{V}}'| \\ &\geq k + (s-1) - (|\tilde{\mathcal{V}}| - 1) \\ &= k + s - |\tilde{\mathcal{V}}|.\end{aligned}$$

Conversely, suppose that $\mathcal{N}^{+}_{\tilde{\mathcal{G}}}(u) \cup \tilde{\mathcal{V}}' = \tilde{\mathcal{V}}$. In this case, we have

$$\begin{aligned}\deg^{+}_{\tilde{\mathcal{G}}'}(u) &= |\mathcal{N}^{+}_{\tilde{\mathcal{G}}}(u) \cap \tilde{\mathcal{V}}'| \\ &= k + (s-1) - |\tilde{\mathcal{V}}|.\end{aligned}$$

□

We now provide an upper bound on $\overline{m}_{\tilde{\mathcal{G}}}(s)$ for k-regular graphs in Lemma 3.

Lemma 3. *For any k-regular graph $\tilde{\mathcal{G}}$, we have*

$$\overline{m}_{\tilde{\mathcal{G}}}(s) \leq \min\Big((|\tilde{\mathcal{V}}| - s + 1)(|\tilde{\mathcal{V}}| - k), s - 1\Big). \tag{3.13}$$

Proof. We can upper-bound $\overline{m}_{\tilde{\mathcal{G}}}(s)$ as follows

$$\begin{aligned}\overline{m}_{\tilde{\mathcal{G}}}(s) &= \max_{\tilde{\mathcal{G}}' \subset \tilde{\mathcal{G}}:\ |\tilde{\mathcal{V}}'| = (s-1)} |\{u \in \tilde{\mathcal{V}}' : \deg^{+}_{\tilde{\mathcal{G}}'}(u) \geq k + s - |\tilde{\mathcal{V}}|\}| \\ &\overset{(a)}{=} \max_{\tilde{\mathcal{G}}' \subset \tilde{\mathcal{G}}:\ |\tilde{\mathcal{V}}'| = (s-1)} \Big| \bigcup_{i \in \tilde{\mathcal{V}} \setminus \tilde{\mathcal{V}}'} \{u \in \tilde{\mathcal{V}}' : i \notin \mathcal{N}^{+}_{\tilde{\mathcal{G}}}(u)\} \Big|\end{aligned}$$

$$
\begin{aligned}
&\overset{(b)}{\leq} \max_{\tilde{\mathcal{G}}' \subset \tilde{\mathcal{G}}:\ |\tilde{\mathcal{V}}'|=(s-1)} \sum_{i \in \mathcal{V} \setminus \tilde{\mathcal{V}}'} |\{u \in \tilde{\mathcal{V}} : i \notin \mathcal{N}^{+}_{\tilde{\mathcal{G}}}(u)\}| \\
&\overset{(c)}{\leq} \max_{\tilde{\mathcal{G}}' \subset \tilde{\mathcal{G}}:\ |\tilde{\mathcal{V}}'|=(s-1)} \sum_{i \in \tilde{\mathcal{V}} \setminus \tilde{\mathcal{V}}'} (|\tilde{\mathcal{V}}| - k) \\
&= (|\tilde{\mathcal{V}}| - s + 1)(|\tilde{\mathcal{V}}| - k),
\end{aligned}
$$

where (a) follows by Lemma 2, (b) follows by the union bound and (c) follows as $\tilde{\mathcal{G}}$ is k-regular.

□

We now consider the two extreme cases of the graph $\tilde{\mathcal{G}} = (\tilde{\mathcal{V}}, \tilde{\mathcal{E}})$. The first case corresponds to a $|\tilde{\mathcal{V}}|$-regular graph $\tilde{\mathcal{G}}$. In this case, we have $\overline{m}_{\tilde{\mathcal{G}}}(s) = 0$. In the second case, there are no edges between the different vertices in $\tilde{\mathcal{G}}$ and the graph has only self-edges. In this case, we have

$$
\begin{aligned}
\overline{m}_{\tilde{\mathcal{G}}}(s) &= (s-1) - \min_{\tilde{\mathcal{G}}' = (\tilde{\mathcal{V}}', \tilde{\mathcal{E}}') \subset \tilde{\mathcal{G}}:\ |\tilde{\mathcal{V}}'|=(s-1)} |\{u \in \tilde{\mathcal{V}}' : \deg^{+}_{\tilde{\mathcal{G}}'}(u) < 1 + s - |\tilde{\mathcal{V}}|\}| \\
&= s - 1.
\end{aligned}
$$

In Example 3 and Example 4, we show cases of regulars graph matching the upper bound of Lemma 3.

Example 3. *Consider a k-regular graph $\tilde{\mathcal{G}} = (\tilde{\mathcal{V}}, \tilde{\mathcal{E}})$, where $k = 5$ and $\tilde{\mathcal{V}} = \{0, 1, 2, \cdots, 11\}$ as shown in Fig. 3.4. Consider that case where $s = |\tilde{\mathcal{V}}| = 12$ and consider a subgraph $\tilde{\mathcal{G}}' = (\tilde{\mathcal{V}}', \tilde{\mathcal{E}}')$, where $\tilde{\mathcal{V}}' = \tilde{\mathcal{V}} \setminus \{0\}$ as shown in Fig. 3.5. In this case, we have*

$$
\begin{aligned}
|\{u \in \tilde{\mathcal{V}}' : \deg^{+}_{\tilde{\mathcal{G}}'}(u) < k + s - |\tilde{\mathcal{V}}|\}| &= |\{u \in \tilde{\mathcal{V}}' : \deg^{+}_{\tilde{\mathcal{G}}'}(u) < 5\}| \\
&= |\{1, 2, 9, 10\}|
\end{aligned}
$$

$$= 4,$$

and hence $\overline{m}_{\tilde{\mathcal{G}}}(12) = 7$*, which matches the upper bound of Lemma 3.*

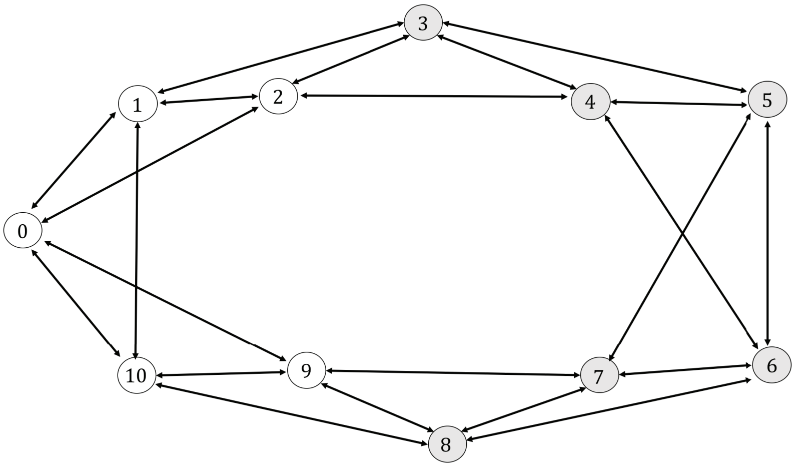


Figure 3.4. A k-regular graph $\tilde{\mathcal{G}} = (\tilde{\mathcal{V}}, \tilde{\mathcal{E}})$ with $k = 5$ and $|\tilde{\mathcal{V}}| = 11$.

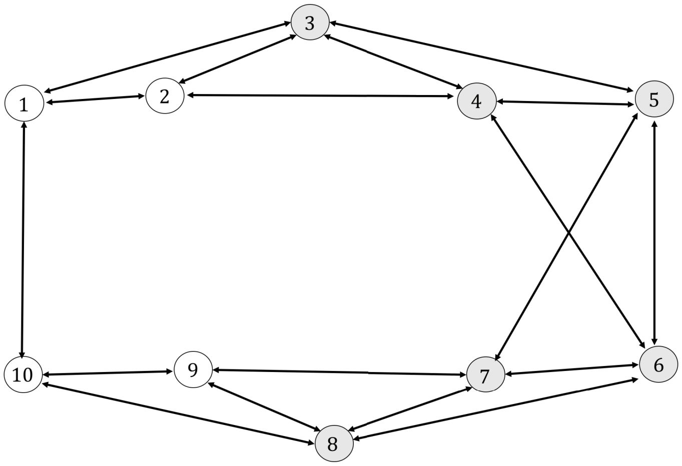


Figure 3.5. A subgraph $\tilde{\mathcal{G}}' = (\tilde{\mathcal{V}}', \tilde{\mathcal{E}}')$, where $\tilde{\mathcal{V}}' = \tilde{\mathcal{V}} \setminus \{0\}$. Vertices $1, 2, 9, 10$ now have an out-degree 4 after the removal of vertex 0.

Example 4. *Consider a* k*-regular graph* $\tilde{\mathcal{G}} = (\tilde{\mathcal{V}}, \tilde{\mathcal{E}})$*, where* $k = 9$ *and* $\tilde{\mathcal{V}} = \{0, 1, \cdots, 10\}$*. The out-neighborhood of* $v \in \tilde{\mathcal{V}}$ *is given by*

$$\mathcal{N}^{+}_{\tilde{\mathcal{G}}}(v) = \{v - (k-1)/2, \cdots, v-2, v-1, v, v+1, v+2, \cdots, v + (k-1)/2\},$$

where the addition is modulo $|\tilde{\mathcal{V}}|$. Consider the case where $s = 9$ and consider a subgraph $\tilde{\mathcal{G}}' = (\tilde{\mathcal{V}}', \tilde{\mathcal{E}}')$, where $\tilde{\mathcal{V}}' = \tilde{\mathcal{V}} \setminus \{0, 3, 7\}$ as shown in Fig. 3.6. In this case, we have

$$\begin{aligned} m_{\tilde{\mathcal{G}}}(9) &= |\{u \in \tilde{\mathcal{V}}' : \deg^{+}_{\tilde{\mathcal{G}}'}(u) < k + s - |\tilde{\mathcal{V}}|\}| \\ &= |\{u \in \tilde{\mathcal{V}}' : \deg^{+}_{\tilde{\mathcal{G}}'}(u) < 7\}| \\ &= |\{4, 10\}| = 2, \end{aligned}$$

which implies that $\overline{m}_{\tilde{\mathcal{G}}}(9) = 6$, which matches the upper bound of Lemma 3.

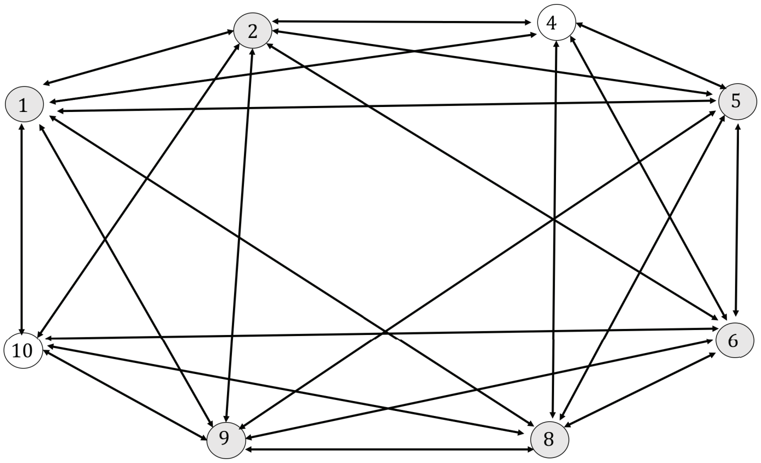


Figure 3.6. A subgraph $\tilde{\mathcal{G}}' = (\tilde{\mathcal{V}}', \tilde{\mathcal{E}}')$, where $\tilde{\mathcal{V}}' = \tilde{\mathcal{V}} \setminus \{0, 3, 7\}$. Vertices $4, 10$ have out-degree of 6.

3.4.3 Code Constructions

In this subsection, we develop our code constructions based on the maximally externally connected subset of the side information graph. In our code constructions, the different versions are encoded separately using MDS codes with different rates, such that any set of encoded symbols that amount to K bits of the version is sufficient to decode this version. Therefore, we describe our code constructions in terms of the amount that each node stores in a given state.

Consider the i-th node and a system state $\mathbf{S} \in \mathcal{P}([\nu])^n$. A version u is encoded using an MDS code and the i-th node stores $\alpha_{i,u}^{\mathbf{S}(\mathcal{H}_i)}$ bits of the codeword of version u in this state. The worst-case storage cost per node is then given by

$$\alpha = \max_{i,\mathbf{S}} \sum_{u=1}^{\nu} \alpha_{i,u}^{\mathbf{S}(\mathcal{H}_i)}. \tag{3.14}$$

In state $\mathbf{S} \in \mathcal{P}([\nu])^n$, we denote the latest version that node i receives that is at least received by $c_W + |\mathcal{H}_i| - n$ nodes in the neighborhood of node i by

$$L_{\mathbf{S}(i)} = \max \{u \in \mathbf{S}(i) : |\mathcal{A}_{\mathbf{S}}(u) \cap \mathcal{H}_i| \geq c_W + |\mathcal{H}_i| - n\}. \tag{3.15}$$

We now provide our first construction. In this construction, when the i-th node observes at least $c_W + |\mathcal{H}_i| - n$ nodes having a $\mathbf{W}_2$, it stores K/c of $\mathbf{W}_2$. Since observing $c_W + |\mathcal{H}_i| - n$ nodes having $\mathbf{W}_2$ does not imply that $\mathbf{W}_2$ is a complete version, the i-th node allocates the remaining storage budget of $(\alpha - K/c)$ to $\mathbf{W}_1$ as it may be the latest complete version. We now provide the construction formally.

Construction 3. *We construct a code as follows for $\nu = 2$*

$$\alpha = \frac{c + \overline{m}_{\mathcal{G}}(c_W)}{c^2} K \tag{3.16}$$

and

$$\alpha_{i,2}^{\mathbf{S}(\mathcal{H}_i)} = \begin{cases} K/c & L_{\mathbf{S}(i)} = 2, \\ 0 & \text{otherwise,} \end{cases} \tag{3.17}$$

$$\alpha_{i,1}^{\mathbf{S}(\mathcal{H}_i)} = \alpha - \alpha_{i,2}^{\mathbf{S}(\mathcal{H}_i)} = \begin{cases} \frac{\overline{m}_{\mathcal{G}}(c_W)}{c^2}K, & L_{\mathbf{S}(i)} = 2, \\ \alpha & \textit{otherwise,} \end{cases} \tag{3.18}$$

where $c_W + |\mathcal{H}_i| \geq n, \forall i \in \mathcal{N}$.

Version 1 *is encoded using an* $(n', c^2/\overline{m}_{\mathcal{G}}(c_W))$ *MDS code* $\mathcal{C}' : [2^K] \to [2^{\overline{m}_{\mathcal{G}}(c_W)K/c^2}]^{n'}$ *over alphabet* $[2^{\overline{m}_{\mathcal{G}}(c_W)K/c^2}]$*, where* $n' = (c + \overline{m}_{\mathcal{G}}(c_W))n/\overline{m}_{\mathcal{G}}(c_W)$ *and* $\mathcal{C}'^{(i)} : [2^K] \to [2^{\overline{m}_{\mathcal{G}}(c_W)K/c^2}]$ *denotes i-th co-ordinate of the output of* $\mathcal{C}'$*. The i-th node stores co-ordinates* $\{(i-1)\frac{c+\overline{m}_{\mathcal{G}}(c_W)}{\overline{m}_{\mathcal{G}}(c_W)} + 1, \cdots, i\frac{c+\overline{m}_{\mathcal{G}}(c_W)}{\overline{m}_{\mathcal{G}}(c_W)}\}$ *of* $\mathcal{C}'$ *if* $L_{\mathbf{S}(i)} = 1$ *and stores co-ordinate* $(i-1)\frac{c+\overline{m}_{\mathcal{G}}(c_W)}{\overline{m}_{\mathcal{G}}(c_W)} + 1$ *otherwise. Version* 2 *is encoded using an* (n, c) *MDS code* $\mathcal{C} : [2^K] \to [2^{K/c}]^n$ *over alphabet* $[2^{K/c}]$ *and the i-th node stores the i-th co-ordinate of the output of* $\mathcal{C}$*,* $\mathcal{C}^{(i)} : [2^K] \to [2^{K/c}]$*, if* $L_{\mathbf{S}(i)} = 2$.

We assume that K/c, $\frac{\overline{m}_{\mathcal{G}}(c_W)}{c^2}K$ and $\frac{c}{\overline{m}_{\mathcal{G}}(c_W)}$ are integers. It is instructive to note that the coding strategy described in the construction with the code $\mathcal{C}'$ is consistent with the storage allocation of (3.16)-(3.18). In particular, each co-ordinate of $\mathcal{C}'$ has $\frac{\overline{m}_{\mathcal{G}}(c_W)K}{c^2}$ bits. Therefore, if $L_{\mathbf{S}(i)} = 2$, then our strategy stores a single co-ordinate of version 1 and the number of bits stored of version 1 is consistent with (3.18). Similarly, if $L_{\mathbf{S}(i)} = 1$, the construction stores $\frac{c+\overline{m}_{\mathcal{G}}(c_W)}{\overline{m}_{\mathcal{G}}(c_W)}$ co-ordinates of $\mathcal{C}'$, which corresponds α bits of version 1. Similarly, the storage cost corresponds to (3.17) for version 2. It is also instructive to note that the dimensions of $\mathcal{C}', \mathcal{C}$ are respectively $\frac{c^2}{\overline{m}_{\mathcal{G}}(c_W)}$ and c. Therefore, by the MDS property, any $\frac{c^2}{\overline{m}_{\mathcal{G}}(c_W)}$ symbols of $\mathcal{C}'$ and any c symbols of $\mathcal{C}$ respectively suffice for decoding the respective encoded symbols; note importantly, that these correspond to a number of symbols equivalent to K bits, which is used in the proof Theorem 4.

Theorem 4. *Construction 3 is a* $(\mathcal{G} = (\mathcal{N}, \mathcal{E}), c_W, c_R, \nu = 2, 2^K, q)$ *multi-version* *code with side information with a worst-case storage cost of*

$$\frac{c + \overline{m}_{\mathcal{G}}(c_W)}{c^2} K, \tag{3.19}$$

where $c_W + |\mathcal{H}_i| \geq n, \forall i \in \mathcal{N}$.

Proof. Consider any state $\mathbf{S} \in \mathcal{P}([\nu])^n$. We show that the latest complete version in this state, version $L_{\mathbf{S}}$, is decodable. Specifically, we consider the following cases.

- **Case** 1 ($L_{\mathbf{S}} = 1$). Since $L_{\mathbf{S}} = 1$, then version 2 is incomplete. In this case, at most $\overline{m}_{\mathcal{G}}(c_W)$ nodes will allocate a storage budget to version 2. According to the construction, each of these nodes will store K/c of version 2 and $(\alpha - K/c)$ of version 1. Therefore, the storage allocation of version 1 is at least

  $$\overline{m}_{\mathcal{G}}(c_W)(\alpha - K/c) + (c - \overline{m}_{\mathcal{G}}(c_W))\alpha = K.$$

  Therefore, by the MDS property, version 1 is decodable in this state.

- **Case** 2 ($L_{\mathbf{S}} = 2$). Since version 2 is complete, then at least c_W nodes have it. The decoder connects to any c_R nodes. Among these c_R nodes, at least c nodes $T = \{t_1, t_2, \cdots, t_c\} \subseteq \mathcal{N}$ have received this version. Server $t_i \in T, i \in [c]$, observes at least $c_W + |\mathcal{H}_{t_i}| - n$ nodes that have received version 2 and hence it stores K/c of version 2. Since each of the c nodes stores K/c of version 2, then version 2 is decodable in this state.

□

Remark 5. *For network topologies where $\overline{m}_{\mathcal{G}}(c_W) < \frac{c(c-1)}{c+1}$, the storage cost of Theorem 4 is strictly less than $\frac{2}{c+1}K$ and hence the side information is useful in those cases.*

We next provide our second construction for any number of versions ν.

Construction 4. *We construct a code as follows*

$$\alpha = \frac{1}{c-(\nu-1)\overline{m}_{\mathcal{G}}(c_W)}K, \tag{3.20}$$

$$\alpha_{i,u}^{\mathbf{S}(\mathcal{H}_i)} = \begin{cases} \alpha & \text{if } L_{\mathbf{S}(i)} = u, \\ 0 & \text{otherwise,} \end{cases} \tag{3.21}$$

where $i \in \mathcal{N}$ and $c > (\nu-1)\overline{m}_{\mathcal{G}}(c_W)$. Specifically, version u is encoded using an $(n, c-(\nu-1)\overline{m}_{\mathcal{G}}(c_W))$ MDS code $\mathcal{C} : [2^K] \rightarrow [2^{K/(c-(\nu-1)\overline{m}_{\mathcal{G}}(c_W))}]^n$ over alphabet $[2^{K/(c-(\nu-1)\overline{m}_{\mathcal{G}}(c_W))}]$ and the i-th node stores the i-th co-ordinate of the output of $\mathcal{C}$ denoted by $\mathcal{C}^{(i)} : [2^K] \rightarrow [2^{K/(c-(\nu-1)\overline{m}_{\mathcal{G}}(c_W))}]$ if $L_{\mathbf{S}(i)} = u$.

It can be readily verified that the code $\mathcal{C}$ corresponds to the storage allocation dictated by (3.21).

Theorem 5. *Construction 4 is a $(\mathcal{G} = (\mathcal{N}, \mathcal{E}), c_W, c_R, \nu, 2^K, q)$ multi-version code with side information with a worst-case storage cost of*

$$\frac{1}{c-(\nu-1)\overline{m}_{\mathcal{G}}(c_W)}K, \tag{3.22}$$

where $c > (\nu-1)\overline{m}_{\mathcal{G}}(c_W)$.

Proof. Consider any state $\mathbf{S} \in \mathcal{P}([\nu])^n$. We show that the latest complete version in this state, version $L_{\mathbf{S}}$, is decodable. Specifically, we have the following cases.

- Consider any other version u such that $u < L_{\mathbf{S}}$. Suppose that the decoder connects to any set of c_R nodes. Among these c_R nodes there are at least

c nodes, denoted by $T = \{t_1, t_2, \cdots, t_c\}$, that have version $L_{\mathbf{S}}$. For any node $t_i \in T, i \in [c], L_{\mathbf{S}} \leq L_{\mathbf{S}(t_i)}$. Therefore, for any node $t_i \in T, i \in [c]$, we have $u < L_{\mathbf{S}(t_i)}$ and hence none of these c nodes store u according to the construction.

