Low Complexity Cache-Aided Communication Schemes for Distributed Data Storage and Distributed Computing

Thesis submitted in partial fulfillment
of the requirements for the degree of

Master of Science
in
Computer Science and Engineering
by Research

by

Vaishya Abhinav Rajeshkumar
2018121003
abhinav.vaishya@research.iiit.ac.in

International Institute of Information Technology, Hyderabad
(Deemed to be University)
Hyderabad - 500 032, INDIA
June, 2023

International Institute of Information Technology
Hyderabad, India

CERTIFICATE

It is certified that the work contained in this thesis, titled "*Low Complexity Cache-Aided Communication Schemes for Distributed Data Storage and Distributed Computing*" by *Vaishya Abhinav Rajeshkumar*, has been carried out under my supervision and is not submitted elsewhere for a degree.

Date	Advisor: Dr. Prasad Krishnan

To my loved ones.

Acknowledgments

I would like to express my sincere gratitude to my advisor, Dr. Prasad Krishnan. I could not have asked for a more supportive and understanding advisor. His guidance and perspectives have shaped me in many ways. I also thank him for understanding my weaknesses and helping me overcome them. I will always cherish the discussions that I have had with him. This thesis would not have been possible without him.

I am forever grateful to my parents and my brother, who have always encouraged me to do what I love. Their constant love, care, and support have made me what I am today.

I owe special thanks to my amazing collaborator Athreya. His motivation and dedication have been truly inspiring. I learned the value of patience through the several attempts that we made together. Without him, this thesis would not have been possible.

Finally, I would like to thank my friends at IIIT who made my life easier and enjoyable and also to all the people who have shaped me in some way and who were just as important in this journey.

Abstract

Distributed data analytics engines are designed to process and analyze large data sets in a distributed environment. The architecture of these engines is based on two key components: a distributed file system and a distributed computing framework. In this thesis, we consider the following two problems, coded data rebalancing in distributed file systems and coded distributed computing.

For the data rebalancing problem, we consider replication-based distributed storage systems in which each node stores the same quantum of data and each data bit stored has the same replication factor across the nodes. Such systems are referred to as balanced distributed databases. When existing nodes leave or new nodes are added to this system, the balanced nature of the database is lost, either due to the reduction in the replication factor, or the non-uniformity of the storage at the nodes. This triggers a rebalancing algorithm, that exchanges data between the nodes so that the balance of the database is reinstated. The goal is then to design rebalancing schemes with minimal communication load. In a recent work [1] by Krishnan et al., coded transmissions were used to rebalance a carefully designed distributed database from a node removal or addition. These coded rebalancing schemes have optimal communication load, however, require the file-size to be at least exponential in the system parameters. In this work, we consider a cyclic balanced database (where data is cyclically placed in the system nodes) and present coded rebalancing schemes for node removal and addition in such a database. These databases (and the associated rebalancing schemes) require the file-size to be only cubic in the number of nodes in the system. We bound the advantage of our node removal rebalancing scheme over the uncoded scheme, and show that our scheme has a smaller communication load. In the node addition scenario, the rebalancing scheme presented is a simple uncoded scheme, which we show has optimal load.

For the distributed computing problem, we consider the widely popular MapReduce distributed computing framework, which consists of three phases, namely Map, Shuffle, and Reduce. In previous works by Li et al. [2, 3], an optimal coded distributed computing scheme has been presented. This optimal scheme however requires very high file complexity, i.e., exponential in the number of servers K. To address this issue, low complexity distributed computing schemes using binary matrices derived from combinatorial designs have been presented in [4, 5]. In our work, we use similar principles to construct a distributed computing scheme via subspace designs, the q-analogs of combinatorial designs. While the scheme requires low file complexity, it has a higher communication load, when compared to the

optimal scheme with equivalent parameters. Also, it is primarily useful for large local storage scenarios. Moreover, we also provide numerical comparisons with some existing baseline schemes.