- Consider any other version $u > L_{\mathbf{S}}$. Since version u is incomplete, we have

$$|\{i \in \mathcal{A}_{\mathbf{S}}(u) : |\mathcal{A}_{\mathbf{S}}(u) \cap \mathcal{H}_i| \geq c_W + |\mathcal{H}_i| - n\}| \leq \overline{m}_{\mathcal{G}}(c_W).$$

A node i of those nodes stores α of version u if $u = L_{\mathbf{S}(i)}$. Since there are at most $(\nu - 1)$ versions that are not equal to $L_{\mathbf{S}}$, at least $c - (\nu - 1)\overline{m}_{\mathcal{G}}(c_W)$ nodes will store $\frac{1}{c-(\nu-1)\overline{m}_{\mathcal{G}}(c_W)}K$ of version $L_{\mathbf{S}}$. Therefore, the storage allocation of version $L_{\mathbf{S}}$ is at least K and version $L_{\mathbf{S}}$ is decodable in this state.

□

Remark 6. *Since our code constructions are threshold-based, a node does not need to be aware of the whole topology and just needs to be aware of the number of nodes in its out-neighborhood.*

Remark 7. *Construction* 3 *has a strictly better storage cost as compared with Construction 4 for the case where* $\nu = 2$ *for* $\overline{m}_{\mathcal{G}}(c_W) > 0$.

Remark 8. *For network topologies where* $\overline{m}_{\mathcal{G}}(c_W) < \frac{c-1}{\nu}$*, the storage cost of Theorem 5 is strictly less than* $\frac{\nu}{c+\nu-1}K$ *and hence exchanging the side information is useful in those cases.*

We illustrate Construction 3 and compare between the different side information regimes in Example 5.

Example 5. *Consider a key-value store with $n = 5$ and $c_W = c_R = 4$ as shown in Fig. 3.1-(c). Recall that for the case where there is no exchange of side information, the storage cost is equal to $K/2$ bits and for complete side information case, the storage cost is equal to $K/3$ bits. In this partial side information case, we have*

$$\begin{aligned}\overline{m}_{\mathcal{G}}(c_W) &= \max_{\mathbf{S}\in\mathcal{P}([\nu])^n:\mathcal{A}_{\mathbf{S}}(u)=(c_W-1)} |\{i \in \mathcal{A}_{\mathbf{S}}(u) : |\mathcal{A}_{\mathbf{S}}(u) \cap \mathcal{H}_i| \geq c_W + |\mathcal{H}_i| - n\}| \\ &= 1.\end{aligned}$$

Hence, using Construction 3, the storage cost is given by

$$\begin{aligned}\alpha &= \frac{c + \overline{m}_{\mathcal{G}}(c_W)}{c^2} \\ &= 4/9\ K.\end{aligned}$$

Thus, our construction lowers the storage cost as compared to the no side information case. In order to illustrate the storage allocation of Construction 3, we consider the specific scenario depicted in Fig. 3.1-(c). In this case, the storage allocations of the different nodes are given in Table 3.1. If the decoder connects to $\mathcal{R} = \{1, 2, 3, 4\}$, then it can decode $\mathbf{W}_1$ as the total number of bits stored in those servers of $\mathbf{W}_1$ is equal to K bits. We summarize the comparison between the different side information regimes in Table 3.2.

Node	0	1	2	3	4
$\mathbf{W}_1$	$4/9\ K$	$4/9\ K$	$1/9\ K$	$4/9\ K$	0
$\mathbf{W}_2$	0	0	$1/3\ K$	0	0

Table 3.1. Storage allocations of Construction 3.

Side Information Regime	Storage cost
No side information	1/2 K
Partial side information	4/9 K
Complete side information	1/3 K

Table 3.2. Storage costs for the various side information regimes.

3.5 Lower Bounds on the storage costs

In this section, we provide lower bounds on the storage cost. In Theorem 6, we provide a lower bound for any general topology that satisfies certain condition. In Theorem 7, we provide a lower bound for a symmetric multi-hop topology where the servers are distributed in a ring such that each server is aware of the states of its h-hop neighbors.

3.5.1 Lower bound for general topology

We begin by studying a general side information topology such that there are $c-a$ servers that are not aware of the states of other a servers, where $c = c_W + c_R - n$ and $a \in \{0, 1, \cdots, c-1\}$. We state our result next in Theorem 6.

Theorem 6. *A $(\mathcal{G} = (\mathcal{N}, \mathcal{E}), c_W, c_R, \nu = 2, 2^K, q)$ multi-version code with side information where there exist $(c-a)$ servers, denoted by $l_0, l_1, \cdots, l_{c-a-1} \in \mathcal{N}$, such that $\left|\mathcal{N} \setminus \bigcup_{i \in \{l_0, l_1, \cdots, l_{c-a-1}\}} \mathcal{H}_i\right| \geq a$ must satisfy*

$$\log q \geq \min\left\{\frac{1}{c-a}, \frac{2}{c+a}\right\} K, \tag{3.23}$$

where $c = c_W + c_R - n$ and $a \in \{0, 1, \cdots, c-1\}$.

We provide the proof of Theorem 6 in Appendix B and explain the main idea of the proof here.

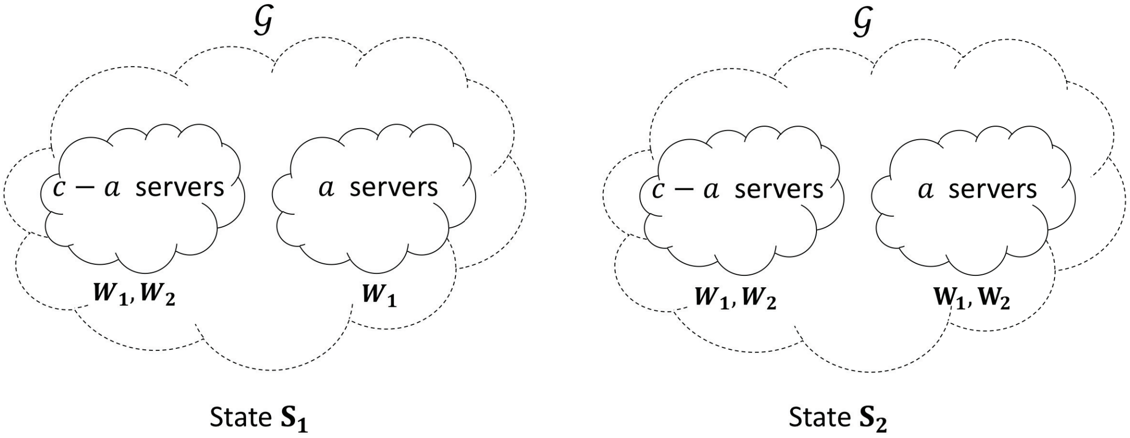


Figure 3.7. Two system states $\mathbf{S}_1$ and $\mathbf{S}_2$ are shown. In $\mathbf{S}_1$, the decoder can return either $\mathbf{W}_1$ or $\mathbf{W}_2$. In $\mathbf{S}_2$, the decoder must return $\mathbf{W}_2$.

The main idea of the proof follows by constructing two states $\mathbf{S}_1, \mathbf{S}_2 \in \mathcal{P}([\nu])^n$, such that $L_{\mathbf{S}_1} = 1$ and $L_{\mathbf{S}_2} = 2$ as shown in Fig. 3.7. Since there exist $(c-a)$ servers, denoted by $l_0, l_1, \cdots, l_{c-a-1} \in \mathcal{N}$, such that $\left|\mathcal{N} \setminus \bigcup_{i \in \{l_0, l_1, \cdots, l_{c-a-1}\}} \mathcal{H}_i\right| \geq a$, then there exist a servers denoted by $\{i_0, i_1, \cdots, i_{a-1}\} \subseteq \mathcal{N} \setminus \bigcup_{i \in \{l_0, l_1, \cdots, l_{c-a-1}\}} \mathcal{H}_i$.

We construct the two states such that servers $l_0, l_1, \cdots, l_{c-a-1}$ have both versions in the two states and only servers $i_0, i_1, \cdots, i_{a-1}$ change their states from having only $\mathbf{W}_1$ in $\mathbf{S}_1$ to having both versions in $\mathbf{S}_2$. In both states, the decoder connects to the same set of servers denoted by $\mathcal{R}$, where

$$\mathcal{A}_{\mathbf{S}_1}(1) \cap \mathcal{R} = \mathcal{A}_{\mathbf{S}_2}(2) \cap \mathcal{R} = \{l_0, l_1, \cdots, l_{c-a-1}\} \cup \{i_0, i_1, \cdots, i_{a-1}\}. \tag{3.24}$$

Importantly, servers $l_0, l_1, \cdots, l_{c-a-1}$ cannot differentiate between the two states as they do not know the states of the servers $i_0, i_1, \cdots, i_{a-1}$. In $\mathbf{S}_2$, $\mathbf{W}_2$ must be decoded as it is the latest complete version. In $\mathbf{S}_1$, the decoder can return either

$\mathbf{W}_1$ or $\mathbf{W}_2$ as $\mathbf{W}_1$ is the latest complete version. Decoding $\mathbf{W}_2$ in $\mathbf{S}_1$ implies that

$$(c-a)\log q \geq K. \tag{3.25}$$

Decoding $\mathbf{W}_1$ in $\mathbf{S}_1$ implies that both versions must be recoverable from the c symbols of servers $\{l_0, \cdots, l_{c-a-1}\} \cup \{i_0, \cdots, i_{a-1}\}$ in $\mathbf{S}_1$ and the a symbols of servers $\{i_0, \cdots, i_{a-1}\}$ in $\mathbf{S}_2$, thus

$$(c+a)\log q \geq 2K. \tag{3.26}$$

Since the decoder in this state can either decode $\mathbf{W}_1$ or $\mathbf{W}_2$, we get the lower bound given by Theorem 6.

Remark 9. *For a side information graph $\mathcal{G} = (\mathcal{N}, \mathcal{E})$ where the set A containing the possible values of a has multiple values, we have*

$$\log q \geq \max_{a \in A} \ \min \left\{ \frac{1}{c-a}, \frac{2}{c+a} \right\} K. \tag{3.27}$$

Corollary 1. *A $(\mathcal{G} = (\mathcal{N}, \mathcal{E}), c_W, c_R, \nu = 2, 2^K, q)$ multi-version code with side information where there exist two servers $l_0, l_1 \in \mathcal{N}$ such that $\left|\mathcal{N} \setminus \bigcup_{i \in \{l_0, l_1\}} \mathcal{H}_i\right| \geq 1$ must satisfy*

$$\log q \geq K/2,$$

where $c = c_W + c_R - n = 3$.

The proof of Corollary 1 follows directly from Theorem 6. The implication of Corollary 1 is that, for $c = 3$, if two servers are not aware of the state of one server, then the side information does not help in reducing the worst-case storage cost.

That follows since a worst-case storage cost of $K/2$ can be achieved in a distributed manner with no side information using the code construction proposed in [7].

3.5.2 Lower bound for a multi-hop topology

We next consider a multi-hop network topology, where the servers are distributed in a ring such that every server is aware of the states of its h-hop neighbors as shown in Fig. 3.8. In this topology, we denote the set of the servers by $\mathcal{N} = \{0, 1, \cdots, n-1\}$ and the neighborhood of server i is given by $\mathcal{H}_i = \{i-h, \cdots, i-2, i-1, i, i+1, i+2, \cdots, i+h\}$, where the addition is modulo n. For this topology, Theorem 7 provides an explicit lower bound on the storage cost.

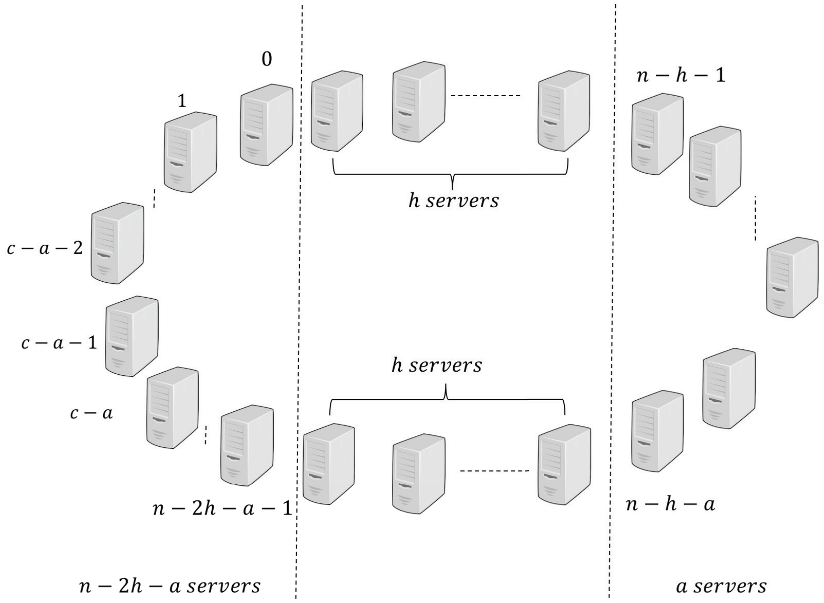


Figure 3.8. The multi-hop topology under consideration where the edges between a node and the h-hop neighbors are not shown.

Theorem 7. *A* $(\mathcal{G} = (\mathcal{N}, \mathcal{E}), c_W, c_R, \nu = 2, 2^K, q)$ *multi-version code with side* *information on the multi-hop topology must satisfy*

$$\log q \geq \frac{2}{2c - \min(n-2h, c) + \lceil \min(n-2h, c)/3 \rceil} K, \tag{3.28}$$

where $c = c_W + c_R - n$.

The proof of Theorem 7 follows also by constructing two states such that some servers cannot differentiate between the two states and is given in Appendix C. For the case where $(n-2h) \geq c$, we have

$$|\mathcal{N} \setminus \bigcup_{i \in \{0,1,\cdots,c-a-1\}} \mathcal{H}_i| = |\{n-h-a, n-h-a+1, \cdots, n-h-1\}| = a. \tag{3.29}$$

Therefore, we can can apply the result obtained in Theorem 6 directly in this case. For the case where $(n-2h) < c$, however, the result of Theorem 6 does not apply. We only provide the basic idea here of the proof.

The main idea of the proof follows by constructing two states $\mathbf{S}_1 \in \mathcal{P}([\nu])^n$, where $\mathbf{W}_1$ is the latest complete version and $\mathbf{S}_2 \in \mathcal{P}([\nu])^n$ where $\mathbf{W}_2$ is the latest complete version. Importantly, servers $0, 1, \cdots, t-a-1$, where $t = \min(n-2h, c)$, cannot differentiate between the two states. In state $\mathbf{S}_2$, the set of servers that have $\mathbf{W}_1$ is the same as the set of servers that have $\mathbf{W}_2$ and is given by the following disjoint union

$$\begin{aligned} \mathcal{A}_{\mathbf{S}_2}(1) = \mathcal{A}_{\mathbf{S}_2}(2) = \{0, 1, \cdots, t-a-1\} \cup \{l_{t-a}, l_{t-a+1}, \cdots, l_{c_W-a-1}\} \cup \\ \{n-h-a, \cdots, n-h-1\}, \end{aligned} \tag{3.30}$$

where $\{l_{t-a}, l_{t-a+1}, \cdots, l_{c_W-a-1}\} \subset \mathcal{N}$. In state $\mathbf{S}_1$, the set of servers that have $\mathbf{W}_1$

is given by

$$\begin{aligned}\mathcal{A}_{\mathbf{S}_1}(1) = \{0, 1, \cdots, t-a-1\} \cup \{l_{t-a}, l_{t-a+1}, \cdots, l_{c_W-a-1}\} \cup \\ \{n-h-a, \cdots, n-h-1\},\end{aligned}$$

and the set of servers that have $\mathbf{W}_2$ is given by

$$\mathcal{A}_{\mathbf{S}_1}(2) = \{0, 1, \cdots, t-a-1\} \cup \{l_{t-a}, l_{t-a+1}, \cdots, l_{c_W-a-1}\}. \tag{3.31}$$

We assume that decoder connects to the following set of servers to read the data

$$\mathcal{R} = \mathcal{N} \setminus \{l_{t-a}, l_{t-a+1}, \cdots, l_{t+c_W-c-a-1}\}. \tag{3.32}$$

Since only $(c-a)$ servers in $\mathcal{R}$ have $\mathbf{W}_2$ in $\mathbf{S}_1$, decoding $\mathbf{W}_2$ in this state implies that the storage cost is lower-bounded as follows

$$(c-a) \log q \geq K. \tag{3.33}$$

Decoding $\mathbf{W}_1$ in $\mathbf{S}_1$ implies that $\mathbf{W}_1$ and $\mathbf{W}_2$ must be recoverable from the symbols of servers $\{0, 1, \cdots, t-a-1\} \cup \{l_{t+c_W-c-a}, \cdots, l_{c_W-a-1}\} \cup \{n-h-a, \cdots, n-h-1\}$ in $\mathbf{S}_1$ and the symbols of servers $\{l_{t+c_W-c-a}, \cdots, l_{c_W-a-1}\} \cup \{n-h-a, \cdots, n-h-1\}$ in $\mathbf{S}_2$. Hence, the storage cost in this case is lower-bounded as follows

$$(2c-t+a) \log q \geq 2K, \tag{3.34}$$

and Theorem 7 follows by choosing a that maximizes the lower bound.

Corollary 2. *A* $(\mathcal{G} = (\mathcal{N}, \mathcal{E}), c_W, c_R, \nu = 2, 2^K, q)$ *multi-version code with side* *information on the multi-hop topology must satisfy*

$$\log q \geq K/2, \tag{3.35}$$

where $h \leq \lfloor (n-3)/2 \rfloor$ and $c = c_W + c_R - n = 3$.

The proof of Corollary 2 follows directly from Theorem 7. Corollary 2 implies that for $c = 3$ even if the server is aware of the states of $(n-3)$ other servers, the side information does not reduce the storage cost beyond the case where there is no exchange of side information.

3.6 Examples

In this section, we provide examples showing the storage gain of our codes. We begin in Section 3.6.1 by showing the storage gain for different regimes of the side information. In Section 3.6.2, we show the potential utility of applying our code constructions to AWS.

3.6.1 Numerical Example

In this subsection, we show the storage gain of our schemes as compared with the case where there is no side information through an example.

Example 6 (Multi-hop network Topology)**.** *Consider a multi-hop network topology where the servers are distributed in a ring such that every server is aware of the states of its h-hop neighbors. In this topology, we denote the set of the servers by $\mathcal{N} = \{0, 1, \cdots, n-1\}$ and the neighborhood of server i is given by $\mathcal{H}_i = \{i-h, \cdots, i-2, i-1, i, i+1, i+2, \cdots, i+h\}$, where the addition is modulo n.*

The achievable storage cost in this case is given by

$$\alpha = \min\left(\frac{2}{c+1}, \frac{c + \overline{m}_{\mathcal{G}}(c_W)}{c^2}\right) K.$$

In Fig. 3.9, we show the ratio between the storage cost of Construction 3 and the storage cost of the completely decentralized case $\alpha_0 = \frac{2}{c+1}K$ *for the case where* $n = 21, \nu = 2, c_W = c_R = 19$, *hence* $c = 17$. *Exchanging the side information and using our code construction does not reduce the storage cost for the cases where* $h \leq 7$. *As* h *exceeds* 7, *using our code construction reduces the storage cost gradually until we reach the case where* $h = 10$ *which corresponds to the complete side information case, hence the storage cost is equal to* K/c.

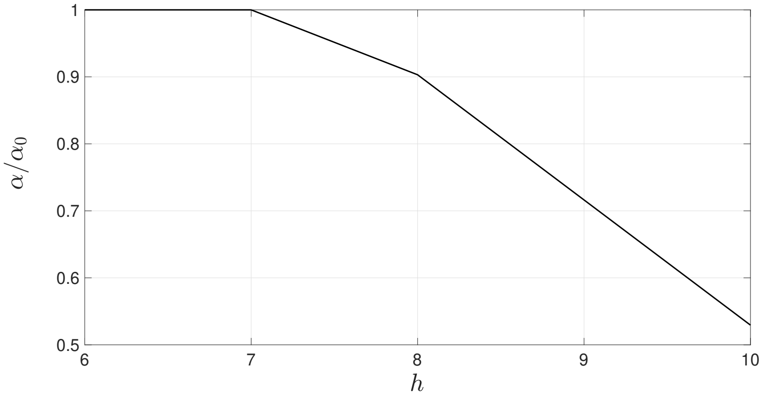


Figure 3.9. The ratio between the storage cost of Construction 3 and storage cost lower bound in the case where there is no side information $\alpha_0 = \frac{2K}{c+1}$ for $n = 21$.

3.6.2 Case Study: Amazon Web Services

In this subsection, we show the potential utility of applying our schemes to the data centers of Amazon. Table 3.3 provides the data centers locations and the storage prices obtained from [67] as of 03/11/2019. In Table 3.4, we provide the latency between 13 data center of Amazon obtained from [68] as of 03/11/2019. We assume that a shared object is stored over the data centers $\mathcal{N} = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13\}$, $\nu = 2$ and $c_W = c_R = 11$, thus $c = 9$.

Data center	Storage Price	Data center	Storage Price	Data center	Storage Price
1. Tokyo	0.019	6. Frankfurt	0.0135	11. Ohio	0.0125
2. Seoul	0.018	7. Ireland	0.0125	12. N. California	0.019
3. Mumbai	0.019	8. London	0.0131	13. Oregon	0.01
4. Singapore	0.016	9. Paris	0.0131		
5. Canada	0.0138	10. N. Virginia	0.0125		

Table 3.3. Storage prices in $/GB.

.

Data center	1	2	3	4	5	6	7	8	9	10	11	12	13
1	0	37.8	157.2	90.8	177.2	249.7	234.4	259.4	259.4	167.5	166.2	119.6	106.5
2	37.9	0	160.1	105.7	199.7	269.9	255.7	269.3	268.2	190.7	189.3	153	128.2
3	136.9	181.5	0	68.8	212.8	129.9	134.4	128	118.3	187.7	202.2	240.8	225
4	90	112.4	82.3	0	240.9	189.7	186.4	181.3	178.5	267.8	232.6	184.7	194.7
5	159.2	189.5	202	222.3	0	103.1	81.7	92	95.4	17.8	27.2	82	81.7
6	241.3	267.3	115.3	174.8	107	0	24.2	19.1	12.8	90.4	98.9	147.8	165.4
7	230	258.4	128.4	180	85.2	23.8	0	14.6	21.6	72.7	84.6	152.8	137.4
8	236.9	265.3	116.9	168	93.9	15.7	13.2	0	10.7	78	88.7	141.7	148.5
9	233.5	301.6	111.6	173	97.6	14.4	20.4	11	0	81.7	99.4	140.7	157.8
10	164.3	188.8	195.8	239.9	18.8	92	73.1	79.8	110.5	0	13.66	67.2	79.3
11	162.4	189.9	199.7	226	27.6	121.5	87.7	91.3	94.6	16.4	0	55.9	74.53
12	111.4	157.9	253.4	178.3	81.7	148.7	150.7	140	146.7	67.8	53.9	0	23.4
13	109.8	139.7	226	166.5	73.4	167.8	137.8	150.8	160.4	84	73	25.8	0

Table 3.4. Latency between data centers in ms.

Although we have not developed a protocol, we assume the existence of a hypothetical protocol with the following property: exchanging side information between two data centers in Fig. 3.4 involves an additional latency corresponding to the delay between these data centers. We discuss the trade-off between the projected additional latency of the write operation and the storage cost reductions we obtain using the side information. By bounding the maximum allowable additional latency, we obtain the side information graph $\mathcal{G} = (\mathcal{N}, \mathcal{E})$. Specifically, an edge exists from a data center $i \in \mathcal{N}$ to a data center $j \in \mathcal{N}$, if the latency from the j-th data center to the i-th data center is below this specified latency bound. We use the latencies based on Table 3.4 to explore the role of our code constructions, via a trade-off between the storage costs and latencies as shown in Fig. 3.10. Intuitively, allowing higher latency, may add more edges to $\mathcal{G}$ and that can decrease $\overline{m}_{\mathcal{G}}(c_W)$ which improves the storage cost. Note that our storage costs and latencies are *projected* values based on Table 3.4. The reported latencies should be interpreted as the additional latency incurred by write operations for writing a new value that would be incurred due to the exchange of side information. The actual latency can be higher as protocols (e.g., [12, 30, 31]) have additional rounds of communication, for instance, to discern the latest logical time-stamp, etc.

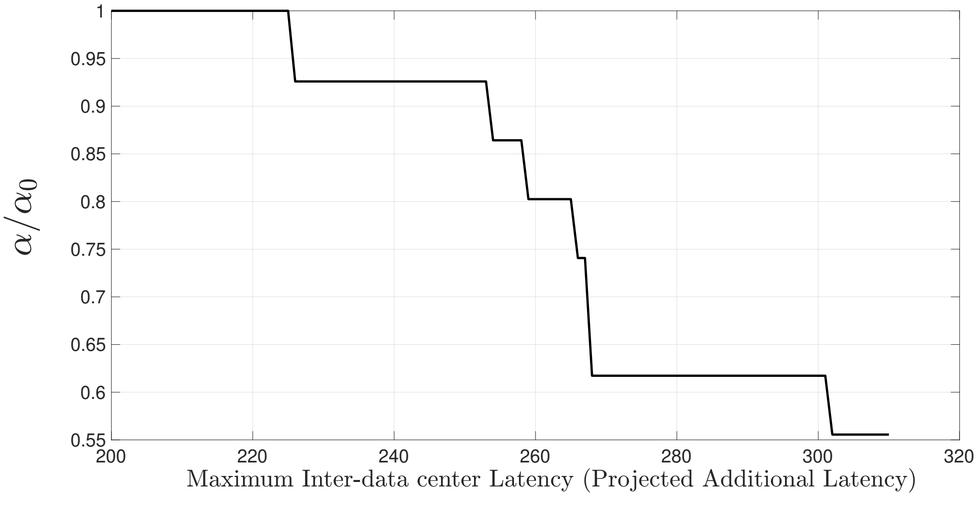


Figure 3.10. Trade-off between the projected normalized storage cost (with respect to the storage cost of the completely decentralized setting) and the projected additional latency.

We now compare between the different side information regimes in terms of the additional latency due to side information exchange and the corresponding storage cost reductions.

A. No side information

The storage cost in this case is given by

$$\alpha = \frac{2}{c+1} K = 0.2K.$$

The storage prices in dollars of the system are given in Table 3.3. We denote the storage price of the i-th data center by p_i, where $i \in \mathcal{N}$. The total storage price denoted by P is expressed as

$$P = \alpha \sum_{i \in \mathcal{N}} p_i,$$

and in our case

$$P = \$0.0384\ K.$$

B. Partial side information

- **Additional Latency.** In the case where the data centers do not exchange their states (completely decentralized system), the nodes do not wait before deciding what to store. In the partial side information case, the nodes wait for the side information of the other nodes to be received before deciding what to store. Therefore, there is an additional latency incurred in the partial side information case as compared with the case of no side information. For the sake of example, we assume that the maximum allowable additional latency is 260 ms.

- The storage cost in this case is given by

$$\alpha = \frac{c + \overline{m}_{\mathcal{G}}(c_W)}{c^2} K.$$

While a side information graph that corresponds to the complete side information case has 169 edges - every edge between the 13 data centers including self-edges - the side information graph $\mathcal{G}$ that corresponds to maximum allowable additional latency of 260 ms has only 162 edges. For instance, there is no edge between the Seoul data center (#2) and Frankfurt data center (#6). In the partial side information case, using a brute-force computer search, we can verify that

$$\overline{m}_{\mathcal{G}}(c_W) = \max_{\mathbf{S} \in \mathcal{P}([\nu])^n : \mathcal{A}_{\mathbf{S}}(u) = (c_W - 1)} |\{i \in \mathcal{A}_{\mathbf{S}}(u) : \\ |\mathcal{A}_{\mathbf{S}}(u) \cap \mathcal{H}_i| \geq c_W + |\mathcal{H}_i| - n\}| = 4.$$

Therefore, the storage cost is given by

$$\alpha = 0.16K$$

and the total storage price is equal to

$$P = \alpha \sum_{i \in \mathcal{N}} p_i = \$0.0307 \ K.$$

Storage Gain. The lower bound on the storage cost in case of no side information implies a storage cost that is at least $0.2K - o(K)$ bits. Therefore, in the partial side information case, we get a storage reduction of 19.75% of our achievable scheme as compared with the optimal achievable scheme with no side information.

C. Complete side information

- The maximum allowable additional latency has to be at least 301.6 ms to allow all data centers to exchange their states.
- The storage cost in bits in this case is given by

$$\alpha = \frac{1}{c}K = 0.1111K.$$

The total storage price in this case is given by

$$P = \alpha \sum_{i \in \mathcal{N}} p_i = \$0.0213K.$$

We summarize the additional latency-storage trade-off in Table 3.5.

Case	Maximum inter-data center latency	Storage price
No side information	0	\$0.0384 K
Partial side information	260 ms	\$0.0307 K
Complete side information	301.6 ms	\$0.0213 K

Table 3.5. The maximum inter-data center latencies (i.e., additional latency due to side information exchange) and the corresponding storage costs for various regimes.

3.7 Conclusion

In this chapter, we have studied geo-distributed key-value stores where a node can acquire side information of which data versions propagated to some other nodes based on the underlying topology. Our code constructions show that exchanging side information results in a better storage cost for some regimes at the expense of the higher latency and a negligible communication cost of exchanging the side information. The potential storage cost reduction of our constructions has been demonstrated through a case study over Amazon web services. Interestingly, our converse results identify topologies where a tremendous amount of side information does not improve the storage cost. Our work differs from the conventional approach in distributed algorithm design, where algorithm design is performed first, and performance and coding related aspects are an afterthought. Our approach mirrors evolution of communication systems design where optimistic assumptions (e.g., multiple antennas) on the system are first hypothesized, and their potential gains are studied. Based on these insights, future work in system design explores the realization of these hypotheses in practice. Similarly, in this work, we hypothesize the existence of a protocol with low meta-data overheads and study the extent to which storage cost can be obtained using an information-theoretic system model.

Thereby, our work opens up several open directions. First, the multi-version coding problem with side information adds to the rich pantheon of combinatorially challenging open problems in network information theory and coding theory. Much like other members of this category such as index coding, regenerating codes and coded caching, we hope that variants of the multi-version coding set up will also see synergistic benefits with these problems, where technical developments in one of these problems fuel developments in the other. Importantly, while our work presents a balanced view with improvements in certain cases, and impossibility results precluding improvements in other cases, the fundamental limits of the problem are generally open for some regimes (specifically $\nu > 2$).

Secondly, our work motivates the interesting problems in distributed algorithm design for linearizable read-write data store design. Notably, the protocols [65, 66] exchange the data itself (not just meta-data) and it is worth examining whether the protocols can be modified for benign crash failures (considered in our work here) to involve just the exchange of meta-data information. A key-challenge in designing such protocol is that the node may not be able to differentiate between the case where its neighbors have not received a data version and the case where the neighbors indeed have received a data version, but the meta-data has not arrived yet due to the asynchrony in the system.

Chapter 4
Latency-Consistency Trade-off of Replication-Based Probabilistic Key-value Stores

Partial or probabilistic quorum systems are widely used in replication-based distributed key-value stores due to their latency benefits at the expense of providing weaker consistency guarantees as eventual consistency. Eventual consistency only guarantees that the clients will eventually retrieve consistent data if there are no new write operations, but it does not specify how fast will this happen. The probabilistically bounded staleness framework (PBS) [33, 34] studied this problem and investigated the latency-consistency trade-off of replication-based partial quorum systems through Monte Carlo event-based simulations. In order to tune the consistency and latency guarantees based on the expected network delays using a Monte Carlo approach, Monte Carlo simulations need to be carried out for all possible network delays. In this chapter, we study the latency-consistency trade-off of such systems analytically instead. We derive a simple closed-form expression for the inconsistency probability for 3-way replication in terms of the quorum sizes and the write and read mean delays assuming exponential delays. Our approach can be used to tune the latency and consistency guarantees in distributed key-value stores based on the write and read mean delays.

4.1 Introduction and Related Work

Key-value stores commonly replicate the data across multiple nodes to make the data available and accessible with low latency despite the possible failures or stragglers. In order to ensure strong consistency, these systems use strict quorum where the write and the read quorums must intersect. Specifically, in a system of n servers where c_W and c_R denote the write and the read quorum sizes respectively, c_W and c_R are chosen such that $c_W+c_R > n$. In order to have fast access to the data, many key-values stores including Amazon Dynamo [15], Apache Cassandra [16] and Riak [17] allow non-strict (partial, probabilistic or sloppy) quorums where $c_W + c_R \leq n$ to provide a lower write and/or read latency and only guarantee that the users will eventually return consistent data if there are no new write operations [18, 19]. However, eventual consistency does not specify how fast this will happen. Specifically, eventual consistency does not specify the probability of returning the value of the latest complete write for a read request issued t units of time after this write completes.

Several works studied probabilistic quorum systems, attempted to quantify the staleness of the data retrieved, how soon users can retrieve consistent data and providing adaptive consistency guarantees depending on the application including [13, 33, 34, 71–86]. In [13], ϵ-intersecting probabilistic quorum systems were designed such that the probability that any two quorums do not intersect is at most ϵ. In [79], an adaptive approach was proposed that tunes the inconsistency probability, assuming that the response time of the servers are neglected, through controlling the number of servers involved in the read operations at the run-time based on a monitoring module. The monitoring module provides a real-time estimate of the network delays. In this approach, the write operation completes when any server responds to the write client. While the data is being propagated to the remaining servers, any server is pessimistically considered stale except the first server that responded to the write operation. Hence, this approach does not fully capture expanding write quorums [87].

In [33, 34], the trade-off that partial quorum systems provide between the staleness of the retrieved data and the latency was studied in 3-way replication-based key-value stores. Specifically, this work answered the question of how stale is the retrieved data through the notion of l-staleness, which measures the probability that the users retrieve one of the latest complete l versions. The question of how eventual a user can read consistent data is also studied in [33, 34] through the notion of t-visibility that measures the probability of returning the value of a write operation t units of times after it completes. While the write operation completes upon receiving acknowledgments from any c_W servers, more servers receive the write request after that and hence the write quorum can continue to expand. Characterizing the t-visibility is challenging as it depends on how the write quorum expands based on the delays of the write and the read requests. Hence, the study of [34] focused on obtaining insights about this question for 3-way replication through Monte Carlo event-based simulations.

4.2 Contributions and Organization

In this chapter, we study the problem of providing probabilistic guarantees for partial quorum systems analytically for replication-based key-value stores. We study the inconsistency probability for such systems in terms of the quorum sizes, mean write and read delays assuming exponentially distributed write and read delays. For 3-way replication-based systems, we derive a closed-form expression for the inconsistency probability in terms of those parameters. Our approach can be used to fine-tune the consistency guarantees based on the network delays estimated by a monitoring module as suggested in [79].

The rest of this chapter is organized as follows. In Section 4.3, we describe the system model and provide a brief background about the order statistics of exponentially distributed random variables. In Section 4.4, we study expanding quorums that have a dynamic size and characterize the probability mass function of the quorum size. We study the inconsistency probability of replication-based partial quorum systems in Section 4.5. Finally, concluding remarks and open problems are discussed in Section 4.6.

4.3 System Model and Background

In this section, we describe our system model and provide a brief background about the order statistics and the sum exponential random variables.

4.3.1 System Model: Partial Quorums

We consider a distributed system with n servers denoted by $\mathcal{N} = \{1, 2, \cdots, n\}$ storing a shared object. A client that issues a write request sends the request to all servers and waits for the acknowledgment of c_W servers for the write operation to be considered complete. We observe that even after the write operation completes, the write quorum expands as more servers respond to the write request. We denote the time that a write request takes to reach to server i in addition to the server's response time by X_i, where $i \in \mathcal{N}$. We assume that $X_1, X_2, \cdots, X_n$ are independent and identically distributed exponential random variables with parameter λ. A client that issues a read request sends the request to all servers and waits for c_R servers to respond before completing the read operation. The time the read request takes to reach server i and the server's response time is denoted by $Z_i, i \in \mathcal{N}$. We assume that the read delays $Z_1, Z_2, \cdots, Z_n$ are independent and identically distributed random variables according to exponential distribution with parameter ξ. We also assume that write and read acknowledgments are instantaneous (See Remark 10).

We recall that in strict quorum systems, c_W and c_R are chosen such that $c_W + c_R > n$. In partial quorum systems however, $c_W + c_R \leq n$ and hence the write and the read quorums may not intersect. This may result in a consistency violation. In real-world quorum systems however, the write quorum expands as the write request propagate to more servers. In [33], the notion of t-visibility was developed which captures the probability of returning the value of a write operation for expanding quorums for a read operation that starts t units of time after this write completes. Formally, t-visibility consistency is defined as follows [33].

Definition 11 (t-visibility)**.** *A quorum system obeys t-visibility consistency if any read that starts t units of time after the write completes returns a value that is at least as recent as the value of that write with probability at least $1 - p_t$.*

Our goal in this work is to characterize the inconsistency probability p_t.

Remark 10 (Instantaneous Acknowledgments)**.** *While we assume that the write acknowledgments are instantaneous for simplicity, a deterministic delay of the acknowledgment denoted by d can be taken into account by studying the consistency $t + d$ units of time after c_W servers respond to the write request.*

Remark 11. *Our model can be also applied to key-value stores, where write and read clients sends the request to a coordinator. In this case, the coordinator forwards* *the request to all remaining servers and waits for the acknowledgments of* $c_W - 1$ *servers or* $c_R - 1$ *servers respectively before completing the operation [1, 2]. However, our model does not capture read repairs [1] in which when a read coordinator receives multiple versions of the data from different servers, it updates the stale servers with the most recent version of the data. Taking read repairs into account is an interesting future research direction.*

4.3.2 Background: Order Statistics and Sum of Exponentials

In this subsection, we provide a brief background about exponential random variables that we build on later in Section 4.4 to study expanding quorums. We first recall the following useful Lemma [88] for the order statistics of independent exponential random variables with a common parameter λ.

Lemma 4 (Order Statistics of Independent Exponentials). *Let* $X_1, X_2, \cdots, X_n$ *be independent and identically distributed random variables according to* $\exp(\lambda)$*, then we have*

$$Y_i \coloneqq X_{(i)} - X_{(i-1)} \sim \exp((n - i + 1)\lambda), \tag{4.1}$$

where $X_{(i)}$ *denotes the* i*-th smallest of* $X_1, X_2, \cdots, X_n$*,* $i \in \{1, 2, \cdots, n\}$ *and* $X_{(0)} = 0$*.*

We also recall the following Lemma from [89] which studies the sum of independent exponential random variables with different parameters.

Lemma 5 (Sum of Exponentials). *Let* $Y_1, Y_2, \cdots, Y_n$ *be independent exponentials random variables with parameters* $\lambda_1, \lambda_2, \cdots, \lambda_n$ *respectively, where* $f_{Y_i}(y)$ *denotes* *the density function of Y_i. The probability density function of*

$$Z \coloneqq \sum_{i=1}^{n} Y_i$$

is given by

$$f_Z(z) = \sum_{i=1}^{n} f_i(z) \prod_{\substack{j=1,\\ j\neq i}}^{n} \frac{\lambda_j}{\lambda_j - \lambda_i}, \;\; z \geq 0. \tag{4.2}$$

4.4 Expanding Quorums

In this section, we characterize the probability mass function (PMF) of the number of servers in the write quorum t units of time after the write completes. As we have explained, a client that issues a write request sends the request to all n servers and waits to receive acknowledgments from any c_W servers. The first c_W received responses determine the write latency $X_{(c_W)}$, but the write quorum will continue to expand as more servers receive the write request. We denote the set of servers that have received the write value t units of time after it completes by $\mathcal{S}(t)$, where $S(t) \coloneqq |\mathcal{S}(t)|$ and $S(0) = c_W$. In Theorem 8, we characterize the PMF of $S(t)$.

Theorem 8 (Dynamic Quorum Size)**.** *The PMF of the number of servers that have received a complete version t units of time after it completes, $S(t)$, is given by*

$$\Pr[S(t) = c_W] = e^{-\lambda_{c_W+1} t}, \tag{4.3}$$

$$\Pr[S(t) = s] = \sum_{i=c_W+1}^{s+1} (-1)^{s-i}(1 - e^{-\lambda_i t}) \binom{n - c_W}{n - i + 1} \binom{n - i + 1}{s - i + 1}, \tag{4.4}$$

for $s \in \{c_W + 1, c_W + 2, \cdots, n\}$, where $\lambda_i = (n - i + 1)\lambda$.

We provide the proof of Theorem 8 in Appendix D.

Since the replication factor is typically low in key-value stores and the 3-way replication is commonly used, we focus on this case in Corollary 3 which follows directly from Theorem 8.

Corollary 3 (Dynamic Quorum Size for $n = 3$)**.**

- *The PMF of the number of $S(t)$ for $c_W = 1$ is given by*

$$\Pr[S(t) = s] = \begin{cases} e^{-2\lambda t}, & \textit{if } s = 1, \\ 2(e^{-\lambda t} - e^{-2\lambda t}), & \textit{if } s = 2, \\ 1 - (2e^{-2\lambda t} - e^{-\lambda t}), & \textit{if } s = 3. \end{cases} \tag{4.5}$$

- *The PMF of the number of $S(t)$ for $c_W = 2$ is given by*

$$\Pr[S(t) = s] = \begin{cases} e^{-\lambda t}, & \textit{if } s = 2, \\ 1 - e^{-\lambda t}, & \textit{if } s = 3. \end{cases} \tag{4.6}$$

In Fig. 4.1, we show the PMF of $S(1)$ for $n = 3, c_W = 1$ and $\lambda = 1$. In Fig. 4.2, we show the PMF of $S(1)$ for $n = 3, c_W = 2$ and $\lambda = 1$.

4.5 Consistency Analysis

In this section, we study the inconsistency probability in replication-based partial quorum systems with expanding write quorums. The worst-case probability of inconsistency assuming non-expanding write quorums and instantaneous reads [33]

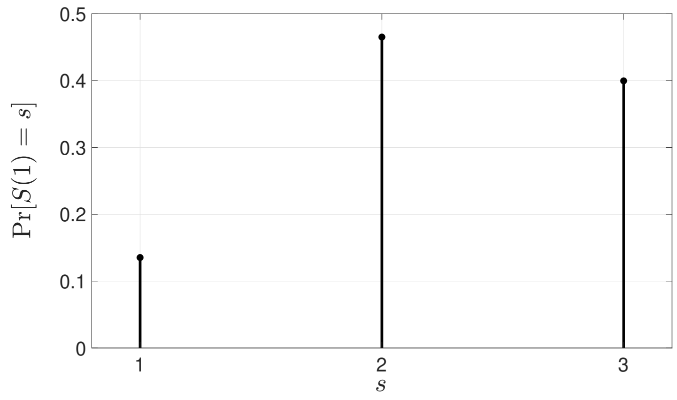


Figure 4.1. The PMF of $S(1)$ for the case where $\lambda = 1, n = 3$ and $c_W = 1$.

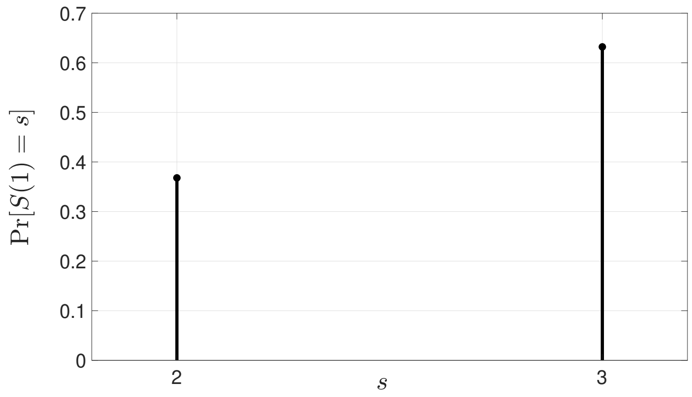


Figure 4.2. The PMF of $S(1)$ for the case where $\lambda = 1, n = 3$ and $c_W = 2$.

is given by

$$p = \frac{\binom{n-c_W}{c_R}}{\binom{n}{c_R}} \tag{4.7}$$

and the probability of not returning one of the latest complete l versions is at most p^l. Since write quorum expands as the write request propagate to more servers, equation (4.7) is in fact an upper bound of the inconsistency probability [33].

Our objective in this section is to characterize the exact inconsistency probability for expanding quorums. The read client returns inconsistent data if the first c_R

servers that respond to the read request return stale data. A server is considered stale if it replies to the read request before receiving the value of the latest compete write operation. That is, server i is stale if

$$X_{(c_W)} + t + Z_i < X_i.$$

Denote the first c_R servers that respond to the read request by $\mathcal{R} = \{r_1, r_2, \cdots, r_{c_R}\}$, where r_1 is the server the replies first, r_2 is the server that replies second and so on. The event that server r_j is stale is expressed as follows

$$\begin{aligned} E_j &= \{X_{(c_W)} + t + Z_{(j)} < X_{r_j}\} \\ &= \{r_j \notin \mathcal{S}(t + Z_{(j)})\}. \end{aligned} \tag{4.8}$$

where $j \in \mathcal{R}$. In order to keep the notation simple, we denote $\mathcal{S}(t + Z_{(j)})$ by $\mathcal{S}_j$. The probability that a read returns stale data t units of time after that latest version completes is the probability that all servers in $\mathcal{R}$ return stale data. Thus, the inconsistency probability can be expressed as follows

$$\begin{aligned} p_t &= \Pr[\text{All servers in } \mathcal{R} \text{ are stale}] \\ &= \Pr\left[\bigcap_{j=1}^{c_R} E_j\right] \\ &= \Pr\left[r_1 \notin \mathcal{S}_1, r_2 \notin \mathcal{S}_2, \cdots, r_{c_R} \notin \mathcal{S}_{c_R}\right]. \end{aligned} \tag{4.9}$$

Since the events $E_1, E_2, \cdots, E_{c_R}$ are dependent, we express the inconsistency probability as follows

$$p_t = \Pr\left[r_{c_R} \notin \mathcal{S}_{c_R} | r_{c_R-1} \notin \mathcal{S}_{c_R-1}, \cdots, r_1 \notin \mathcal{S}_1\right] \cdots \Pr\left[r_2 \notin \mathcal{S}_2 | r_1 \notin \mathcal{S}_1\right] \Pr\left[r_1 \notin \mathcal{S}_1\right]. \tag{4.10}$$

We first characterize the PMF of the number of servers in the write quorum $t + Z_{(j)}$ units of time after the write completes.