Contents

List of Figures

List of Tables

Notations and Terminology

$[n]$	Set $\{1, \ldots, n\}$, for $n \in \mathbb{Z}^+$
$[0]$	ϕ
$\langle n \rangle$	Set $\{0, 1, \ldots, n-1\}$, for $n \in \mathbb{Z}^+$
$i \boxplus_K j$	$\begin{cases} i+j, & \text{if } i+j \leq K \\ i+j-K, & \text{if } i+j > K \end{cases}$ (for $i, j, K \in \mathbb{Z}^+$ such that $i, j \leq K$)
$i \boxminus_K j$	$\begin{cases} i-j, & \text{if } i-j > 0 \\ i-j+K, & \text{if } i-j \leq 0 \end{cases}$ (for $i, j, K \in \mathbb{Z}^+$ such that $i, j \leq K$)
$\{i \boxplus_K A\}$	Set $\{i \boxplus_K a \ : \ a \in A\}$, for $A \subseteq [K]$ and $i \in [K]$
$X_1 \vert X_2$	Concatenation of two binary vectors, X_1 and X_2
$P(n, r)$	Permutation formula, for $n, r \in \mathbb{Z}^+$ such that $1 \leq r \leq n$
$\binom{n}{r}$	Binomial coefficient for $n, r \in \mathbb{Z}^+$ such that $1 \leq r \leq n$
$A(r, l)$	The element in the r^{th} row and l^{th} column in a matrix A
$A \backslash B$	The elements in A but not in B, for sets A, B
$A \backslash i$	$A \backslash \{i\}$, for some element i
$(j \oplus_k 1)$	$(j+1) mod\ k$, for $j \in \{0, 1, \ldots, (k-1)\}$
$\left\langle v, k \right\rangle$	Gaussian binomial coefficient, for non-negative integers $k \leq v$ and a prime power q. It is also the number of subspaces of dimension k in any v dimensional vector space over $\mathbb{F}_q$, the finite field with q elements. $\left(\left\langle v, k \right\rangle \triangleq \frac{(q^v-1)\ldots(q^{v-k+1}-1)}{(q^k-1)\ldots(q-1)} \right)$

Chapter 1

Introduction

In the field of big data processing, distributed data analytics engines are becoming increasingly popular, as they enable efficient and scalable processing of large volumes of data across a cluster of machines. They have become a powerful and an essential tool for organizations and businesses, that deal with large volumes of data. These engines consist of two key components: a distributed file system and a distributed computing framework.

A distributed file system is used to store and manage data across multiple nodes in a network. They allow multiple users to access and share files and resources in the network. They are also used to store and manage large data sets, used in data analysis. In a distributed file system, the data is spread out across multiples nodes, which helps to improve fault tolerances and reduce the risk of data loss in case of node failures. One example of such system is the Hadoop File System (HDFS) [6]. In HDFS, the data is broken up into several smaller chunks and stored across multiple nodes in a distributed fashion, with some redundancy (or replication factor). This ensures that the data is preserved when a node fails. It also provides high-throughput access to the data.

A distributed computing framework is a software framework that is used to process the data stored in a distributed file system in parallel across multiple nodes in the network, i.e., multiple nodes in the network work together to process and analyze data. It provides the necessary tools and infrastructure to perform such tasks in a distributed environment. MapReduce [7] is one of the most popular distributed computing framework. It is a programming model that allows for distributed processing of data across multiple nodes. It works by dividing the data into smaller chunks that are assigned to different nodes so that they can be processed on different nodes in the network. Each node processes its own chunk of data, and the results are combined to produce the final output.

In both the components, i.e., distributed file system and distributed computing framework, the nodes need to exchange data with each other. The reason for this varies according to the application of the system, but the general idea is that each node contains a set of data files in its local storage and demands another set of files stored in other nodes. It is clear that uncoded transmissions will result in a high communication load. Thus, a natural goal is to design coded communication schemes with communication loads as small as possible. There has been a significant amount of work related to coded transmissions

in the presence of local storage in several cache-enabled multi-receiver communication problems, such as Coded Caching [8], Coded Distributed Computing [2, 3], and Coded Data Shuffling [9, 10, 11].

In this thesis, we consider two such problems, namely Data Rebalancing in distributed file systems and Distributed Computing, which require data exchange between the nodes in the system. We describe the two problems briefly here.

- **Data Rebalancing:** In a replication-based distributed file system, the available data is split into a number of chunks and stored in the nodes in a distributed fashion with some replication factor. This ensures that the data can be recovered when a node fails. A node failure causes non-uniformity in the data distribution across the nodes, which is known as data skew. This also happens when a new node is added to the system. Data skew results in traffic imbalance in the system and also in the creation of stragglers. To prevent this, several distributed file systems use a technique known as data rebalancing. A rebalancing scheme would require moving high volumes of data across the storage nodes in order to bring the data skew below a certain threshold. This arises a necessity for an efficient data rebalancing algorithm. In [1], the authors have presented a scheme which rectifies the data skew and reinstates the replication factor, in case of a single node removal or addition.