Lemma 6. *The PMF of the number of servers in the write quorum $t + Z_{(j)}$ units of time, where $j \in \mathcal{R}$, after the write completes is given by*

$$\Pr[S(t + Z_{(j)}) = c_W] = e^{-\lambda_{c_W+1} t} \sum_{l=1}^{j} \binom{n}{j}\binom{j}{l} \frac{(-1)^{j-l} \xi_{n-l+1}}{\xi_l + \lambda_{c_W+1}}, \tag{4.11}$$

$$\Pr[S(t + Z_{(j)}) = s] = \sum_{i=c_W+1}^{s+1} (-1)^{s-i} \binom{n - c_W}{n-i+1}\binom{n-i+1}{s-i+1} \left(1 - e^{-\lambda_i t} \sum_{l=1}^{j} \binom{n}{j}\binom{j}{l} \frac{(-1)^{j-l} \xi_{n-l+1}}{\xi_l + \lambda_i}\right), \tag{4.12}$$

for $s \in \{c_W + 1, \cdots, n\}$, where $\xi_j = (n - j + 1)\xi$ and $\lambda_j = (n - j + 1)\lambda$.

The proof of Lemma 6 is straightforward, but we provide it in Appendix E for completeness. In Theorem 9, we provide our main result in which we characterize the inconsistency probability of the widely-used 3-way replication technique.

Theorem 9 (Inconsistency Probability of Replication-based Systems with $n = 3$)**.**

- *The worst-case inconsistency probability for the case where $c_W = 1$ and $c_R = 1$ is given by*

$$p_t = \frac{2\xi e^{-\lambda t}}{\lambda + 3\xi}. \tag{4.13}$$

- *The worst-case inconsistency probability for the case where $c_W = 2$ and $c_R = 1$ is given by*

$$p_t = \frac{\xi e^{-\lambda t}}{\lambda + 3\xi}. \tag{4.14}$$

- *The worst-case inconsistency probability for the case where $c_W = 1$ and $c_R = 2$ is given by*

$$p_t = \frac{6\xi^3 e^{-2\lambda t}}{(\lambda + 2\xi)(\lambda + 3\xi)} \left(\frac{2\lambda}{(\lambda + 2\xi)(\lambda + 3\xi)} - \frac{(\lambda - \xi)e^{-\lambda t}}{(\lambda + \xi)(2\lambda + 3\xi)} \right). \tag{4.15}$$

We provide the proof of Theorem 9 in Appendix F. We show the probability of inconsistency for the different cases in Fig. 4.3, Fig. 4.4 and Fig. 4.5 as a function of t.

Remark 12. *It can be verified that at $t = 0$, the limit of the inconsistency probability of Theorem 9 as $\xi \to \infty$ is equal to the inconsistency probability assuming instantaneous reads given in (4.7). That is, we have*

$$\lim_{\xi \to \infty} p_0 = p. \tag{4.16}$$

Remark 13. *It is worth noting that the upper bound of the inconsistency probability given in (4.7) is loose. In order to see this, we observe that this bound gives an inconsistency probability of $1/3$ for the case where $c_W = 2, c_R = 1$ and also for the case where $c_W = 1, c_R = 2$. Hence, this upper bound does not differentiate between these two cases.*

Remark 14 (Asymmetry)**.** *It is worth noting that the inconsistency probability given in Theorem 9 is asymmetric in the write and read quorum sizes and also the*

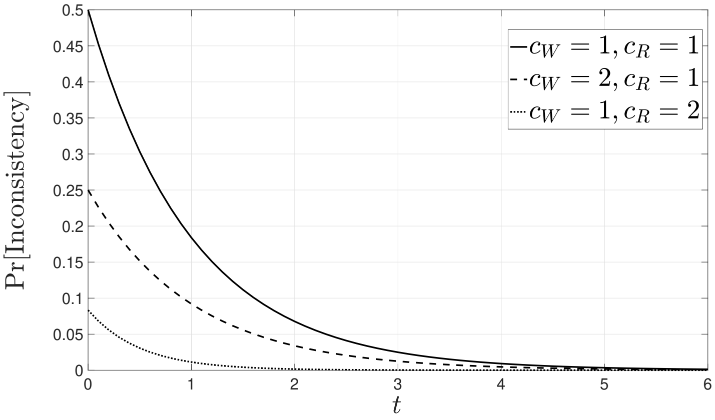


Figure 4.3. The probability of inconsistency for the case where $n = 3, \lambda = 1$ and $\xi = 1$.

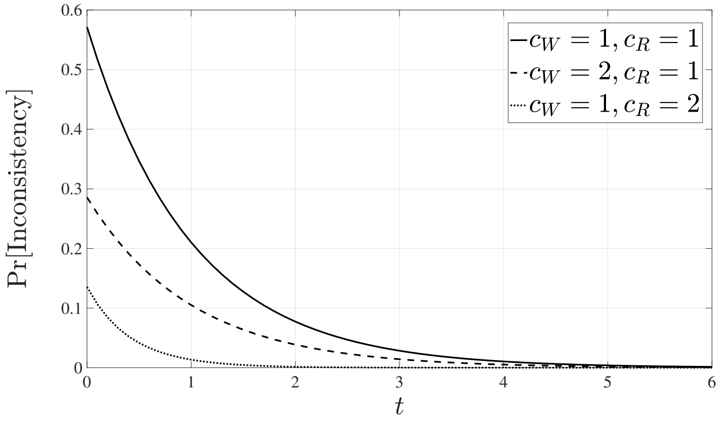


Figure 4.4. The probability of inconsistency for the case where $n = 3, \lambda = 1$ and $\xi = 2$.

write and read mean delays. In particular, we notice that the mean delay of the write operations is more crucial in determining the inconsistency probability. This suggests that the write operations should be given a higher priority in distributed key-value stores. In addition, the read quorum size is more crucial to the inconsistency probability as compared with the write quorum size.

Remark 15 (Replication Factor)**.** *While the case of $n = 3$ is the typical case in replication-based systems, our approach can be also used to derive a closed-form expression for the inconsistency probability for general n, c_W and c_R. In general,*

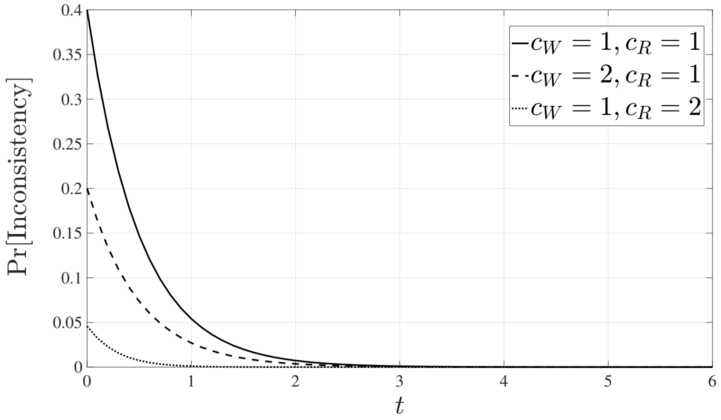
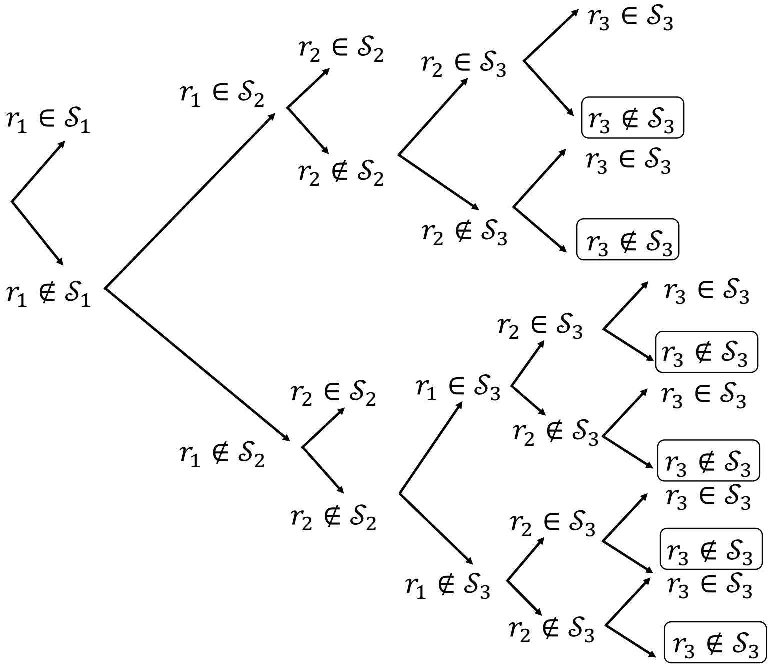


Figure 4.5. The probability of inconsistency for the case where $n = 3, \lambda = 2$ and $\xi = 1$.

there are $c_R!$ cases to be considered. For the case where $c_R = 3$, for instance, the



$r_1 \in \mathcal{S}_1$
$r_1 \notin \mathcal{S}_1$
$r_1 \in \mathcal{S}_2$
$r_1 \notin \mathcal{S}_2$
$r_2 \in \mathcal{S}_2$
$r_2 \notin \mathcal{S}_2$
$r_2 \in \mathcal{S}_3$
$r_2 \notin \mathcal{S}_3$
$r_3 \in \mathcal{S}_3$
$r_3 \notin \mathcal{S}_3$
$r_3 \in \mathcal{S}_3$
$r_3 \notin \mathcal{S}_3$
$r_2 \in \mathcal{S}_2$
$r_2 \notin \mathcal{S}_2$
$r_1 \in \mathcal{S}_3$
$r_1 \notin \mathcal{S}_3$
$r_2 \in \mathcal{S}_3$
$r_2 \notin \mathcal{S}_3$
$r_2 \in \mathcal{S}_3$
$r_2 \notin \mathcal{S}_3$
$r_3 \in \mathcal{S}_3$
$r_3 \notin \mathcal{S}_3$
$r_3 \in \mathcal{S}_3$
$r_3 \notin \mathcal{S}_3$
$r_3 \in \mathcal{S}_3$
$r_3 \notin \mathcal{S}_3$
$r_3 \in \mathcal{S}_3$
$r_3 \notin \mathcal{S}_3$

Figure 4.6. Inconsistency cases for the case where $c_R = 3$.

following cases shown in Fig. 4.6 lead to violating the consistency requirements

1. $(r_3 \notin \mathcal{S}_3, r_2 \in \mathcal{S}_3 - \mathcal{S}_2, r_1 \in \mathcal{S}_2 - \mathcal{S}_1)$,

2. $(r_3 \notin \mathcal{S}_3, r_2 \notin \mathcal{S}_3, r_1 \in \mathcal{S}_2 - \mathcal{S}_1)$,

3. $(r_3 \notin \mathcal{S}_3, r_2 \in \mathcal{S}_3 - \mathcal{S}_2, r_1 \in \mathcal{S}_3 - \mathcal{S}_2)$,

4. $(r_3 \notin \mathcal{S}_3, r_2 \notin \mathcal{S}_3, r_1 \in \mathcal{S}_3 - \mathcal{S}_2)$,

5. $(r_3 \notin \mathcal{S}_3, r_2 \in \mathcal{S}_3 - \mathcal{S}_2, r_1 \notin \mathcal{S}_3)$,

6. $(r_3 \notin \mathcal{S}_3, r_2 \notin \mathcal{S}_3, r_1 \notin \mathcal{S}_3)$.

Hence, the inconsistency probability can be expressed as follows

$$\begin{aligned} p_t = {} & \Pr[r_3 \notin \mathcal{S}_3, r_2 \in \mathcal{S}_3 - \mathcal{S}_2, r_1 \in \mathcal{S}_2 - \mathcal{S}_1] + \Pr[r_3 \notin \mathcal{S}_3, r_2 \notin \mathcal{S}_3, r_1 \in \mathcal{S}_2 - \mathcal{S}_1] \\ & + \Pr[r_3 \notin \mathcal{S}_3, r_2 \in \mathcal{S}_3 - \mathcal{S}_2, r_1 \in \mathcal{S}_3 - \mathcal{S}_2] + \Pr[r_3 \notin \mathcal{S}_3, r_2 \notin \mathcal{S}_3, r_1 \in \mathcal{S}_3 - \mathcal{S}_2] \\ & + \Pr[r_3 \notin \mathcal{S}_3, r_2 \in \mathcal{S}_3 - \mathcal{S}_2, r_1 \notin \mathcal{S}_3] + \Pr[r_3 \notin \mathcal{S}_3, r_2 \notin \mathcal{S}_3, r_1 \notin \mathcal{S}_3], \end{aligned} \tag{4.17}$$

where

$$\begin{aligned} \Pr[r_3 \notin \mathcal{S}_3, r_2 \in \mathcal{S}_3 - \mathcal{S}_2, r_1 \in \mathcal{S}_2 - \mathcal{S}_1] = {} & \Pr[r_3 \notin \mathcal{S}_3 | r_2 \in \mathcal{S}_3, r_1 \in \mathcal{S}_3] \\ & (\Pr[r_2 \in \mathcal{S}_3 | r_1 \in \mathcal{S}_3] - \Pr[r_2 \in \mathcal{S}_2 | r_1 \in \mathcal{S}_2]) \\ & (\Pr[r_1 \in \mathcal{S}_2] - \Pr[r_1 \in \mathcal{S}_1]), \\ \Pr[r_3 \notin \mathcal{S}_3, r_2 \notin \mathcal{S}_3, r_1 \in \mathcal{S}_2 - \mathcal{S}_1] = {} & \Pr[r_3 \notin \mathcal{S}_3 | r_2 \notin \mathcal{S}_3, r_1 \in \mathcal{S}_3] \\ & \Pr[r_2 \notin \mathcal{S}_3 | r_1 \in \mathcal{S}_3] (\Pr[r_1 \in \mathcal{S}_2] - \Pr[r_1 \in \mathcal{S}_1]), \\ \Pr[r_3 \notin \mathcal{S}_3, r_2 \in \mathcal{S}_3 - \mathcal{S}_2, r_1 \in \mathcal{S}_3 - \mathcal{S}_2] = {} & \Pr[r_3 \notin \mathcal{S}_3 | r_2 \in \mathcal{S}_3, r_1 \in \mathcal{S}_3] \\ & (\Pr[r_2 \in \mathcal{S}_3 | r_1 \in \mathcal{S}_3] - \Pr[r_2 \in \mathcal{S}_2 | r_1 \notin \mathcal{S}_2]) \\ & (\Pr[r_1 \in \mathcal{S}_3] - \Pr[r_1 \in \mathcal{S}_2]), \\ \Pr[r_3 \notin \mathcal{S}_3, r_2 \notin \mathcal{S}_3, r_1 \in \mathcal{S}_3 - \mathcal{S}_2] = {} & \Pr[r_3 \notin \mathcal{S}_3 | r_2 \notin \mathcal{S}_3, r_1 \in \mathcal{S}_3] \\ & \Pr[r_2 \notin \mathcal{S}_3 | r_1 \in \mathcal{S}_3] \\ & (\Pr[r_1 \in \mathcal{S}_3] - \Pr[r_1 \in \mathcal{S}_2]), \end{aligned}$$

$$
\begin{aligned}
\Pr[r_3 \notin \mathcal{S}_3, r_2 \in \mathcal{S}_3 - \mathcal{S}_2, r_1 \notin \mathcal{S}_3] = {} & \Pr[r_3 \notin \mathcal{S}_3 | r_2 \in \mathcal{S}_3, r_1 \notin \mathcal{S}_3] \\
& (\Pr[r_2 \in \mathcal{S}_3 | r_1 \notin \mathcal{S}_3] - \Pr[r_2 \in \mathcal{S}_2 | r_1 \notin \mathcal{S}_2]) \\
& \Pr[r_1 \notin \mathcal{S}_3], \\
\Pr[r_3 \notin \mathcal{S}_3, r_2 \notin \mathcal{S}_3, r_1 \notin \mathcal{S}_3] = {} & \Pr[r_3 \notin \mathcal{S}_3 | r_2 \notin \mathcal{S}_3, r_1 \notin \mathcal{S}_3] \\
& \Pr[r_2 \notin \mathcal{S}_3 | r_1 \notin \mathcal{S}_3] \Pr[r_1 \notin \mathcal{S}_3], \qquad (4.18)
\end{aligned}
$$

and each term can be expressed using Theorem 8 to get a closed-form expression for the inconsistency probability in this case.

4.6 Conclusion

In this chapter, we have studied the latency-consistency trade-off of replication-based key-value stores analytically and derived simple a closed-form expression for the inconsistency probability for the 3-way replication technique assuming exponential write and read delays. Our study allows tuning the latency and consistency guarantees based on the mean values of the write and read delays of the data store. An immediate future work is to incorporate our tuning policy in a distributed key-value store and evaluate the performance. Extending this study to derive a tight upper bound on the inconsistency probability for any given distributions of the write delays, read delays, non-deterministic acknowledgments delays and read repairs are also interesting future research directions.

Chapter 5
Latency-Consistency Trade-off of Erasure-Coded Probabilistic Key-value Stores

The probabilistically bounded staleness framework (PBS) [33, 34] studied the latency-consistency trade-off of replication-based probabilistic key-value stores through Monte Carlo simulations. In Chapter 4, we studied the latency-consistency trade-off of such systems analytically and derived a closed-form expression for the inconsistency probability for 3-way replication. In this work, we extend this study and consider erasure-coded probabilistic key-value stores. Specifically, we consider a simple erasure-coded key-value store with n servers, where each version is encoded using a maximum distance separable (MDS) code of dimension k. In order to ensure strong consistency, strict quorums are used where the write quorum size c_W and the read quorum size c_R must be selected such that $c_W + c_R - n \geq k$. In probabilistic simple erasure-coded key-value stores, the quorum sizes can be chosen such that $c_W + c_R - n < k$ to enhance the latency and only provide weaker consistency guarantees. We study this latency-consistency trade-off and show how the inconsistency probability of such systems can be derived in terms of the quorum sizes, the write and read mean delays and the dimension of the erasure code k. We also provide examples showing that probabilistic erasure-coded systems enrich the latency-consistency trade-off and provide more flexibility as compared with replication-based systems through controlling the dimension of the erasure code k.

5.1 Introduction

Much research has considered eventual consistency, probabilistic quorums and the latency-consistency trade-off in replication-based key-value stores [13, 33, 34, 71–81, 83–86, 90], but only few works studied these aspects in erasure-coded key-value stores including [91]. In simple erasure-coded strict quorum systems, each version of the data is encoded using a maximum distance separable (MDS) code of dimension k and c_W and c_R are chosen such that $c_W + c_R - n \geq k$. In erasure-coded probabilistic quorum systems however, the quorum sizes may be selected such that $c_W + c_R - n < k$ to provide faster access to the data. Hence, the intersection of the write and read quorums may contain less than k servers and this may result in a violation of the strong consistency. While the write operation is considered complete once c_W server respond to the write request, and that time determines the write latency, more servers will continue to respond to the write request after that and the write quorum may continue to expand. In a simple erasure-coded system, unlike replication-based systems, the write client needs only to send $1/k$ of the data to each server in the system. Using an MDS code of dimension $k > 1$ reduces the communication cost of the write and read operations, can also minimize the storage cost [30] and can allow the write quorum to expand faster. However, this may result in a higher probability of inconsistency as compared with replication as the write and read quorums need to intersect in at least k servers and not only one server as in replication. Our objective in this work is to study the effect of the dimension of the MDS code k on the inconsistency probability of probabilistic simple erasure-coded key-value stores.

The rest of this chapter is organized as follows. In Section 5.2, we present our system model. In Section 5.3, we show how the inconsistency probability can be characterized in erasure-coded probabilistic key-value stores. Section 5.4 presents an example illustrating how to choose the dimension of the underlying MDS code in distributed key-value stores. Finally, concluding remarks and open problems are discussed in Section 5.5.

5.2 System Model and Background

In this section, we describe our system model. We consider a distributed erasure-coded storage system with n servers denoted by $\mathcal{N} = \{1, 2, \cdots, n\}$ storing a shared object. A client that issues a write request sends the request to all servers and waits for the acknowledgment of c_W servers for the write request to complete. We consider a simple erasure coding scheme, where each server stores $1/k$ of the each version that it receives using an (n, k) maximum distance separable (MDS) code. Thus, the write client sends only $1/k$ of the data version and not the entire data version to each server. We denote the time that a write request takes to reach to server i in addition to the server's response time by X_i, where $i \in \mathcal{N}$. We assume that $X_1, X_2, \cdots, X_n$ are independent and identically distributed according to exponential distribution with parameter $k\lambda$. We denote the j-th smallest of $X_1, X_2, \cdots, X_n$ by $X_{(j)}$. A client that issues a read request sends the request to all servers and waits for c_R servers to respond before completing the read operation. The time the read request takes to reach server i and the server's response time is denoted by $Z_i, i \in \mathcal{N}$. We assume that the read delays $Z_1, Z_2, \cdots, Z_n$ are independent and identically distributed random variables according to exponential distribution with parameter $k\xi$. We denote the j-th smallest of $Z_1, Z_2, \cdots, Z_n$ by

$Z_{(j)}$. Finally, we assume that write and read acknowledgments are instantaneous.

In strict quorum simple erasure-coded systems, c_W and c_R are chosen such that $c_W + c_R - n \geq k$. In probabilistic quorum simple erasure-coded systems however, $c_W + c_R - n < k$ and hence the write and the read quorums may intersect in less than k servers. This may lead to read requests returning inconsistent data. Although the write operation completes once the write client receives acknowledgments from c_W servers at time $X_{(c_W)}$, the write quorum expands as the write request propagate to more servers after that. In [33], the notion of t-visibility was introduced for replication-based systems which captures the probability of inconsistency for expanding quorums for a read operation that starts t units of time after the write completes. We denote the inconsistency probability for a read operation issued t units of time after the latest complete write operation by p_t. We denote the set of servers that have received the write value t units of time after it completes by $\mathcal{S}(t)$, where $S(t) \coloneqq |\mathcal{S}(t)|$ and $S(0) = c_W$. We denote the write quorum at time $t + Z_{(j)}$, $\mathcal{S}(t + Z_{(j)})$, by $\mathcal{S}_j$ for simplicity. Finally, we recall from Theorem 8, proved in Appendix D, that the probability mass function (PMF) of the number of servers that have received a complete version t units of time after it completes, $S(t)$, is given by

$$\Pr[S(t) = c_W] = e^{-\lambda_{c_W+1} t}, \tag{5.1}$$

$$\Pr[S(t) = s] = \sum_{i=c_W+1}^{s+1} (-1)^{s-i} (1 - e^{-\lambda_i t}) \binom{n - c_W}{n - i + 1} \binom{n - i + 1}{s - i + 1}, \tag{5.2}$$

for $s \in \{c_W + 1, c_W + 2, \cdots, n\}$, where $\lambda_i = (n - i + 1)k\lambda$. We also recall from Lemma 6, proved in Appendix E, that the PMF of the number of servers in the write quorum $t + Z_{(j)}$ units of time, where $j \in \mathcal{R}$, after the write completes is expressed as follows

$$\Pr[S(t+Z_{(j)})=c_W]=e^{-\lambda_{c_W+1}t}\sum_{l=1}^{j}\binom{n}{j}\binom{j}{l}\frac{(-1)^{j-l}\xi_{n-l+1}}{\xi_l+\lambda_{c_W+1}}, \tag{5.3}$$

$$\Pr[S(t+Z_{(j)})=s]=\sum_{i=c_W+1}^{s+1}(-1)^{s-i}\binom{n-c_W}{n-i+1}\binom{n-i+1}{s-i+1} \left(1-e^{-\lambda_i t}\sum_{l=1}^{j}\binom{n}{j}\binom{j}{l}\frac{(-1)^{j-l}\xi_{n-l+1}}{\xi_l+\lambda_i}\right), \tag{5.4}$$

for $s\in\{c_W+1,\cdots,n\}$, where $\xi_j=k(n-j+1)\xi$ and $\lambda_j=k(n-j+1)\lambda$.

5.3 Consistency Analysis

In this section, we study the probability of inconsistency for simple erasure-coded probabilistic quorum systems with expanding quorums. Assuming non-expanding write quorums and instantaneous reads it is straightforward to see that the inconsistency probability is given by

$$p=\sum_{i=\max(c_W+c_R-n,0)}^{k-1}\frac{\binom{c_W}{i}\binom{n-c_W}{c_R-i}}{\binom{n}{c_R}}. \tag{5.5}$$

We note that equation (5.5) gives only an upper bound on the inconsistency probability for expanding quorums and we aim to find the exact probability of inconsistency assuming expanding write quorums.

A read client returns inconsistent data if there are at least c_R-k+1 servers in the read quorum that respond with stale data. The coded data of server is stale if the server replies to the read request before receiving the coded data corresponding to the latest compete version. We denote the first c_R servers that respond to the read request by $\mathcal{R}=\{r_1,r_2,\cdots,r_{c_R}\}$, where r_1 is the server the replies first, r_2

is the server that replies second and so on. The event that server r_j is stale is expressed as follows

$$E_j = \left\{ X_{(c_W)} + t + Z_{(j)} < X_{r_j} \right\} = \{ r_j \notin \mathcal{S}_j \}. \tag{5.6}$$

where $j \in \mathcal{R}$. We also define a binary indicator random variable for each server in the read quorum as follows

$$I_j = \begin{cases} 1 & \text{if } r_j \notin \mathcal{S}_j, \\ 0 & \text{otherwise.} \end{cases} \tag{5.7}$$

Therefore, we can express the inconsistency probability as follows

$$\begin{aligned} p_t &= \Pr\left[\text{At least } c_R - k + 1 \text{ servers in } \mathcal{R} \text{ are stale}\right] \\ &= \Pr\left[\sum_{j=1}^{c_R} I_j \geq c_R - k + 1 \right]. \end{aligned} \tag{5.8}$$

For the case where $c_R = 3$, for instance, and $k = 2$ shown in Fig. 5.1, we can express the inconsistency probability as follows

$$\begin{aligned} p_t = {} & \Pr[r_3 \notin \mathcal{S}_3, r_2 \in \mathcal{S}_3 - \mathcal{S}_2, r_1 \in \mathcal{S}_1] + \Pr[r_3 \notin \mathcal{S}_3, r_2 \notin \mathcal{S}_3, r_1 \in \mathcal{S}_1] \\ & + \Pr[r_3 \notin \mathcal{S}_3, r_2 \in \mathcal{S}_2, r_1 \in \mathcal{S}_2 - \mathcal{S}_1] + \Pr[r_2 \notin \mathcal{S}_2, r_1 \in \mathcal{S}_2 - \mathcal{S}_1] \\ & + \Pr[r_3 \notin \mathcal{S}_3, r_2 \in \mathcal{S}_2, r_1 \in \mathcal{S}_3 - \mathcal{S}_2] + \Pr[r_3 \notin \mathcal{S}_3, r_2 \in \mathcal{S}_2, r_1 \notin \mathcal{S}_3] \\ & + \Pr[r_2 \notin \mathcal{S}_2, r_1 \notin \mathcal{S}_2]. \end{aligned} \tag{5.9}$$

In order to express the inconsistency probability in this case, we note that the first

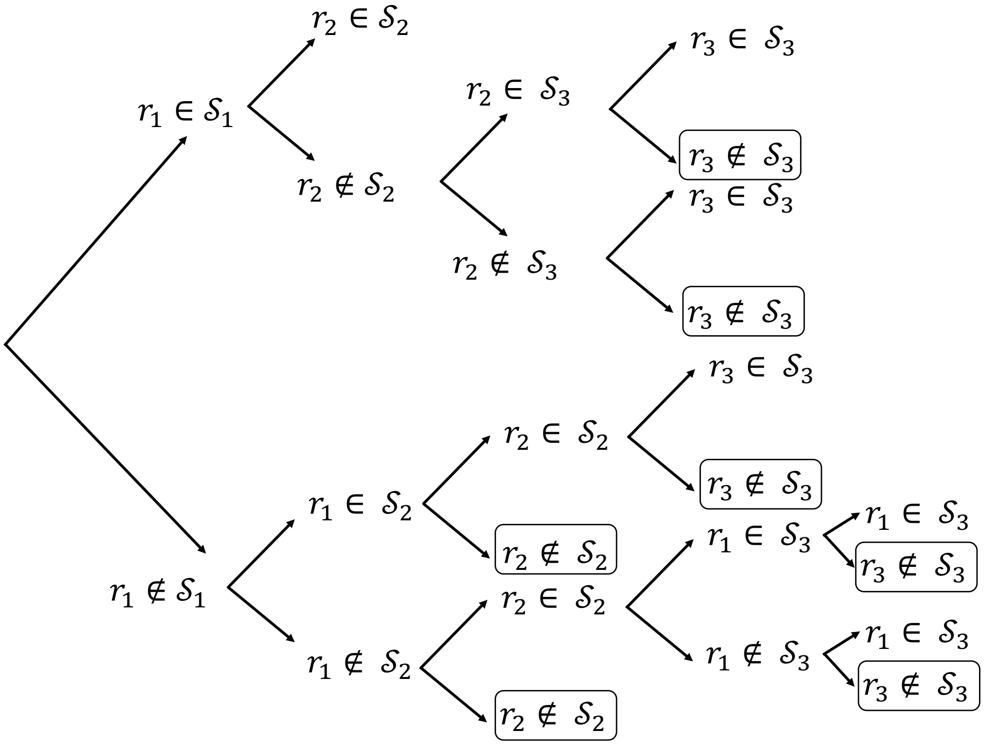

$r_2 \in \mathcal{S}_2$
$r_3 \in \mathcal{S}_3$
$r_2 \in \mathcal{S}_3$
$r_1 \in \mathcal{S}_1$
$r_3 \notin \mathcal{S}_3$
$r_2 \notin \mathcal{S}_2$
$r_3 \in \mathcal{S}_3$
$r_2 \notin \mathcal{S}_3$
$r_3 \notin \mathcal{S}_3$
$r_3 \in \mathcal{S}_3$
$r_2 \in \mathcal{S}_2$
$r_3 \notin \mathcal{S}_3$
$r_1 \in \mathcal{S}_2$
$r_1 \in \mathcal{S}_3$
$r_1 \in \mathcal{S}_3$
$r_2 \notin \mathcal{S}_2$
$r_3 \notin \mathcal{S}_3$
$r_1 \notin \mathcal{S}_1$
$r_2 \in \mathcal{S}_2$
$r_1 \in \mathcal{S}_3$
$r_1 \notin \mathcal{S}_3$
$r_1 \notin \mathcal{S}_2$
$r_3 \notin \mathcal{S}_3$
$r_2 \notin \mathcal{S}_2$

Figure 5.1. Inconsistency cases for the case where $c_R = 3$ and $k = 2$.

term in (5.9) can be expressed as follows

$$\begin{aligned}\Pr[r_3 \notin \mathcal{S}_3, r_2 \in \mathcal{S}_3 - \mathcal{S}_2, r_1 \in \mathcal{S}_1] &= \Pr[r_3 \notin \mathcal{S}_3 | r_2 \in \mathcal{S}_3, r_1 \in \mathcal{S}_3] \\ &\quad (\Pr[r_2 \in \mathcal{S}_3 | r_1 \in \mathcal{S}_3] - \Pr[r_2 \in \mathcal{S}_2 | r_1 \in \mathcal{S}_2]) \Pr[r_1 \in \mathcal{S}_1]. \quad (5.10)\end{aligned}$$

The second term in (5.9) can be expressed as follows

$$\begin{aligned}\Pr[r_3 \notin \mathcal{S}_3, r_2 \notin \mathcal{S}_3, r_1 \in \mathcal{S}_1] &= \Pr[r_3 \notin \mathcal{S}_3 | r_2 \notin \mathcal{S}_3, r_1 \in \mathcal{S}_3] \\ &\quad \Pr[r_2 \notin \mathcal{S}_3 | r_1 \in \mathcal{S}_3] \Pr[r_1 \in \mathcal{S}_1]. \quad (5.11)\end{aligned}$$

The third term in (5.9) can be expressed as follows

$$\begin{aligned}\Pr[r_3 \notin \mathcal{S}_3, r_2 \in \mathcal{S}_2, r_1 \in \mathcal{S}_2 - \mathcal{S}_1] = \Pr[r_3 \notin \mathcal{S}_3 | r_2 \in \mathcal{S}_3, r_1 \in \mathcal{S}_3] \Pr[r_2 \in \mathcal{S}_2 | r_1 \in \mathcal{S}_2]&\\ (\Pr[r_1 \in \mathcal{S}_2] - \Pr[r_1 \in \mathcal{S}_1]).& \qquad (5.12)\end{aligned}$$

The fourth term in (5.9) can be expressed as follows

$$\Pr[r_2 \notin \mathcal{S}_2, r_1 \in \mathcal{S}_2 - \mathcal{S}_1] = \Pr[r_2 \notin \mathcal{S}_2 | r_1 \in \mathcal{S}_2](\Pr[r_1 \in \mathcal{S}_2] - \Pr[r_1 \in \mathcal{S}_1]). \quad (5.13)$$

The fifth term in (5.9) can be expressed as follows

$$\begin{aligned}\Pr[r_3 \notin \mathcal{S}_3, r_2 \in \mathcal{S}_2, r_1 \in \mathcal{S}_3 - \mathcal{S}_2] =& \Pr[r_3 \notin \mathcal{S}_3 | r_2 \in \mathcal{S}_3, r_1 \in \mathcal{S}_3]\\ &\Pr[r_2 \in \mathcal{S}_2 | r_1 \notin \mathcal{S}_2]\\ &(\Pr[r_1 \in \mathcal{S}_3] - \Pr[r_1 \in \mathcal{S}_2]). \qquad (5.14)\end{aligned}$$

The sixth term in (5.9) is expressed as follows

$$\begin{aligned}\Pr[r_3 \notin \mathcal{S}_3, r_2 \in \mathcal{S}_2, r_1 \notin \mathcal{S}_3] =& \Pr[r_3 \notin \mathcal{S}_3 | r_2 \in \mathcal{S}_3, r_1 \notin \mathcal{S}_3]\\ &\Pr[r_2 \in \mathcal{S}_2 | r_1 \notin \mathcal{S}_2] \Pr[r_1 \notin \mathcal{S}_3]. \qquad (5.15)\end{aligned}$$

Finally, the last term in (5.9) is expressed as follows

$$\Pr[r_2 \notin \mathcal{S}_2, r_1 \notin \mathcal{S}_2] = \Pr[r_2 \notin \mathcal{S}_2 | r_1 \notin \mathcal{S}_2] \Pr[r_1 \notin \mathcal{S}_2]. \qquad (5.16)$$

For $c_R = 3$ and $k = 3$, we can express the inconsistency probability as follows

$$p_t = 1 - \Pr[r_3 \in \mathcal{S}_3, r_2 \in \mathcal{S}_2, r_1 \in \mathcal{S}_1]$$

$$= 1 - \Pr[r_3 \in \mathcal{S}_3 | r_2 \in \mathcal{S}_3, r_1 \in \mathcal{S}_3] \Pr[r_2 \in \mathcal{S}_2 | r_1 \in \mathcal{S}_2] \Pr[r_1 \in \mathcal{S}_1]. \tag{5.17}$$

5.4 Numerical Example

We now give a numerical example showing the benefits of erasure-coded probabilistic quorum systems as compared with replication-based quorum systems. In this example, we fix the write and the read quorum sizes for all schemes to fix the number of stragglers (slow servers) that the system can tolerate for all schemes.

Example 7 (Replication-based and simple erasure-coded distributed key-value stores)**.** *Consider a distributed storage system with $n = 5$ servers and $c_W = c_R = 3$. We study a replication-based system with replication factor of 5, a simple erasure-coded system with $k = 2$ and a simple erasure-coded system with $k = 3$.*

- *In replication-based system, the write and read quorums always intersect and hence strong consistency is always satisfied.*

- *In a simple erasure-coded system with $k = 2$, we first need to calculate the following terms to characterize the inconsistency probability.*

$$\begin{aligned}\Pr[r_3 \notin \mathcal{S}_3 | r_2 \in \mathcal{S}_3, r_1 \in \mathcal{S}_3] &= \sum_{s=c_W}^{n} \left(1 - \frac{s-2}{n-2}\right) \Pr[S(t + Z_{(3)}) = s] \\ &= \frac{40(2\xi)^3 e^{-2\lambda t}}{(2\lambda + 6\xi)(2\lambda + 8\xi)(2\lambda + 10\xi)}.\end{aligned}$$

$$\begin{aligned}\Pr[r_3 \notin \mathcal{S}_3 | r_2 \notin \mathcal{S}_3, r_1 \in \mathcal{S}_3] &= \sum_{s=c_W}^{n-1} \left(1 - \frac{s-1}{n-2}\right) \Pr[S(t + Z_{(3)}) = s] \\ &= \frac{20(2\xi)^3 e^{-4\lambda t}}{(4\lambda + 6\xi)(4\lambda + 8\xi)(4\lambda + 10\xi)}.\end{aligned}$$

$$\begin{aligned}\Pr[r_2 \in \mathcal{S}_3 | r_1 \in \mathcal{S}_3] &= \sum_{s=c_W}^{n} \left(\frac{s-1}{n-1}\right) \Pr[S(t+Z_{(3)}) = s] \\ &= 1 - \frac{30(2\xi)^3 e^{-2\lambda t}}{(2\lambda+6\xi)(2\lambda+8\xi)(2\lambda+10\xi)}.\end{aligned}$$

$$\begin{aligned}\Pr[r_2 \in \mathcal{S}_2 | r_1 \in \mathcal{S}_2] &= \sum_{s=c_W}^{n} \left(\frac{s-1}{n-1}\right) \Pr[S(t+Z_{(2)}) = s] \\ &= 1 - \frac{10(2\xi)^2 e^{-2\lambda t}}{(2\lambda+8\xi)(2\lambda+10\xi)}.\end{aligned}$$

$$\begin{aligned}\Pr[r_2 \notin \mathcal{S}_2 | r_1 \notin \mathcal{S}_2] &= \sum_{s=c_W}^{n-1} \left(1 - \frac{s}{n-1}\right) \Pr[S(t+Z_{(2)}) = s] \\ &= \frac{5(2\xi)^2 e^{-4\lambda t}}{(4\lambda+8\xi)(4\lambda+10\xi)}.\end{aligned}$$

$$\Pr[r_1 \in \mathcal{S}_1] = \sum_{s=c_W}^{n} \frac{s}{n} \Pr[S(t+Z_{(1)}) = s] = 1 - \frac{4\xi e^{-2\lambda t}}{2\lambda+10\xi}.$$

$$\Pr[r_1 \in \mathcal{S}_2] = \sum_{s=c_W}^{n} \frac{s}{n} \Pr[S(t+Z_{(2)}) = s] = 1 - \frac{8(2\xi)^2 e^{-2\lambda t}}{(2\lambda+8\xi)(2\lambda+10\xi)}.$$

$$\Pr[r_1 \in \mathcal{S}_3] = \sum_{s=c_W}^{n} \frac{s}{n} \Pr[S(t+Z_{(3)}) = s] = 1 - \frac{24(2\xi)^3 e^{-2\lambda t}}{(2\lambda+6\xi))(2\lambda+8\xi)(2\lambda+10\xi)}.$$

The inconsistency probability p_t can then be expressed using (5.9).

In this case, using the upper bound of (5.5) gives an inconsistency probability *of* $p = 3/10$ *and it can be verified that*

$$\lim_{\xi \to \infty} p_0 = 3/10. \tag{5.18}$$

- *In a simple erasure-coded system with* $k = 3$*, we first need to calculate the following terms to get a closed-form expression of the inconsistency probability.*

$$\Pr[r_3 \in \mathcal{S}_3 | r_2 \in \mathcal{S}_3, r_1 \in \mathcal{S}_3] = 1 - \frac{40(3\xi)^3 e^{-3\lambda t}}{(3\lambda + 9\xi)(3\lambda + 12\xi)(3\lambda + 15\xi)}.$$

$$\Pr[r_2 \in \mathcal{S}_2 | r_1 \in \mathcal{S}_2] = 1 - \frac{10(3\xi)^2 e^{-3\lambda t}}{(3\lambda + 12\xi)(3\lambda + 15\xi)}.$$

$$\Pr[r_1 \in \mathcal{S}_1] = 1 - \frac{6\xi e^{-3\lambda t}}{3\lambda + 15\xi}.$$

The inconsistency probability p_t *can be then expressed using (5.17).*

In this case, using the upper bound of (5.5) gives an inconsistency probability of $p = 9/10$ *and we can verify that*

$$\lim_{\xi \to \infty} p_0 = 9/10. \tag{5.19}$$

In Fig. 5.2, Fig. 5.3 and Fig. 5.4, we show in inconsistency probability for erasure-coded partial quorum systems for $k = 2$ *and* $k = 3$ *for various cases of the write and read mean delays. Increasing* k *has benefits as minimizing the communication cost, storage costs and the latency, but that leads to a higher chances of returning inconsistent data. Hence,* k *can be chosen based on maximum probability* *of inconsistency that the application can tolerate.*

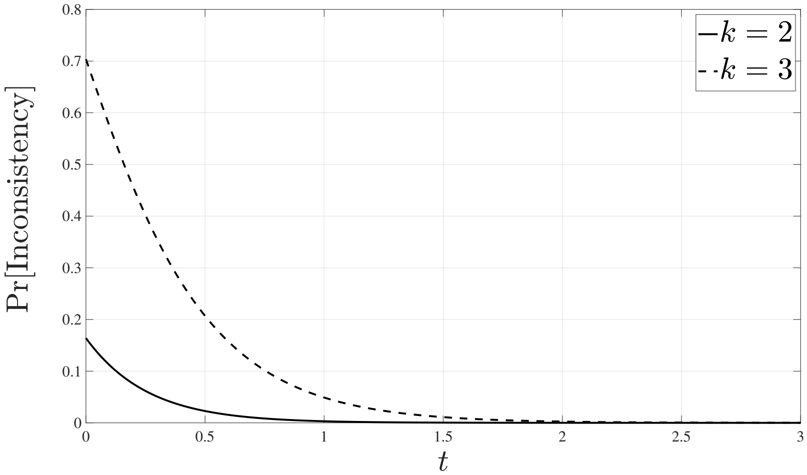


Figure 5.2. The probability of inconsistency for the case where $n = 5, c_W = 3, c_R = 3, \lambda = 1$ and $\xi = 1$.

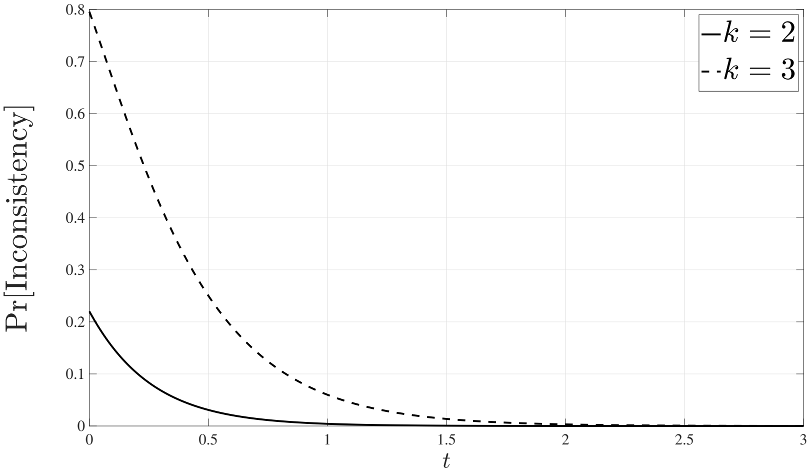


Figure 5.3. The probability of inconsistency for the case where $n = 5, c_W = 3, c_R = 3, \lambda = 1$ and $\xi = 2$.

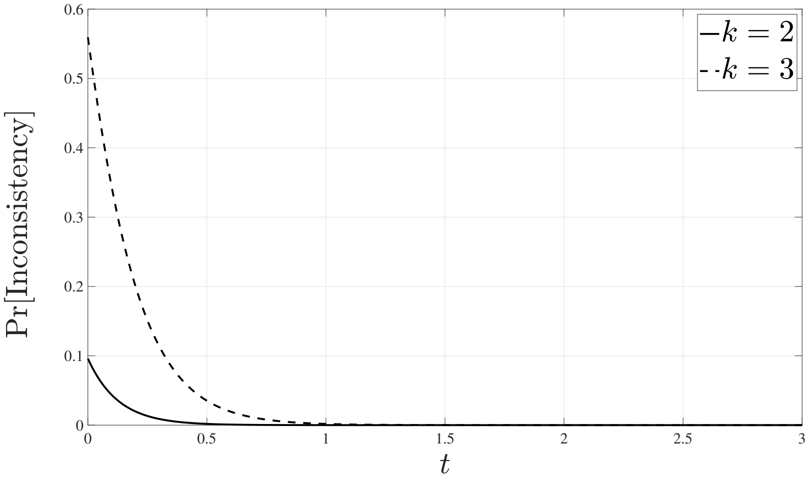


Figure 5.4. The probability of inconsistency for the case where $n = 5, c_W = 3, c_R = 3, \lambda = 2$ and $\xi = 1$.