- **Distributed Computing:** In MapReduce distributed computing framework, the overall computation is divided into three phases, namely Map, Shuffle, and Reduce phases. First, the available data is divided into several chunks and then assigned to a set of nodes in a distributed fashion. In the Map phase, the nodes generate some intermediate values (IVAs) using the Map functions on the parts of the data assigned to them. Then, the generated IVAs are exchanged among the nodes in order to compute the outputs using the Reduce functions. This data exchange happens in the Shuffle Phase. This phase involves exchanging a huge amount of data among the nodes, thus resulting in reducing the performance of a distributed computing task. In [2, 3], the authors have presented a scheme in which they show that by careful mapping of the data chunks at the computing nodes, coding opportunities arise, which help in bringing down the communication load involved in the Shuffle Phase.

In previous works [1], [3], which present the schemes with optimal communication loads for Coded Data Rebalancing and Coded Distributed Computing respectively, the file-size requirement (or file complexity) is too large and hence is a major drawback for the practical implementations of these schemes. We address this problem in both the settings and provide schemes with low file-size requirement.

1.1 Summary of Contributions

This thesis presents schemes for coded data rebalancing for distributed data storage systems with cyclic storage, with the file-size requirement cubic in the number of nodes. We also present a low complexity distributed computing scheme via subspace designs. We summarize our contributions here.

1.1.1 Coded Data Rebalancing for Distributed Data Storage Systems with Cyclic Storage

The following is an outline of our work on coded data rebalancing for distributed data storage systems with cyclic storage.

- We overcome the issue of the number of segments of a file being exponential in the number of users [1] by constructing data rebalancing schemes for cyclic databases, in which each segment of a file is placed in a consecutive set of nodes, in a wrap-around fashion. In our schemes, the file size is only cubic in the number of nodes in the system.
- We give two rebalancing schemes for single node removal scenario.
- We compare the communication loads of both the schemes with the optimal scheme and the uncoded scheme. We bound the advantage of both the schemes over the uncoded scheme, and further show that the minimum of their communication loads is always strictly less than the uncoded scheme.
- For the node addition scenario, we present a rebalancing scheme and show that it is optimal.
- We give the proof of correctness for all the schemes presented in this work.

1.1.2 A New Low Complexity Distributed Computing Scheme via Subspace Designs

The following highlights some of the aspects of our work on distributed computing via combinatorial objects known as subspace designs.

- We present a novel low complexity distributed computing scheme via subspace designs. The scheme is derived by constructing a binary computing matrix based on the properties of a subspace design.
- We provide numerical comparisons with some existing baseline schemes.

1.2 Organization of the thesis

The remainder of this thesis is organized as follows. Chapter 2 reports our work on coded data rebalancing for distributed data storage systems with cyclic storage. Chapter 3 describes our contributions towards a low complexity distributed computing scheme via subspace designs. Chapter 4 provides some concluding remarks and future directions.

Chapter 2

Coded Data Rebalancing for Distributed Data Storage Systems with Cyclic Storage

2.1 Introduction

In replication-based distributed storage systems, the available data is stored in a distributed fashion in the storage nodes with some replication factor. Doing this helps prevent data loss in case of node failures, and also provides for greater data availability and thus higher throughputs. In [1], replication-based distributed storage systems in which (A) each bit is available in the same number of nodes (i.e., the replication factor of each bit is the same) and (B) each node stores the same quantum of data, were referred to as balanced distributed databases. In such databases, when a storage node fails, or when a new node is added, the 'balanced' nature of the database is disturbed (i.e., the properties (A) or (B) do not hold anymore); this is known as *data skew*. Data skew results in various issues in the performance of such distributed databases (see Section 1 of [1]). Correcting such data skew requires some communication between the nodes of the database. In distributed systems literature, this communication phase is known as *data rebalancing* (see, for instance, [12, 13, 14, 15]). In traditional distributed storage systems, an uncoded rebalancing phase is initiated, where uncoded bits are exchanged between the nodes to recreate the balanced conditions in the new collection of nodes. Clearly, a primary goal in such a scenario would be to minimize the rebalancing or communication load (i.e., the total amount of data exchanged between the nodes) on the network during the rebalancing phase.