5.5 Discussion

In this work, we studied simple erasure-coded probabilistic quorum systems, showed how to characterize the inconsistency probability of such systems and presented examples showing that erasure coding can provide more flexibility to the latency-consistency trade-off in probabilistic key-value stores as compared with replication. While our study shows how to characterize the inconsistency probability of erasure-coded probabilistic quorum systems exactly assuming exponential write and read delays, it is an open problem to derive simple yet tight bounds on the inconsistency probability in such systems for any given distribution of the write and read delays.

Chapter 6
Conclusion and Future Work

In this dissertation, we have studied the fundamental limits of strongly consistent distributed key-value stores, developed code constructions that can offer significant storage cost savings, derived lower bounds on the storage cost and also studied probabilistic consistency of eventually consistent key-value stores. In this chapter, we summarize this work and discuss several interesting future research directions.

- In Chapter 2, we have proposed code constructions that exploit the possible correlations between the data versions and minimize the storage cost in strongly consistent distributed erasure-coded key-value stores. Our study shows that correlation between versions can be exploited in consistent asynchronous decentralized systems, even if there is no single node that is aware of all data versions. Specifically, we have proposed code construction based on Reed-Solomon code for the case where the correlation coefficient δ_K is unknown a priori that is unable to harness the gains of both the erasure and delta coding simultaneously. It is an open problem to investigate whether a code construction can be developed that can harness both the erasure and delta coding gain simultaneously while not being aware of δ_K a priori.

  We have also showed that there exist linear code constructions that can harness both the erasure and delta coding gains simultaneously for the case where δ_K is known a priori through a random binning argument. Our lower bound on the storage cost shows that these constructions are within a factor of 2 from the information-theoretic optimum in certain interesting regimes. The development of practical coding schemes for the case where the correlation coefficient δ_K is known a priori that can harness both the erasure and delta coding gain simultaneously is also an interesting research direction. Extending this work also beyond the simple Hamming-based correlation model is an interesting research direction.

- In Chapter 3, we have studied the fundamental limits of strongly consistent key-value stores where a node can acquire side information of which versions propagated to some other nodes based on the underlying network topology. Our code constructions show that exchanging side information results in a better storage cost for some side information regimes at the expense of the additional latency cost of associated with exchanging the side information. Interestingly, our converse results identify topologies where a significant amount of side information does not improve the storage cost beyond the storage cost of the case where the nodes do not exchange side information.

  There are several problems that we left as open problems. While our code constructions lead to significant storage cost savings in some cases, the question of the whether these code constructions are optimal remains open especially for the case where $\nu > 2$. Developing key-value stores protocols where the nodes just exchange meta-data information of which versions they have received and not the data versions themselves is also an interesting research direction as the nodes exchange the data itself in the protocols developed in [65, 66].

The main challenge in developing such protocol is that the node may not be able to differentiate between the case where its neighbors have not received a data version and the case where the neighbors indeed have received a data version, but the meta-data has not arrived yet due to the asynchrony.

- In Chapter 4, we have studied the latency-consistency trade-off for replication-based probabilistic quorum systems, where the write and read quorums may not intersect and hence strong consistency is not guaranteed. We derived a closed-form expression for the inconsistency probability for 3-way replication-based partial quorum systems for the case where the write and read delays are exponentially distributed. Our approach allows tuning the latency and consistency guarantees in replication-based key-value stores based on the expected network delays.

  There are several interesting possible extensions for this work. First, this study can be extended to derive upper bounds on the inconsistency probability for write and read delays of any given distribution and for non-deterministic acknowledgment delays. Such bounds can be derived in terms of the first and second moments of these delays. Second, this study also can be extended to derive a closed-form expression for any replication-based key-value store in general and not necessarily the 3-way replication-based key-value stores. Finally, implementing our tuning strategy using a monitoring module that estimates the network delays and adapt the consistency guarantees accordingly as suggested in [79] is also an interesting future direction.

- In Chapter 5, we have studied the latency-consistency trade-off for simple erasure-coded probabilistic key-value stores. We showed how to characterize the inconsistency probability of such systems and presented examples showing that erasure coding can provide more flexibility to the latency-consistency trade-off as compared with replication. While our study shows how to characterize the inconsistency probability exactly assuming exponential write and read delays, it is an open problem to study the inconsistency probability in such systems for any given distribution of the write and read delays.

Appendix A
Linear Binning

In this appendix, we show that there exist *linear* codes that achieve the storage cost of Theorem 2. The proof uses linear binning instead of random binning, but mirrors the random binning proof in other respects and we only focus on the key differences here.

Lemma 7. *Let G be an $N \times M$ matrix whose entries are chosen according to Bernoulli(p) independently of each other. Let $\mathbf{u}$ be any non-zero $N \times 1$ vector. We have*

$$\mathbb{P}(\mathbf{u}^{\mathrm{T}} G = 0) = ((1 + (1-2p)^{w_H(u)})/2)^M. \tag{A.1}$$

Proof. Consider k Bernoulli trials where the probability of success of each trial is p. It can be shown that an even number of successes among the k trials occurs with probability

$$(1 + (1-2p)^k)/2.$$

Therefore, we have

$$\mathbb{P}(\mathbf{u}^{\mathrm{T}} G = 0) = ((1 + (1-2p)^{w_H(u)})/2)^M.$$

□

We now explain the code construction.

Construction 5 (Random Linear binning multi-version code)**.** *Suppose that the i-th server receives the versions $\mathbf{S}(i) = \{s_1, s_2, \cdots, s_{|\mathbf{S}(i)|}\} \subseteq [\nu]$, where $s_1 < s_2 < \cdots < s_{|\mathbf{S}(i)|}$.*

- Random code generation: *At the i-th server, for version s_j the encoder creates a random binary matrix $G_{s_j}^{(i)}$ with K rows and $(K + (\nu-1)\log Vol(\delta_K K, K) + (\nu - 1) - \log \epsilon 2^{-\nu n})/c$ columns, where each entry is chosen as Bernoulli(1/2) independently of all the other entries in the matrix and all other matrices. We denote by $G_{s_j,m}^{(i)}$ the first m columns of $G_{s_j}^{(i)}$.*
- Encoding*: The server stores $\mathbf{W}_{s_j}^{\mathrm{T}} G^{(\ell)}_{s_j,KR^{(i)}_{s_j}/c}$ for version s_j, where $R^{(i)}_{s_j}/c$ is the rate assigned by the i-th server to version s_j. The decoder is also aware of the matrix $G^{(i)}_{s_j}$ a priori. The encoding function of the i-th server is defined as follows*

$$\varphi_S^{(i)} = (\mathbf{W}_{s_1}^{\mathrm{T}} G^{(i)}_{s_1,KR^{(i)}_{s_1}/c}, \mathbf{W}_{s_2}^{\mathrm{T}} G^{(i)}_{s_2,KR^{(i)}_{s_2}/c}, \ldots, \mathbf{W}^{\mathrm{T}}_{s_{|\mathbf{S}(i)|}} G^{(i)}_{s_{|\mathbf{S}(i)|},KR^{(i)}_{s_{|\mathbf{S}(i)|}}/c}), \quad \text{(A.2)}$$

  where we choose the rates as given by (2.14), (2.15).
- Decoding*: Consider a state $\mathbf{S} \in \mathcal{P}([\nu])^n$ and assume that the decoder connects to the servers $T = \{t_1, t_2, \cdots, t_c\}$. Let $\mathbf{W}_{u_L}$ be the latest common version among these servers and that the versions $\mathbf{W}_{u_1}, \mathbf{W}_{u_2}, \cdots, \mathbf{W}_{u_{L-1}}$ are the older versions such that each is received by at least one server out of those c servers. This set of versions is denoted by $\mathbf{S}_{\mathrm{T}}$ and defined in (2.18). Given the bin indices $(b_{u_1}, b_{u_2}, \cdots, b_{u_L})$, the decoder finds all tuples $(\mathbf{w}_{u_1}, \mathbf{w}_{u_2}, \cdots, \mathbf{w}_{u_L}) \in$*

A_{δ_K} such that

$$(\varphi_{u_1}(\mathbf{w}_{u_1}) = b_{u_1}, \varphi_{u_2}(\mathbf{w}_{u_2}) = b_{u_2}, \cdots, \varphi_{u_L}(\mathbf{w}_{u_L}) = b_{u_L}).$$

If all tuples have the same latest common version $\mathbf{w}_{u_L}$, the decoder declares $\mathbf{w}_{u_L}$ to be the estimate of the latest common version $\hat{\mathbf{W}}_{\mathbf{u_L}}$. Otherwise, the decoder declares an error.

Proof of Theorem 2 using linear binning. The probability of error in decoding the latest common version among the c servers is upper-bounded as follows

$$\begin{aligned} P_e(\mathbf{S}, T) &= P(E) \\ &= P\left(\bigcup_{\mathcal{I} \subseteq \mathbf{S}_T : u_L \in \mathcal{I}} E_{\mathcal{I}}\right) \\ &\leq \sum_{\mathcal{I} \subseteq \mathbf{S}_T : u_L \in \mathcal{I}} P(E_{\mathcal{I}}), \end{aligned}$$

and we require that

$$P_e(\mathbf{S}, T) \leq \epsilon 2^{-\nu n}.$$

We proceed in a case by case manner and first consider the case where $u_{L-1} \notin \mathcal{I}$, later we consider the case where $u_{L-1} \in \mathcal{I}$. For the case where $u_{L-1} \notin \mathcal{I}$, we have the following

$$\begin{aligned} E_{\mathcal{I}} \subset \tilde{E}_{u_{L-1}} \coloneqq \{\exists \mathbf{w}'_{u_L} \neq \mathbf{W}_{u_L} : \mathbf{w}'^{\mathrm{T}}_{u_L} G^{(t_i)}_{u_L, KR^{(t_i)}_{u_L}/c} = \mathbf{W}^{\mathrm{T}}_{u_L} G^{(t_i)}_{u_L, KR^{(t_i)}_{u_L}/c}, \forall i \in [c] \\ \text{and } (\mathbf{W}_{u_{L-1}}, \mathbf{w}'_{u_L}) \in A_{\delta_K}\}. \qquad \text{(A.3)} \end{aligned}$$

Consequently, we have $P(E_{\mathcal{I}}) < P(\tilde{E}_{u_{L-1}})$, and we can upper-bound $P(\tilde{E}_{u_{L-1}})$ as follows

$$\begin{aligned}
&P(\tilde{E}_{u_{L-1}}) = \sum_{(\mathbf{w}_{u_{L-1}},\mathbf{w}_{u_L})} p(\mathbf{w}_{u_{L-1}}, \mathbf{w}_{u_L}) \\
&P\Big(\exists \mathbf{w}'_{u_L} \neq \mathbf{w}_{u_L} : \mathbf{w}'^{\mathrm{T}}_{u_L} G^{(t_i)}_{u_L, KR^{(t_i)}_{u_L}/c} = \mathbf{w}^{\mathrm{T}}_{u_L} G^{(t_i)}_{u_L, KR^{(t_i)}_{u_L}/c}, \\
&\ \forall i \in [c],\ (\mathbf{w}_{u_{L-1}}, \mathbf{w}'_{u_L}) \in A_{\delta_K}\Big) \\
&\overset{(a)}{\leq} \sum_{(\mathbf{w}_{u_{L-1}},\mathbf{w}_{u_L})} p(\mathbf{w}_{u_{L-1}}, \mathbf{w}_{u_L}) \\
&\sum_{\substack{\mathbf{w}'_{u_L} \neq \mathbf{w}_{u_L} \\ (\mathbf{w}_{u_{L-1}}, \mathbf{w}'_{u_L}) \in A_{\delta_K}}} \prod_{i=1}^{c} P\left((\mathbf{w}'_{u_L} + \mathbf{w}_{u_L})^{\mathrm{T}} G^{(t_i)}_{u_L, KR^{(t_i)}_{u_L}/c} = 0\right) \\
&\overset{(b)}{=} \sum_{(\mathbf{w}_{u_{L-1}},\mathbf{w}_{u_L})} p(\mathbf{w}_{u_{L-1}}, \mathbf{w}_{u_L}) Vol((u_L - u_{L-1})\delta_K K, K) \\
&\quad \prod_{i=1}^{c} 2^{-KR^{(t_i)}_{u_L}/c} \\
&= 2^{-(KR_{u_L} - \log Vol((u_L - u_{L-1})\delta_K K, K))}.
\end{aligned}$$

where (a) follows since the matrices $G^{(t_1)}, G^{(t_2)}, \ldots, G^{(t_c)}$ are chosen independently and (b) follows from Lemma 7. Choosing R_{u_L} to satisfy

$$KR_{u_L} \geq \log Vol((u_L - u_{L-1})\delta_K K, K) + (L-1) - \log \epsilon 2^{-\nu n}$$

ensures that

$$P(E_{\mathcal{I}}) \leq \epsilon 2^{-(L-1)} 2^{-\nu n}.$$

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

$$\begin{aligned} E_{\mathcal{I}} \subseteq \tilde{E}_{u_{L-2}} := \{ & \exists \mathbf{w}'_{u_{L-1}} \neq \mathbf{W}_{u_{L-1}}, \mathbf{w}'_{u_L} \neq \mathbf{W}_{u_L} : \\ & \mathbf{w}'^{\mathrm{T}}_{u_{L-1}} G^{(t_i)}_{u_{L-1}, KR^{(t_i)}_{u_{L-1}}/c} = \mathbf{W}^{\mathrm{T}}_{u_{L-1}} G^{(t_i)}_{u_{L-1}, KR^{(t_i)}_{u_{L-1}}/c}, \\ & \mathbf{w}'^{\mathrm{T}}_{u_L} G^{(t_i)}_{u_L, KR^{(t_i)}_{u_L}/c} = \mathbf{W}^{\mathrm{T}}_{u_L} G^{(t_i)}_{u_L, KR^{(t_i)}_{u_L}/c} \text{ and} \\ & (\mathbf{W}_{u_{L-2}}, \mathbf{w}'_{u_{L-1}}, \mathbf{w}'_{u_L}) \in A_{\delta_K} \}. \end{aligned} \tag{A.4}$$

In this case, we choose the rates to satisfy

$$K(R_{u_{L-1}} + R_{u_L}) \geq \sum_{j=L-1}^{L} \log Vol((u_j - u_{j-1})\delta_K K, K) + (L-1) - \log \epsilon 2^{-\nu n}.$$

By applying the above argument repeatedly, we obtain the region in (2.29). □

Appendix B
Proof of Theorem 6

Proof. We construct two states $\mathbf{S}_1$ and $\mathbf{S}_2$ with different decoding requirements such that the set of servers $\{l_0, l_1, \cdots, l_{c-a-1}\}$ cannot differentiate between the two states due to the limited side information. In particular, $\mathbf{S}_1$ and $\mathbf{S}_2$ are constructed such that only a servers, denoted by $\{i_0, i_1, \cdots, i_{a-1}\} \subseteq \mathcal{N} \setminus \bigcup_{i \in \{l_0, l_1, \cdots, l_{c-a-1}\}} \mathcal{H}_i$ change their states from $\mathbf{S}_1$ to $\mathbf{S}_2$.

1. **State $\mathbf{S}_2$.** In this state, $\mathbf{W}_2$ is the latest complete version and hence it must be decoded. In particular, c_W servers have both $\mathbf{W}_1$ and $\mathbf{W}_2$ and the remaining $n - c_W$ do not have any version. The set of servers that have $\mathbf{W}_{[2]}$ is given by the following disjoint union

$$\begin{aligned}\mathcal{A}_{\mathbf{S}_2}(1) = \mathcal{A}_{\mathbf{S}_2}(2) = \{l_0, l_1, \cdots, l_{c-a-1}\} &\cup \{l_{c-a}, l_{c-a+1}, \cdots, l_{c_W-a-1}\} \\ &\cup \{i_0, i_1, \cdots, i_{a-1}\},\end{aligned}$$

where $\{l_{c-a}, l_{c-a+1}, \cdots, l_{c_W-a-1}\} \subset \mathcal{N}$. In this state, the decoder connects to the following set of servers

$$\mathcal{R} = \mathcal{N} \setminus \{l_{c-a}, l_{c-a+1}, \cdots, l_{c_W-a-1}\}.$$

We denote the value stored at the i-th server, $i \in \mathcal{N}$, in this state by

$$X_i = \varphi^{(i)}_{\mathbf{S}_2(\mathcal{H}_i)}(\mathbf{W}_{\mathbf{S}_2(i)}) \in [q]. \tag{B.1}$$

Since $\mathbf{W}_2$ is the latest complete version in this state, we must have

$$H(\mathbf{W}_2 | X_{\{l_0, l_1, \cdots, l_{c-a-1}\} \cup \{i_0, i_1, \cdots, i_{a-1}\}}) = 0. \tag{B.2}$$

Thus, we have the following inequalities

$$\begin{aligned}\sum_{j \in \{l_0, \cdots, l_{c-a-1}\} \cup \{i_0, \cdots, i_{a-1}\}} H(X_j | \mathbf{W}_1) &\geq H(X_{\{l_0, l_1, \cdots, l_{c-a-1}\} \cup \{i_0, i_1, \cdots, i_{a-1}\}} | \mathbf{W}_1) \\ &= H(\mathbf{W}_2 | \mathbf{W}_1) - H(\mathbf{W}_2 | \mathbf{W}_1, X_{\{l_0, l_1, \cdots, l_{c-a-1}\} \cup \{i_0, i_1, \cdots, i_{a-1}\}}) \\ &\overset{(a)}{=} H(\mathbf{W}_2) - H(\mathbf{W}_2 | X_{\{l_0, l_1, \cdots, l_{c-a-1}\} \cup \{i_0, i_1, \cdots, i_{a-1}\}}) \\ &= K, \end{aligned} \tag{B.3}$$

where (a) follows since we assume that $\mathbf{W}_1, \mathbf{W}_2$ are independent and uniformly distributed over $[2^K]$ as the code should work for any distribution of $\mathbf{W}_{[2]}$.

2. **State $\mathbf{S}_1$.** In this state, $\mathbf{W}_1$ is the latest complete version. Therefore, the decoder can either return $\mathbf{W}_2$ or $\mathbf{W}_1$. In particular, $c_W - a$ servers have both $\mathbf{W}_1$, $\mathbf{W}_2$, a servers have only $\mathbf{W}_1$ and the remaining servers do not have any version. The set of servers that have $\mathbf{W}_1$ is given by

$$\mathcal{A}_{\mathbf{S}_1}(1) = \{l_0, \cdots, l_{c-a-1}\} \cup \{l_{c-a}, \cdots, l_{c_W-a-1}\} \cup \{i_0, \cdots, i_{a-1}\},$$

and the set of servers that have $\mathbf{W}_2$ is given by

$$\mathcal{A}_{\mathbf{S}_1}(2) = \{l_0, \cdots, l_{c-a-1}\} \cup \{l_{c-a}, \cdots, l_{c_W-a-1}\}.$$

Suppose that the decoder connects to the following set of servers

$$\mathcal{R} = \mathcal{N} \setminus \{l_{c-a}, l_{c-a+1}, \cdots, l_{c_W-a-1}\}.$$

We denote the value stored at the i-th server, $i \in \mathcal{N}$, in this state by

$$Y_i = \varphi^{(i)}_{\mathbf{S}_1(\mathcal{H}_i)}(\mathbf{W}_{\mathbf{S}_1(i)}) \in [q]. \tag{B.4}$$

Since servers $l_0, l_1, \cdots, l_{c-a-1}$ observe the same information in both states, we have

$$Y_i = X_i,\ i \in \{l_0, l_1, \cdots, l_{c-a-1}\}. \tag{B.5}$$

In this state, the decoder must either return $\mathbf{W}_2$ or $\mathbf{W}_1$. We consider these cases next.

(a) In order to decode $\mathbf{W}_2$ in this state, we must have

$$H(\mathbf{W}_2 | X_{\{l_0, l_1, \cdots, l_{c-a-1}\}}) = 0. \tag{B.6}$$

Consequently, we have the following inequities

$$(c-a)\log q \overset{(a)}{\geq} \sum_{j \in \{l_0, l_1, \cdots, l_{c-a-1}\}} H(X_j)$$

$$\begin{aligned}
&\geq \sum_{j\in\{l_0,l_1,\cdots,l_{c-a-1}\}} H(X_j|\mathbf{W}_1)\\
&\geq H(X_{\{l_0,l_1,\cdots,l_{c-a-1}\}}|\mathbf{W}_1)\\
&= H(X_{\{l_0,l_1,\cdots,l_{c-a-1}\}}|\mathbf{W}_1) - H(X_{\{l_0,l_1,\cdots,l_{c-a-1}\}}|\mathbf{W}_1,\mathbf{W}_2)\\
&= H(\mathbf{W}_2|\mathbf{W}_1) - H(\mathbf{W}_2|\mathbf{W}_1, X_{\{l_0,l_1,\cdots,l_{c-a-1}\}})\\
&= K,
\end{aligned}$$

where (a) follows since $X_i \in [q], \forall i \in \mathcal{N}$. Therefore, the storage cost is lower-bounded as follows in this case

$$\log q \geq \frac{1}{c-a}K. \tag{B.7}$$

(b) In order to decode $\mathbf{W}_1$ in this state, we must have

$$H(\mathbf{W}_1|X_{\{l_0,l_1,\cdots,l_{c-a-1}\}}, Y_{\{i_0,i_1,\cdots,i_a\}}) = 0. \tag{B.8}$$

Consequently, we have the following inequalities

$$\begin{aligned}
&\sum_{j\in\{l_0,\cdots,l_{c-a-1}\}} H(X_j|\mathbf{W}_2) + \sum_{j\in\{i_0,\cdots,i_{a-1}\}} H(Y_j|\mathbf{W}_2)\\
&\geq H(X_{\{l_0,l_1,\cdots,l_{c-a-1}\}}, Y_{\{i_0,i_1,\cdots,i_a\}}|\mathbf{W}_2)\\
&= H(X_{\{l_0,l_1,\cdots,l_{c-a-1}\}}, Y_{\{i_0,i_1,\cdots,i_a\}}|\mathbf{W}_2)\\
&- H(X_{\{l_0,l_1,\cdots,l_{c-a-1}\}}, Y_{\{i_0,i_1,\cdots,i_a\}}|\mathbf{W}_{[2]})\\
&= H(\mathbf{W}_1|\mathbf{W}_2) - H(\mathbf{W}_1|\mathbf{W}_2, X_{\{l_0,l_1,\cdots,l_{c-a-1}\}}, Y_{\{i_0,i_1,\cdots,i_a\}})\\
&= K.
\end{aligned} \tag{B.9}$$

Moreover, since $H(X_i|\mathbf{W}_{[2]}) = 0,\ i \in \mathcal{N}$ and $\mathbf{W}_1$ and $\mathbf{W}_2$ are independent, we have

$$H(X_i) = H(X_i|\mathbf{W}_1) + H(X_i|\mathbf{W}_2),$$
$$H(Y_i) = H(Y_i|\mathbf{W}_1) + H(Y_i|\mathbf{W}_2),$$

$\forall i \in \mathcal{N}$. Therefore, (B.9) can be rewritten as follows

$$\begin{aligned}\sum_{j\in\{l_0,\cdots,l_{c-a-1}\}} H(X_j) + \sum_{j\in\{i_0,\cdots,i_{a-1}\}} H(Y_j) \geq K &\\ + \sum_{j\in\{l_0,\cdots,l_{c-a-1}\}} H(X_j|\mathbf{W}_1) &\\ + \sum_{j\in\{i_0,\cdots,i_{a-1}\}} H(Y_j|\mathbf{W}_1) &\\ = K + \sum_{j\in\{l_0,\cdots,l_{c-a-1}\}} H(X_j|\mathbf{W}_1), &\end{aligned}$$

where the last equality follows since

$$H(Y_j|\mathbf{W}_1) = 0, \forall j \in \{i_0, \cdots, i_{a-1}\} \tag{B.10}$$

as those servers only have $\mathbf{W}_1$. This implies with (B.3) the following

$$\begin{aligned}c\log q &\geq \sum_{j\in\{l_0,\cdots,l_{c-a-1}\}} H(X_j) + \sum_{j\in\{i_0,\cdots,i_{a-1}\}} H(Y_j)\\ &\geq K + \sum_{j\in\{l_0,\cdots,l_{c-a-1}\}} H(X_j|\mathbf{W}_1)\\ &\geq 2K - \sum_{j\in\{i_0,\cdots,i_{a-1}\}} H(X_j|\mathbf{W}_1).\end{aligned}$$

Therefore, the storage cost in this case is lower-bounded as follows

$$\log q \geq \frac{2}{c+a}K. \tag{B.11}$$

Since in state $\mathbf{S}_1$ the decoder can decode either $\mathbf{W}_1$ or $\mathbf{W}_2$, the storage cost is lower-bounded as follows

$$\log q \geq \min\left\{\frac{K}{c-a}, \frac{2K}{c+a}\right\}. \tag{B.12}$$

□

Appendix C
Proof of Theorem 7

Proof. We consider the multi-hop network and construct two states $\mathbf{S}_2$ and $\mathbf{S}_1$. The two states have different decoding requirements, but the set of servers $\{0, 1, \cdots, n-2h-a-1\}$, where $a \in \{0, 1, \cdots, \min(n-2h, c)-1\}$, cannot differentiate between the two states. To keep the notation simple, we denote $\min(n-2h, c)$ by t.

1. **State $\mathbf{S}_2$.** In this state $\mathbf{W}_2$ is the latest complete version. Therefore, the decoder must return $\mathbf{W}_2$ in this state. The set of servers that have $\mathbf{W}_1$ is the same as the set of servers that have $\mathbf{W}_2$ and is given by the following disjoint union

$$\begin{aligned}\mathcal{A}_{\mathbf{S}_2}(1) = \mathcal{A}_{\mathbf{S}_2}(2) = \{0, 1, \cdots, t-a-1\} \cup \{l_{t-a}, l_{t-a+1}, \cdots, l_{c_W-a-1}\}\\ \cup \{n-h-a, \cdots, n-h-1\},\end{aligned}$$

   where $\{l_{t-a}, l_{t-a+1}, \cdots, l_{c_W-a-1}\} \subset \mathcal{N}$. The decoder connects to the following set of servers to decode $\mathbf{W}_2$

$$\mathcal{R} = \mathcal{N} \setminus \{l_{t-a}, l_{t-a+1}, \cdots, l_{t+c_W-c-a-1}\}.$$

The symbol stored by the i-th server in this state is denoted by

$$X_i = \varphi^{(i)}_{\mathbf{S}_2(\mathcal{H}_i)}(\mathbf{W}_{\mathbf{S}_2(i)}), \tag{C.1}$$

where $i \in \mathcal{N}$. Since $\mathbf{W}_2$ is the latest complete version in this state, we must have

$$H(\mathbf{W}_2|X_{\{0,1,\cdots,t-a-1\}\cup\{l_{t+c_W-c-a},\cdots,l_{c_W-a-1}\}\cup\{n-h-a,\cdots,n-h-1\}}) = 0. \tag{C.2}$$

Therefore, we have the following inequalities

$$\begin{aligned}
&\sum_{j\in\{0,1,\cdots,t-a-1\}\cup\{l_{t+c_W-c-a},\cdots,l_{c_W-a-1}\}\cup\{n-h-a,\cdots,n-h-1\}} H(X_j|\mathbf{W}_1)\\
&\geq H(X_{\{0,1,\cdots,t-a-1\}\cup\{l_{t+c_W-c-a},\cdots,l_{c_W-a-1}\}\cup\{n-h-a,\cdots,n-h-1\}}|\mathbf{W}_1)\\
&- H(X_{\{0,1,\cdots,t-a-1\}\cup\{l_{t+c_W-c-a},\cdots,l_{c_W-a-1}\}\cup\{n-h-a,\cdots,n-h-1\}}|\mathbf{W}_{[2]})\\
&= H(\mathbf{W}_2|\mathbf{W}_1)\\
&- H(\mathbf{W}_2|\mathbf{W}_1, X_{\{0,1,\cdots,t-a-1\}\cup\{l_{t+c_W-c-a},\cdots,l_{c_W-a-1}\}\cup\{n-h-a,\cdots,n-h-1\}})\\
&= K.
\end{aligned} \tag{C.3}$$

2. **State $\mathbf{S}_1$**. In this state $\mathbf{W}_1$ is the latest complete version. The decoder in this case must either return $\mathbf{W}_1$ or $\mathbf{W}_2$. The set of servers that have $\mathbf{W}_1$ is given by

$$\begin{aligned}
\mathcal{A}_{\mathbf{S}_1}(1) = \{0, 1, \cdots, t-a-1\} \cup \{l_{t-a}, l_{t-a+1}, \cdots, l_{c_W-a-1}\}\\
\cup \{n-h-a, \cdots, n-h-1\}.
\end{aligned}$$

The set of servers that have $\mathbf{W}_2$ is given by

$$\mathcal{A}_{\mathbf{S}_1}(2) = \{0, 1, \cdots, t-a-1\} \cup \{l_{t-a}, l_{t-a+1}, \cdots, l_{c_W-a-1}\}.$$

The decoder connects to the following set of servers to decode either $\mathbf{W}_1$ or $\mathbf{W}_2$

$$\mathcal{R} = \mathcal{N} \setminus \{l_{t-a}, l_{t-a+1}, \cdots, l_{t+c_W-c-a-1}\}.$$

The symbol stored at the i-th server in this state is denoted by

$$Y_i = \varphi^{(i)}_{\mathbf{S}_1(\mathcal{H}_i)}(\mathbf{W}_{\mathbf{S}_1(i)}), \tag{C.4}$$

where $i \in \mathcal{N}$. Since servers $0, 1, \cdots, n-2h-a$ observe the same information in both states, we have

$$Y_i = X_i,\ i \in \{0, 1, \cdots, n-2h-a\}. \tag{C.5}$$

Depending on which version the decoder will return, we consider the following cases.

(a) In order to decode $\mathbf{W}_2$ in this state, we must have

$$H(\mathbf{W}_2 | X_{\{0,1,\cdots,t-a-1\}}, Y_{\{l_{t+c_W-c-a},\cdots,l_{c_W-a-1}\}}) = 0. \tag{C.6}$$

Consequently, we have the following inequities

$$(c-a)\log q \overset{(a)}{\geq} \sum_{j\in\{0,1,\cdots,t-a-1\}} H(X_j) + \sum_{j\in\{l_{t+c_W-c-a},\cdots,l_{c_W-a-1}\}} H(Y_j)$$

$$
\begin{aligned}
&\geq \sum_{j\in\{0,1,\cdots,t-a-1\}} H(X_j|\mathbf{W}_1)+\\
&\sum_{j\in\{l_{t+c_W-c-a},\cdots,l_{c_W-a-1}\}} H(Y_j|\mathbf{W}_1)\\
&\geq H(X_{\{0,1,\cdots,t-a-1\}},Y_{\{l_{t+c_W-c-a},\cdots,l_{c_W-a-1}\}}|\mathbf{W}_1)\\
&= H(X_{\{0,1,\cdots,t-a-1\}},Y_{\{l_{t+c_W-c-a},\cdots,l_{c_W-a-1}\}}|\mathbf{W}_1)\\
&- H(X_{\{0,1,\cdots,t-a-1\}},Y_{\{l_{t+c_W-c-a},\cdots,l_{c_W-a-1}\}}|\mathbf{W}_{[2]})\\
&= H(\mathbf{W}_2|\mathbf{W}_1)-\\
&\ H(\mathbf{W}_2|\mathbf{W}_1,X_{\{0,1,\cdots,t-a-1\}},Y_{\{l_{t+c_W-c-a},\cdots,l_{c_W-a-1}\}})\\
&= K,
\end{aligned}
$$

where (a) follows since $X_i \in [q]$ and $Y_i \in [q], \forall i \in \mathcal{N}$. Therefore, decoding $\mathbf{W}_2$ in this state implies a storage cost that is lower-bounded as follows

$$\log q \geq \frac{1}{c-a}K. \tag{C.7}$$

(b) In order to decode $\mathbf{W}_1$ in this state, we must have

$$H(\mathbf{W}_1|X_{\{0,1,\cdots,t-a-1\}},Y_{\{l_{t+c_W-c-a},\cdots,l_{c_W-a-1}\}\cup\{n-h-a,\cdots,n-h-1\}})=0.$$

Consequently, we have the following inequalities

$$
\begin{aligned}
&\sum_{j\in\{0,1,\cdots,t-a-1\}} H(X_j|\mathbf{W}_2)+\sum_{j\in\{l_{t+c_W-c-a},\cdots,l_{c_W-a-1}\}} H(Y_j|\mathbf{W}_2) \qquad \text{(C.8)}\\
&+\sum_{j\in\{n-h-a,\cdots,n-h-1\}} H(Y_j|\mathbf{W}_2)\\
&\geq H(X_{\{0,1,\cdots,t-a-1\}},Y_{\{l_{t+c_W-c-a},\cdots,l_{c_W-a-1}\}},Y_{\{n-h-a,\cdots,n-h-1\}}|\mathbf{W}_2)
\end{aligned}
$$

$$\begin{aligned}
&- H(X_{\{0,1,\cdots,t-a-1\}}, Y_{\{l_{t+c_W-c-a},\cdots,l_{c_W-a-1}\}}, Y_{\{n-h-a,\cdots,n-h-1\}}|\mathbf{W}_{[2]}) \\
&= H(\mathbf{W}_1|\mathbf{W}_2) \\
&- H(\mathbf{W}_1|\mathbf{W}_2, X_{\{0,1,\cdots,t-a-1\}}, Y_{\{l_{t+c_W-c-a},\cdots,l_{c_W-a-1}\}}, Y_{\{n-h-a,\cdots,n-h-1\}}) \\
&= K. \qquad \text{(C.9)}
\end{aligned}$$

Since $\mathbf{W}_1$ and $\mathbf{W}_2$ are independent, we also have

$$\begin{aligned}
H(X_i) &= H(X_i|\mathbf{W}_1) + H(X_i|\mathbf{W}_2) \\
H(Y_i) &= H(Y_i|\mathbf{W}_1) + H(Y_i|\mathbf{W}_2), \qquad \text{(C.10)}
\end{aligned}$$

$\forall i \in \mathcal{N}$. Therefore, we have

$$\begin{aligned}
c \log q \geq& \sum_{j\in\{0,1,\cdots,t-a-1\}} H(X_j) + \sum_{j\in\{l_{t+c_W-c-a},\cdots,l_{c_W-a-1}\}} H(Y_j) \\
&+ \sum_{j\in\{n-h-a,\cdots,n-h-1\}} H(Y_j) \\
&\geq K + \sum_{j\in\{0,1,\cdots,t-a-1\}} H(X_j|\mathbf{W}_1) + \sum_{j\in\{l_{t+c_W-c-a},\cdots,l_{c_W-a-1}\}} H(Y_j|\mathbf{W}_1) \\
&+ \sum_{j\in\{n-h-a,\cdots,n-h-1\}} H(Y_j|\mathbf{W}_1) \\
&= K + \sum_{j\in\{0,1,\cdots,t-a-1\}} H(X_j|\mathbf{W}_1) + \sum_{j\in\{l_{t+c_W-c-a},\cdots,l_{c_W-a-1}\}} H(Y_j|\mathbf{W}_1),
\end{aligned}$$

where the last equality follows as $H(Y_j|\mathbf{W}_1) = 0, \forall j \in \{n-h-a, \cdots, n-h-1\}$ since those servers only have received $\mathbf{W}_1$. This implies with (C.3) the following

$$c \log q \geq 2K - \sum_{j\in\{l_{t+c_W-c-a},\cdots,l_{c_W-a-1}\}\cup\{n-h-a,\cdots,n-h-1\}} H(X_j|\mathbf{W}_1)$$

$$+ \sum_{j \in \{l_{t+c_W-c-a}, \cdots, l_{c_W-a-1}\}} H(Y_j | \mathbf{W}_1).$$

Therefore decoding $\mathbf{W}_1$ implies a storage cost that lower-bounded as follows

$$\log q \geq \frac{2}{2c - \min(c, n-2h) + a} K. \tag{C.11}$$

Since in state $\mathbf{S}_1$ the decoder can decode either $\mathbf{W}_1$ or $\mathbf{W}_2$, the storage cost is lower-bounded as follows

$$\log q \geq \min \left\{ \frac{1}{c-a}, \frac{2}{2c - \min(c, n-2h) + a} \right\} K. \tag{C.12}$$

Choosing $a = \lceil \frac{\min(n-2h, c)}{3} \rceil$, we get

$$\log q \geq \frac{2}{2c - \min(n-2h, c) + \lceil \min(n-2h, c)/3 \rceil} K. \tag{C.13}$$

□

Appendix D
Expanding Quorums: Proof of Theorem 8

For the case where $s = c_W$, we have

$$\begin{aligned}
\Pr[S(t) = c_W] &= \Pr[S(t) \le c_W] \\
&= \Pr[X_{(c_W+1)} - X_{(c_W)} > t] \\
&= e^{-\lambda_{c_W+1} t},
\end{aligned}$$

where the last equality follows Lemma 4.

For the case were $s \in \{c_W + 1, c_W + 2, \cdots, n\}$, we have

$$\begin{aligned}
\Pr[S(t) = s] &= \Pr[S(t) \le s] - \Pr[S(t) \le s - 1] \\
&= \Pr[X_{(s+1)} - X_{(c_W)} > t] - \Pr[X_{(s)} - X_{(c_W)} > t] \\
&= \Pr[X_{(s)} - X_{(c_W)} \le t] - \Pr[X_{(s+1)} - X_{(c_W)} \le t] \\
&= \Pr\left[\sum_{i=c_W+1}^{s} X_{(i)} - X_{(i-1)} \le t\right] \\
&- \Pr\left[\sum_{i=c_W+1}^{s+1} X_{(i)} - X_{(i-1)} \le t\right] \\
&= \Pr\left[\sum_{i=c_W+1}^{s} Y_i \le t\right] - \Pr\left[\sum_{i=c_W+1}^{s+1} Y_i \le t\right].
\end{aligned}$$

Since $X_1, X_2, \cdots, X_n$ are independent and identical exponential random variables, then $Y_i = X_{(i)} - X_{(i-1)}$ is an exponential random variable with parameters $\lambda_i = (n-i+1)\lambda$, where $i \in \{2, 3, \cdots, n\}$ from Lemma 4. Since $Y_1, Y_2, \cdots, Y_n$ are independent exponential random variables, from Lemma 5, we have the following

$$\begin{aligned}
\Pr[S(t) = s] &= \Pr\left[\sum_{i=c_W+1}^{s} Y_i \leq t\right] - \Pr\left[\sum_{i=c_W+1}^{s+1} Y_i \leq t\right] \\
&= \sum_{i=c_W+1}^{s} F_i(t) \prod_{\substack{j=c_W+1,\\ j\neq i}}^{s} \frac{\lambda_j}{\lambda_j - \lambda_i} - \sum_{i=c_W+1}^{s+1} F_i(t) \prod_{\substack{j=c_W+1,\\ j\neq i}}^{s+1} \frac{\lambda_j}{\lambda_j - \lambda_i} \\
&= \sum_{i=c_W+1}^{s} F_i(t) \frac{\lambda_i}{\lambda_i - \lambda_{s+1}} \prod_{\substack{j=c_W+1,\\ j\neq i}}^{s} \frac{\lambda_j}{\lambda_j - \lambda_i} - F_{s+1}(t) \prod_{j=c_W+1}^{s} \frac{\lambda_j}{\lambda_j - \lambda_{s+1}} \\
&= \sum_{i=c_W+1}^{s} F_i(t) \frac{n-i+1}{s-i+1} \prod_{\substack{j=c_W+1,\\ j\neq i}}^{s} \frac{n-j+1}{i-j} - F_{s+1}(t) \prod_{j=c_W}^{s-1} \frac{n-j}{s-j} \\
&= \sum_{i=c_W+1}^{s} F_i(t) \frac{n-i+1}{s-i+1} \prod_{\substack{j=c_W+1,\\ j\neq i}}^{s} \frac{n-j+1}{i-j} - F_{s+1}(t) \binom{n-c_W}{n-s} \\
&= \sum_{i=c_W+1}^{s} (1 - e^{-\lambda_i t}) \frac{n-i+1}{s-i+1} \prod_{\substack{j=c_W+1,\\ j\neq i}}^{s} \frac{n-j+1}{i-j} - (1 - e^{-\lambda_{s+1} t}) \binom{n-c_W}{n-s} \\
&= \sum_{i=c_W+1}^{s+1} (-1)^{s-i} (1 - e^{-\lambda_i t}) \binom{n-c_W}{n-i+1} \binom{n-i+1}{s-i+1}.
\end{aligned}$$

Appendix E
Proof of Lemma 6

Based on Lemma 5, we can express the probability density function of

$$Z_{(j)} = \sum_{l=1}^{j} Z_{(l)} - Z_{(l-1)}$$

as follows

$$\begin{aligned} f_{Z_{(j)}}(z) &= \sum_{l=1}^{j} f_l(z) \prod_{\substack{i=1,\\ i\neq l}}^{j} \frac{\xi_i}{\xi_i - \xi_l} \\ &= \sum_{l=1}^{j} (-1)^{j-l} \xi_{n-l+1} \binom{n}{j} \binom{j}{l} e^{-\xi_l z}, \end{aligned}$$

where $z \geq 0$. Therefore, using Theorem 8, we can express $\Pr[S(t + Z_{(j)}) = c_W]$ as follows

$$\begin{aligned} \Pr[S(t + Z_{(j)}) = c_W] &= \int_0^{\infty} e^{-\lambda_{c_W+1}(t+z)} f_{Z(j)}(z)\ dz \\ &= e^{-\lambda_{c_W+1} t} \\ &\sum_{l=1}^{j} \binom{n}{j} \binom{j}{l} \frac{(-1)^{j-l} \xi_{n-l+1}}{\xi_l + \lambda_{c_W+1}}. \end{aligned}$$

where $\xi_j = (n - j + 1)\xi$ and $\lambda_j = (n - j + 1)\lambda$.