The rebalancing problem was formally introduced in the information theoretic setting in [1] by Krishnan et al. The idea of *coded data rebalancing* was presented in [1], based on principles similar to the landmark paper on coded caching [16]. In coded data rebalancing, coded data bits are exchanged between the nodes; the decoding of the required bits can then be done by using the prior stored data available. In [1], coded data rebalancing schemes were presented to rectify the data skew and reinstate the replication factor, in case of a single node removal or addition, for a carefully designed balanced database. The communication loads for these rebalancing schemes were characterized and shown to offer multiplicative benefits over the communication required for uncoded rebalancing. Information theoretic converse results for the communication loads were also presented in [1], proving the optimal-

ity of the achievable loads. These results were extended to the setting of *decentralized* databases in [17], where each bit of the file is placed in some random subset of the K nodes.

While Krishnan et al. [1] present an optimal scheme for the coded data rebalancing problem, the centralized database design in [1] requires that the number of segments in the data file be a large function of the number of nodes (denoted by K) in the system. In fact, as K grows, the number of file segments, and thus also the file size, have to grow exponentially in K. Thus, this scheme would warrant a high level of coordination to construct the database and perform the rebalancing. Because of these reasons, the scheme in [1] could be impractical in real-world settings. Motivated by this, in the present work, we study the rebalancing problem for *cyclic* balanced databases, in which each segment of the data file is placed in a consecutive set of nodes, in a wrap-around fashion. For such cyclic placement, the number of segments of the file could be as small as linear in K. Constructing such cyclic databases and designing rebalancing schemes for them also may not require much coordination, owing to the simplicity of the cyclic placement technique. Such cyclic storage systems have been proposed for use in distributed systems [18, 19], as well as in recent works on information theoretic approaches to private information retrieval [20] and distributed computing [21].

2.1.1 Contributions and Organization

This chapter presents coded data rebalancing schemes for distributed storage systems with cyclic storage in case of single node removal and addition. The organization and contributions of this chapter are as follows. We first review the previous work and results on coded data rebalancing for distributed storage systems in Section 2.2. Following that, we describe the system model for cyclic databases in Section 2.3. In Section 2.4, we present the main result (Theorem 2) of this work. In Section 2.5 we present a coded data rebalancing algorithm (Algorithm 1) for node removal in balanced cyclic databases. Algorithm 1 chooses between two coded transmission schemes (Scheme 1 and Scheme 2) based on the system parameters. Each of the two schemes has lower communication load than the other in certain parameter regimes determined by the values of K and the replication factor r. We bound the advantage of our schemes over the uncoded scheme, and show that the minimum of their communication loads is always strictly smaller than the uncoded rebalancing scheme, which does not permit coded transmissions. Further, the segmentation required for the scheme is only quadratic in K, and the size of the file itself is required only to be cubic in K (thus much smaller than that of [1]). In Section 2.6, we present a rebalancing scheme for addition of a single node to the cyclic database, and show that its load is optimal. Finally, in Section 2.7, we conclude this chapter with the comparisons between the schemes for single node removal in this work, the uncoded scheme, and the lower bound from [1].

2.2 Background and Previous Results

In this section, we first review the optimal scheme [1] for coded data rebalancing in distributed data storage systems, which is based on the principles similar to that of the coded caching scheme (MaN) [8] by Maddah-Ali et al. Further, we will briefly discuss the relationship between the two schemes.

2.2.1 Coded Data Rebalancing based on MaN Scheme

Consider a data file W of N bits, i.e. $W = \{W_n : n \in [N]\}, W_n \in \{0,1\}$. The system consists of K nodes which are connected to each other via a bus link. Thus, a noiseless broadcast channel exists between any pair of nodes. Each subfile $W_i, i \in [N]$ is stored in exactly r out of K nodes and each node stores same number of subfiles, i.e., $\frac{rN}{K}$ subfiles. Thus, we have an r-balanced distributed database across K nodes. We assume that the node K has failed.