Similarly for $s \in \{c_W + 1, c_W + 2, \cdots, n\}$, we have

$$\Pr[S(t + Z_{(j)}) = s] = \sum_{i=c_W+1}^{s+1} (-1)^{s-i} \binom{n - c_W}{n - i + 1} \binom{n - i + 1}{s - i + 1} \\ \left(1 - e^{-\lambda_i t} \sum_{l=1}^{j} \binom{n}{j} \binom{j}{l} \frac{(-1)^{j-l} \xi_{n-l+1}}{\xi_l + \lambda_i}\right).$$

Appendix F
Proof of Theorem 9

- The probability of inconsistency for the case where $c_W = 1$ and $c_R = 1$ can be expressed as follows

$$\begin{aligned} p_t = \Pr[r_1 \notin \mathcal{S}_1] &= \sum_{s=c_W}^{n} \Pr[r_1 \notin \mathcal{S}_1 | S(t+Z_{(1)}) = s] \Pr[S(t+Z_{(1)}) = s] \\ &= \sum_{s=c_W}^{n} \left(1 - \frac{s}{n}\right) \Pr[S(t+Z_{(1)}) = s] \\ &= \frac{2}{3} \Pr[S(t+Z_{(1)}) = 1] + \frac{1}{3} \Pr[S(t+Z_{(1)}) = 2] \\ &= \frac{2}{3} \frac{\xi_1 e^{-2\lambda t}}{2\lambda + \xi_1} + \frac{1}{3} \left(\frac{2\xi_1 e^{-\lambda t}}{\lambda + \xi_1} - \frac{2\xi e^{-2\lambda t}}{2\lambda + \xi_1} \right) \\ &= \frac{2}{3} \frac{\xi_1 e^{-\lambda t}}{\lambda + \xi_1} = \frac{2\xi e^{-\lambda t}}{\lambda + 3\xi}. \end{aligned} \tag{F.1}$$

- For the case where $c_W = 2$ and $c_R = 1$, we have

$$\begin{aligned} p_t = \frac{1}{3} \Pr[S(t+Z_{(1)}) = 2] &= \frac{1}{3} \frac{\xi_1 e^{-\lambda t}}{\lambda + \xi_1} \\ &= \frac{\xi e^{-\lambda t}}{\lambda + 3\xi}. \end{aligned} \tag{F.2}$$

- Finally, for the case where $c_R = 2$, we can express the probability inconsistency as follows

$$\begin{aligned} p_t &= \Pr\left[r_2 \notin \mathcal{S}_2, r_1 \notin \mathcal{S}_1\right] \\ &= \Pr\left[r_2 \notin \mathcal{S}_2 | r_1 \notin \mathcal{S}_1\right] \Pr\left[r_1 \notin \mathcal{S}_1\right]. \end{aligned} \tag{F.3}$$

If $r_1 \notin \mathcal{S}_1$, it may happen that $r_1 \notin \mathcal{S}_2$ as well or $r_1 \in \mathcal{S}_2$ and these two cases need to be handled separately. Therefore, we express the inconsistency probability as follows

$$\begin{aligned} p_t &= \Pr\left[r_2 \notin \mathcal{S}_2, r_1 \notin \mathcal{S}_1\right] \\ &= \Pr\left[r_2 \notin \mathcal{S}_2 | r_1 \notin \mathcal{S}_2\right] \Pr\left[r_1 \notin \mathcal{S}_2\right] + \\ &\quad \Pr\left[r_2 \notin \mathcal{S}_2 | r_1 \in \mathcal{S}_2 - \mathcal{S}_1\right] \Pr[r_1 \in \mathcal{S}_2 - \mathcal{S}_1]. \end{aligned} \tag{F.4}$$

It is important to note that

$$\Pr\left[r_2 \notin \mathcal{S}_2 | r_1 \in \mathcal{S}_2 - \mathcal{S}_1\right] = \Pr\left[r_2 \notin \mathcal{S}_2 | r_1 \in \mathcal{S}_2\right]. \tag{F.5}$$

Hence, we have

$$\begin{aligned} p_t &= \Pr\left[r_2 \notin \mathcal{S}_2 | r_1 \notin \mathcal{S}_2\right] \Pr\left[r_1 \notin \mathcal{S}_2\right] \\ &+ \Pr\left[r_2 \notin \mathcal{S}_2 | r_1 \in \mathcal{S}_2\right] (\Pr[r_1 \in \mathcal{S}_2] - \Pr[r_1 \in \mathcal{S}_1]), \end{aligned} \tag{F.6}$$

where

$$\begin{aligned} \Pr\left[r_2 \notin \mathcal{S}_2 | r_1 \notin \mathcal{S}_2\right] &= \sum_{s=c_W}^{n-1} \left(1 - \frac{s}{n-1}\right) \Pr\left[S(t + Z_{(2)}) = s\right] \\ &= \frac{1}{2} \Pr\left[S(t + Z_{(2)}) = 1\right], \end{aligned} \tag{F.7}$$

$$\begin{aligned}\Pr\left[r_2 \notin \mathcal{S}_2 | r_1 \in \mathcal{S}_2\right] &= \sum_{s=c_W}^{n} \left(1 - \frac{s-1}{n-1}\right) \Pr\left[S(t+Z_{(2)}) = s\right] \\ &= \frac{1}{2} \Pr\left[S(t+Z_{(2)}) = 2\right], \qquad \text{(F.8)}\end{aligned}$$

$$\begin{aligned}\Pr[r_1 \in \mathcal{S}_1] &= \sum_{s=c_W}^{n} \frac{s}{n} \Pr[S(t+Z_{(1)}) = s] \\ &= \frac{1}{3} \Pr[S(t+Z_{(1)}) = 1] + \\ &\quad \frac{2}{3} \Pr[S(t+Z_{(1)}) = 2] + \Pr[S(t+Z_{(1)}) = 3], \qquad \text{(F.9)}\end{aligned}$$

and

$$\begin{aligned}\Pr[r_1 \in \mathcal{S}_2] &= \sum_{s=c_W}^{n} \frac{s}{n} \Pr[S(t+Z_{(2)}) = s] \\ &= \frac{1}{3} \Pr[S(t+Z_{(2)}) = 1] \\ &+ \frac{2}{3} \Pr[S(t+Z_{(2)}) = 2] + \Pr[S(t+Z_{(2)}) = 3]. \qquad \text{(F.10)}\end{aligned}$$

Therefore, we can express the probability of inconsistency in this case as follows

$$p_t = \frac{6\xi^3 e^{-2\lambda t}}{(\lambda+2\xi)(\lambda+3\xi)} . \left(\frac{2\lambda}{(\lambda+2\xi)(\lambda+3\xi)} - \frac{(\lambda-\xi)e^{-\lambda t}}{(\lambda+\xi)(2\lambda+3\xi)}\right).$$

Bibliography

[1] DeCandia, G., D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels (2007) “Dynamo: Amazon’s highly available key-value store,” in *SOSP*, vol. 7, pp. 205–220.

[2] Sumbaly, R., J. Kreps, L. Gao, A. Feinberg, C. Soman, and S. Shah (2012) “Serving large-scale batch computed data with project voldemort,” in *Proceedings of the 10th USENIX conference on File and Storage Technologies*, USENIX Association, pp. 18–18.

[3] Zawodny, J. (2009) “Redis: lightweight key/value store that goes the extra mile Linux Magazine,” *QuarterPower Media*.

[4] Hewitt, E. (2010) *Cassandra: the definitive guide*, " O’Reilly Media, Inc.".

[5] Lynch, N. A. (1996) *Distributed Algorithms*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[6] Bailis, P., S. Venkataraman, M. J. Franklin, J. M. Hellerstein, and I. Stoica (2012) “Probabilistically bounded staleness for practical partial quorums,” *Proceedings of the VLDB Endowment*, **5**(8), pp. 776–787.

[7] Wang, Z. and V. R. Cadambe (2018) "Multi-Version Coding- An Information-Theoretic Perspective of Consistent Distributed Storage," *IEEE Transactions on Information Theory*, **64**(6), pp. 4540–4561.

[8] Dimakis, A. G., P. B. Godfrey, Y. Wu, M. J. Wainwright, and K. Ramchandran (2010) "Network coding for distributed storage systems," *IEEE Transactions on Information Theory*, **56**(9), pp. 4539–4551.

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

[10] Gifford, D. K. (1979) "Weighted voting for replicated data," in *Proceedings of the seventh ACM symposium on Operating systems principles*, pp. 150–162.

[11] Lamport, L. (1978) "Time, clocks, and the ordering of events in a distributed system," *Communications of the ACM*, **21**(7), pp. 558–565.

[12] Attiya, H., A. Bar-Noy, and D. Dolev (1995) "Sharing Memory Robustly in Message-passing Systems," *J. ACM*, **42**(1), pp. 124–142.

[13] Malkhi, D., M. K. Reiter, A. Wool, and R. N. Wright (2001) "Probabilistic quorum systems," *Information and Computation*, **170**(2), pp. 184–206.

[14] Abraham, I. and D. Malkhi (2003) "Probabilistic quorums for dynamic systems," in *International Symposium on Distributed Computing*, Springer, pp. 60–74.

[15] DeCandia, G., D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels (2007) "Dynamo: amazon's highly available key-value store," in *ACM SIGOPS operating systems review*, vol. 41, ACM, pp. 205–220.

[16] Lakshman, A. and P. Malik (2010) "Cassandra: a decentralized structured storage system," *ACM SIGOPS Operating Systems Review*, **44**(2), pp. 35–40.

[17] "A Technical Overview of Riak KV Enterprise," Available at https://riak.com.

[18] Vogels, W. (2009) "Eventually consistent," *Communications of the ACM*, **52**(1), pp. 40–44.

[19] Abadi, D. (2012) "Consistency tradeoffs in modern distributed database system design: CAP is only part of the story," *Computer*, **45**(2), pp. 37–42.

[20] Kozat, U. C. (2016), "Method and apparatus for low delay access to key-value based storage systems using FEC techniques," US Patent 9,426,517.

[21] Chen, Y. L., S. Mu, J. Li, C. Huang, J. Li, A. Ogus, and D. Phillips "Giza: Erasure Coding Objects across Global Data Centers," in *2017 USENIX Annual Technical Conference (USENIX ATC)*, Santa Clara, CA.

[22] Chen, H., H. Zhang, M. Dong, Z. Wang, Y. Xia, H. Guan, and B. Zang (2017) "Efficient and available in-memory KV-store with hybrid erasure coding and replication," *ACM Transactions on Storage (TOS)*, **13**(3), p. 25.

[23] Shankar, D., X. Lu, and D. K. Panda (2017) "High-performance and resilient key-value store with online erasure coding for big data workloads," in *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, IEEE, pp. 527–537.

[24] Li, S., Q. Zhang, Z. Yang, and Y. Dai (2017) "Bcstore: Bandwidth-efficient in-memory kv-store with batch coding," *Proc. of IEEE MSST*.

[25] Rashmi, K., M. Chowdhury, J. Kosaian, I. Stoica, and K. Ramchandran (2016) "EC-cache: load-balanced, low-latency cluster caching with online

erasure coding," in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, USENIX Association, pp. 401–417.

[26] Cadambe, V. R., Z. Wang, and N. Lynch (2016) "Information-Theoretic Lower Bounds on the Storage Cost of Shared Memory Emulation," in *Proceedings of the ninth annual ACM symposium on Principles of distributed computing*, PODC '16, ACM, pp. 305–314.

[27] Spiegelman, A., Y. Cassuto, G. Chockler, and I. Keidar (2016) "Space bounds for reliable storage: Fundamental limits of coding," in *Proceedings of the 2016 ACM Symposium on Principles of Distributed Computing*, ACM, pp. 249–258.

[28] Konwar, K. M. and V. R. Cadambe (2018) "TREAS-OPT:A Storage-Efficient Two-round Erasure-coded Algorithm for Atomic Storage," in *2nd Workshop on Storage, Control, Networking in Dynamic Systems (SCNDS)*.

[29] Zorgui, M., R. Mateescu, F. Blagojevic, C. Guyot, and Z. Wang (2018) "Storage-Efficient Shared Memory Emulation," *CoRR*, **abs/1803.01098**, 1803.01098.
URL http://arxiv.org/abs/1803.01098

[30] Cadambe, V. R., N. Lynch, M. Medard, and P. Musial (2014) "A Coded Shared Atomic Memory Algorithm for Message Passing Architectures," in *2014 IEEE 13th International Symposium on Network Computing and Applications (NCA)*, IEEE, pp. 253–260.

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

[32] Ali, R. E. and V. R. Cadambe (2019) "Harnessing correlations in distributed erasure-coded key-value stores," *IEEE Transactions on Communications*, **67**(9), pp. 5907–5920.

[33] Bailis, P., S. Venkataraman, M. J. Franklin, J. M. Hellerstein, and I. Stoica (2012) "Probabilistically bounded staleness for practical partial quorums," *Proceedings of the VLDB Endowment*, **5**(8), pp. 776–787.

[34] ——— (2014) "Quantifying eventual consistency with PBS," *The VLDB Journal*, **23**(2), pp. 279–302.

[35] El Gamal, A. and Y.-H. Kim (2011) *Network information theory*, Cambridge university press.

[36] Slepian, D. and J. K. Wolf (1973) "Noiseless coding of correlated information sources," *IEEE Transactions on Information theory*, **19**(4), pp. 471–480.

[37] Wyner, A. (1974) "Recent results in the Shannon theory," *IEEE Transactions on information Theory*, **20**(1), pp. 2–10.

[38] Pradhan, S. S. and K. Ramchandran (2003) "Distributed source coding using syndromes (DISCUS): Design and construction," *IEEE Transactions on Information Theory*, **49**(3), pp. 626–643.

[39] Schonberg, D., K. Ramchandran, and S. S. Pradhan (2004) "Distributed code constructions for the entire Slepian-Wolf rate region for arbitrarily correlated sources," in *Data Compression Conference, 2004. Proceedings. DCC 2004*, IEEE, pp. 292–301.

[40] Schonberg, D. H. (2007) *Practical distributed source coding and its application to the compression of encrypted data*, University of California, Berkeley.

[41] Stankovic, V., A. D. Liveris, Z. Xiong, and C. N. Georghiades (2004) “Design of Slepian-Wolf codes by channel code partitioning,” in *Data Compression Conference, 2004. Proceedings. DCC 2004*, IEEE, pp. 302–311.

[42] Caire, G., S. Shamai, and S. Verdu “Practical schemes for interactive data exchange,” in *IEEE International Symposium on Information Theory and its Applications (ISITA), 2004.*

[43] Wang, Q., V. Cadambe, S. Jaggi, M. Schwartz, and M. Médard (2015) “File updates under random/arbitrary insertions and deletions,” in *IEEE Information Theory Workshop (ITW)*, pp. 1–5.

[44] Ma, N., K. Ramchandran, and D. Tse (2012) “A compression algorithm using mis-aligned side-information,” in *IEEE International Symposium on Information Theory Proceedings (ISIT)*, pp. 16–20.

[45] Harshan, J., A. Datta, and F. Oggier (2015) “Compressed Differential Erasure Codes for Efficient Archival of Versioned Data,” *arXiv preprint arXiv:1503.05434*.

[46] Rouayheb, S. E., S. Goparaju, H. M. Kiah, and O. Milenkovic (2014) “Synchronizing Edits in Distributed Storage Networks,” *arXiv preprint arXiv:1409.1551*.

[47] Anthapadmanabhan, N. P., E. Soljanin, and S. Vishwanath “Update-efficient codes for erasure correction,” in *Allerton Conference on Communication, Control, and Computing, 2010*, IEEE, pp. 376–382.

[48] Mazumdar, A., V. Chandar, and G. W. Wornell (2014) "Update-efficiency and local repairability limits for capacity approaching codes," *IEEE Journal on Selected Areas in Communications*, **32**(5), pp. 976–988.

[49] Nakkiran, P., N. B. Shah, and K. Rashmi "Fundamental limits on communication for oblivious updates in storage networks," in *Global Communications Conference (GLOBECOM), 2014, IEEE*, pp. 2363–2368.

[50] Prakash, N. and M. Médard (2016) "Communication Cost for Updating Linear Functions when Message Updates are Sparse: Connections to Maximally Recoverable Codes," *arXiv preprint arXiv:1605.01105*.

[51] Hassanzadeh, P., A. Tulino, J. Llorca, and E. Erkip (2016) "Correlation-aware distributed caching and coded delivery," in *Information Theory Workshop (ITW), 2016 IEEE*, IEEE, pp. 166–170.

[52] Cover, T. M. and J. A. Thomas (2012) *Elements of information theory*, John Wiley & Sons.

[53] Wolf, J. K. (1973) "Data reduction for multiple correlated sources," in *Proc. 5th Colloquium Microwave Communication*, pp. 287–295.

[54] Ahlswede, R. and J. Korner (1975) "Source coding with side information and a converse for degraded broadcast channels," *IEEE Transactions on Information Theory*, **21**(6), pp. 629–637.

[55] Wyner, A. (1975) "On source coding with side information at the decoder," *IEEE Transactions on Information Theory*, **21**(3), pp. 294–300.

[56] Cover, T. M. (1975) “A proof of the data compression theorem of Slepian and Wolf for ergodic sources (Corresp.),” *IEEE Transactions on Information Theory*, **21**(2), pp. 226–228.

[57] Bandemer, B., A. El Gamal, and Y.-H. Kim (2012) “Simultaneous nonunique decoding is rate-optimal,” in *Communication, Control, and Computing (Allerton), 2012 50th Annual Allerton Conference on*, IEEE, pp. 9–16.

[58] Bidokhti, S. S. and V. M. Prabhakaran (2014) “Is non-unique decoding necessary?” *IEEE Transactions on Information Theory*, **60**(5), pp. 2594–2610.

[59] Langberg, M. and M. Effros (2011) “Network coding: Is zero error always possible?” in *2011 49th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, IEEE, pp. 1478–1485.

[60] Körner, J. (1973) “Coding of an information source having ambiguous alphabet and the entropy of graphs,” in *6th Prague conference on information theory*, pp. 411–425.

[61] Hendricks, J., G. R. Ganger, and M. K. Reiter (2007) “Low-overhead byzantine fault-tolerant storage,” *ACM SIGOPS Operating Systems Review*, **41**(6), pp. 73–86.

[62] Cadambe, V. R., N. Lynch, M. Medard, and P. Musial (2017) “A coded shared atomic memory algorithm for message passing architectures,” *Distributed Computing*, **30**(1), pp. 49–73.

[63] Abd-El-Malek, M., G. R. Ganger, G. R. Goodson, M. K. Reiter, and J. J. Wylie (2005) “Fault-scalable Byzantine fault-tolerant services,” in *ACM SIGOPS Operating Systems Review*, vol. 39, pp. 59–74.

[64] Androulaki, E., C. Cachin, D. Dobre, and M. Vukolić (2014) "Erasure-coded Byzantine storage with separate metadata," in *International Conference on Principles of Distributed Systems*, Springer, pp. 76–90.

[65] Cachin, C. and S. Tessaro "Optimal resilience for erasure-coded Byzantine distributed storage," in *International Conference on Dependable Systems and Networks, 2006*, pp. 115–124.

[66] Konwar, K. M., N. Prakash, E. Kantor, N. Lynch, M. Médard, and A. A. Schwarzmann (2016) "Storage-Optimized Data-Atomic Algorithms for Handling Erasures and Errors in Distributed Storage Systems," in *IEEE International Parallel and Distributed Processing Symposium*, pp. 720–729.

[67] "AWS Pricing," Available at https://aws.amazon.com/pricing/services/.

[68] "AWS Inter-Region Latency Monitoring Project," Available at https://www.cloudping.co/.

[69] "Overleaf," https://www.overleaf.com.

[70] "Amazon Simple Storage Service," https://aws.amazon.com/s3/.

[71] Brewer, E. A. (2000) "Towards robust distributed systems," in *Proceedings of the nineteenth annual ACM symposium on Principles of distributed computing*, vol. 7.

[72] Gilbert, S. and N. Lynch (2002) "Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services," *Acm Sigact News*, **33**(2), pp. 51–59.

[73] Kraska, T., M. Hentschel, G. Alonso, and D. Kossmann (2009) "Consistency rationing in the cloud: pay only when it matters," *Proceedings of the VLDB Endowment*, **2**(1), pp. 253–264.

[74] Wang, X., S. Yang, S. Wang, X. Niu, and J. Xu (2010) "An application-based adaptive replica consistency for cloud storage," in *2010 Ninth International Conference on Grid and Cloud Computing*, IEEE, pp. 13–17.

[75] Sakr, S., L. Zhao, H. Wada, and A. Liu (2011) "Clouddb autoadmin: Towards a truly elastic cloud-based data store," in *2011 IEEE International Conference on Web Services*, IEEE, pp. 732–733.

[76] Wada, H., A. Fekete, L. Zhao, K. Lee, and A. Liu (2011) "Data Consistency Properties and the Trade-offs in Commercial Cloud Storage: the Consumers' Perspective." in *CIDR*, vol. 11, pp. 134–143.

[77] Brewer, E. (2012) "CAP Twelve years Later: How the "rules" have changed," *Computer*, (2), pp. 23–29.

[78] Li, C., D. Porto, A. Clement, J. Gehrke, N. Preguiça, and R. Rodrigues (2012) "Making geo-replicated systems fast as possible, consistent when necessary," in *Presented as part of the 10th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 12)*, pp. 265–278.

[79] Chihoub, H.-E., S. Ibrahim, G. Antoniu, and M. S. Perez (2012) "Harmony: Towards automated self-adaptive consistency in cloud storage," in *2012 IEEE International Conference on Cluster Computing*, IEEE, pp. 293–301.

[80] ——— (2013) "Consistency in the cloud: When money does matter!" in *2013 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing*, IEEE, pp. 352–359.

[81] Golab, W., M. R. Rahman, A. AuYoung, K. Keeton, and I. Gupta (2014) "Client-centric benchmarking of eventual consistency for cloud storage systems," in *2014 IEEE 34th International Conference on Distributed Computing Systems*, IEEE, pp. 493–502.

[82] Liu, S., S. Nguyen, J. Ganhotra, M. R. Rahman, I. Gupta, and J. Meseguer (2015) "Quantitative analysis of consistency in NoSQL key-value stores," in *International Conference on Quantitative Evaluation of Systems*, Springer, pp. 228–243.

[83] McKenzie, M., H. Fan, and W. Golab (2015) "Fine-tuning the consistency-latency trade-off in quorum-replicated distributed storage systems," in *2015 IEEE International Conference on Big Data (Big Data)*, IEEE, pp. 1708–1717.

[84] Guerraoui, R., M. Pavlovic, and D.-A. Seredinschi (2016) "Incremental consistency guarantees for replicated objects," in *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, pp. 169–184.

[85] Chatterjee, S. and W. Golab (2017) "Brief announcement: A probabilistic performance model and tuning framework for eventually consistent distributed storage systems," in *Proceedings of the ACM Symposium on Principles of Distributed Computing*, ACM, pp. 259–261.

[86] Rahman, M. R., L. Tseng, S. Nguyen, I. Gupta, and N. Vaidya (2017) "Characterizing and adapting the consistency-latency tradeoff in distributed

key-value stores," *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, **11**(4), p. 20.

[87] Demers, A., D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, and D. Terry (1987) "Epidemic algorithms for replicated database maintenance," in *Proceedings of the sixth annual ACM Symposium on Principles of distributed computing*, pp. 1–12.

[88] Rényi, A. (1953) "On the theory of order statistics," *Acta Mathematica Hungarica*, **4**(3-4), pp. 191–231.

[89] Bibinger, M. (2013) "Notes on the sum and maximum of independent exponentially distributed random variables with different scale parameters," *arXiv preprint arXiv:1307.3945*.

[90] Ali, R. E. (2020) "Consistency Analysis of Replication-Based Probabilistic Key-Value Stores," *arXiv preprint arXiv:2002.06098*.

[91] Anderson, E., X. Li, A. Merchant, M. A. Shah, K. Smathers, J. Tucek, M. Uysal, and J. J. Wylie (2010) "Efficient eventual consistency in Pahoehoe, an erasure-coded key-blob archive," in *IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, IEEE, pp. 181–190.

Vita

Ramy E. Ali

Ramy E. Ali received the B.Sc. degree in electrical engineering from Alexandria University, Alexandria, Egypt, in 2011, and the M.Sc. degree in wireless communications from Wireless Intelligent Networks Center (WINC), Nile University, Giza, Egypt, in 2014. He was a research intern at Osram Sylvania (Siemens Business), Beverly, MA, USA in 2015. He was a research intern at the Math of Communications department, Algorithms, Analytics & Augmented Intelligence group, at Bell Labs, Crawford Hill, NJ, USA in 2017 and 2018. He was also a research intern at the IP Networking department, E2E Network & Service Automation (ENSA) group, at Bell Labs, Murray Hill, NJ, USA in 2019. He is currently a Ph.D. candidate in the Department of Electrical Engineering, Pennsylvania State University, University Park, PA, USA. He is a recipient of the Young Innovators Award (YIA) in 2011, Nile University's Graduate Fellowship in 2011, the Pennsylvania State University's Graduate Fellowship in 2015, the Newport Research Excellence Award in 2016 and the Unix 50 Presentation Award, Bell Labs in 2019. His research interests include distributed systems, information theory, coding theory and wireless communications.

Selected Publications

- R. E. Ali, V.R. Cadambe, J. Llorca and A. M. Tulino "Fundamental Limits of Erasure-Coded Key-Value Stores with Side Information", to appear in the IEEE Transactions on Communications 2020.

- R. E. Ali, V. R. Cadambe "Harnessing Correlations in Distributed Erasure-Coded Key-Value Stores ", in the IEEE Transactions on Communications, VOL. 67, NO. 9, September 2019.

- R. E. Ali, V. R. Cadambe, J. Llorca and A. M. Tulino "Multi-version Coding with Side Information", in the IEEE International Symposium on Information Theory (ISIT), Veil, Colorado, United States, June 2018.

- R. E. Ali, V. R. Cadambe "Consistent Distributed Storage of Correlated Data Updates Via Multi-version Coding", in the IEEE Information Theory Workshop (ITW), Cambridge, United Kingdom, September 2016.