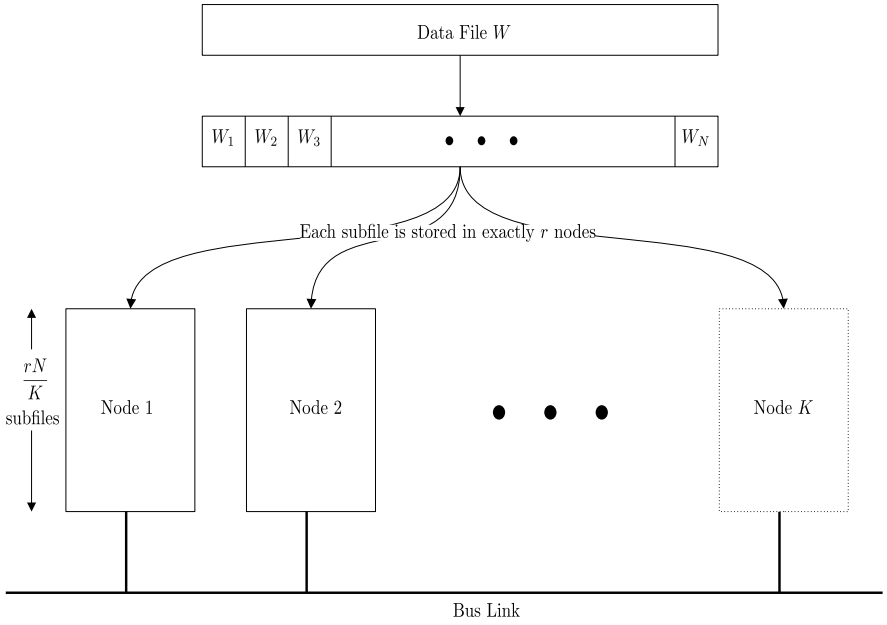

Figure 2.1: An r-balanced database on nodes $[K]$. Node K is shown dotted as it is removed from the system.

2.2.1.1 Main Result

The main result of this work is given by the following theorem (Section III of [1]).

Theorem 1 ([1]). *For balanced distributed databases on K nodes with replication factor $r \geq 2$, the following (normalized) rebalancing load L is achievable*

$$L = \frac{1}{r-1} + 1,$$

if the file size N is a multiple of $(r-1)\,P(K+1, K+1-r)$, where the symbol $P(K+1, K+1-r)$ denotes $(K+1)!\,/(r!)$. Further, the above load is optimal for a given replication factor r, i.e., $L^*(r) = \frac{1}{r-1} + 1$.

Thus, by theorem 1, we conclude that the rebalancing load in case of node removal is reduced by a factor of $r-1$, when compared to the uncoded scheme. In case of node removal, the rebalancing load is

the same as that of the uncoded scheme. For the achievability of the above theorem, the authors provide a construction of the r-balanced distributed database, for storing the subfiles across K nodes, each with a replication factor of r. Rebalancing schemes for both node removal and addition scenarios are also provided. The rebalancing scheme for node removal scenario can be illustrated using the following example.

Example: Let the number of nodes $K = 5$ and the replication factor $r = 3$. The data file W is divided into $P(5, 2) = 20$ subfiles, each indexed by $S([5], 2)$. Each node $i \in [5]$ stores all the subfiles whose indices do not contain i. Thus, each node stores $P(4, 2) = 12$ subfiles as given below.

- Node 1 stores $W_{[2\ 3]}, W_{[3\ 2]}, W_{[2\ 4]}, W_{[4\ 2]}, W_{[2\ 5]}, W_{[5\ 2]}, W_{[3\ 4]}, W_{[4\ 3]}, W_{[3\ 5]}, W_{[5\ 3]}, W_{[4\ 5]}, W_{[5\ 4]}$.
- Node 2 stores $W_{[1\ 3]}, W_{[3\ 1]}, W_{[1\ 4]}, W_{[4\ 1]}, W_{[1\ 5]}, W_{[5\ 1]}, W_{[3\ 4]}, W_{[4\ 3]}, W_{[3\ 5]}, W_{[5\ 3]}, W_{[4\ 5]}, W_{[5\ 4]}$.
- Node 3 stores $W_{[2\ 1]}, W_{[1\ 2]}, W_{[2\ 4]}, W_{[4\ 2]}, W_{[2\ 5]}, W_{[5\ 2]}, W_{[1\ 4]}, W_{[4\ 1]}, W_{[1\ 5]}, W_{[5\ 1]}, W_{[4\ 5]}, W_{[5\ 4]}$.
- Node 4 stores $W_{[2\ 3]}, W_{[3\ 2]}, W_{[2\ 1]}, W_{[1\ 2]}, W_{[2\ 5]}, W_{[5\ 2]}, W_{[3\ 1]}, W_{[1\ 3]}, W_{[3\ 5]}, W_{[5\ 3]}, W_{[1\ 5]}, W_{[5\ 1]}$.
- Node 5 stores $W_{[2\ 3]}, W_{[3\ 2]}, W_{[2\ 4]}, W_{[4\ 2]}, W_{[2\ 1]}, W_{[1\ 2]}, W_{[3\ 4]}, W_{[4\ 3]}, W_{[3\ 1]}, W_{[1\ 3]}, W_{[4\ 1]}, W_{[1\ 4]}$.

Suppose that the node 5 is removed from the system. The subfiles that were present in node 5 are partitioned into 4 groups as: $\mathcal{G}_1 = \{W_{[2\ 1]}, W_{[3\ 1]}, W_{[4\ 1]}\}, \mathcal{G}_2 = \{W_{[1\ 2]}, W_{[3\ 2]}, W_{[4\ 2]}\}, \mathcal{G}_3 = \{W_{[1\ 3]}, W_{[2\ 3]}, W_{[4\ 3]}\}$, and $\mathcal{G}_4 = \{W_{[1\ 4]}, W_{[2\ 4]}, W_{[3\ 4]}\}$. The nodes associated to a group can be obtained by considering the first element of each subfile index. For instance, the nodes associated with $\mathcal{G}_1$ are $\{2, 3, 4\}$. Similarly, the nodes associated with other groups are obtained.

Consider $W_{[2\ 1]}$ in $\mathcal{G}_1$. It is present in all the nodes associated with $\mathcal{G}_1$ except one node, which is the first element of this subfile index, i.e., node 2. Therefore, $W_{[2\ 1]}$ will be sent to node 2. Similarly, the other subfiles of $\mathcal{G}_1$, i.e., $W_{[3\ 1]}$ and $W_{[4\ 1]}$ will be sent to nodes 3 and 4 respectively. This will follow similarly for all the groups.

Now, each subfile in each group will be divided into 2 parts. For instance, $W_{[2\ 1]}$ of $\mathcal{G}_1$ will be divided into $W_{[2\ 1],3}$ and $W_{[2\ 1],4}$. Similarly, $W_{[3\ 1]}$ will be divided into $W_{[3\ 1],2}$ and $W_{[3\ 1],4}$, and $W_{[4\ 1]}$ will be divided into $W_{[4\ 1],2}$ and $W_{[4\ 1],3}$. For exchanging the subfiles within the group $\mathcal{G}_1$, node 2 will broadcast $W_{[3\ 1],2} \oplus W_{[4\ 1],2}$, node 3 will broadcast $W_{[2\ 1],3} \oplus W_{[4\ 1],3}$, and node 4 will broadcast $W_{[2\ 1],4} \oplus W_{[3\ 1],4}$. Observe that node 2 contains $W_{[4\ 1],3}$ and $W_{[3\ 1],4}$. Therefore, it can decode $W_{[2\ 1],3}$ and $W_{[2\ 1],4}$ from the transmissions broadcast by nodes 3 and 4. Similarly, nodes 3 and 4 can decode the subfiles $W_{[3\ 1]}$ and $W_{[4\ 1]}$ respectively. The transmissions for this data exchange corresponding to $\mathcal{G}_1$ are of $\frac{3}{2}^{rds}$ of the size of a subfile.

The same exchange protocol will follow for the other 4 groups. Thus, the total transmission size will be equal to 6 subfiles, which results in a communication load of $\frac{6}{12} = \frac{1}{2}$.

2.2.2 Relationship with the MaN Scheme

The achievable scheme, as discussed above, is inspired from the coded caching scheme (MaN) [8] by Maddah-Ali et al. However, the converse arguments are similar to that of [3].

For the achievability of theorem 1, the authors have presented a specific construction of the family of distributed databases. This construction uses the principles similar to those used in the placement phase of the MaN scheme. In particular, in the MaN scheme, the subfile indices indicate the set of users containing that subfile, whereas in the rebalancing scheme, the subfile indices indicate the set of nodes where that subfile is absent. This careful designing of the placement results in a symmetric placement, which enables maximal coding opportunities to serve all the nodes involved in the rebalancing process. This also ensures that the structural invariance is maintained after the process.

2.3 System Model for Cyclic Databases

Consider a binary file W consisting of a set of N equal-sized segments where the i^{th}segment is denoted by W_i, where $|W_i| = T$ bits. The system consists of K nodes indexed by $[K]$ and each node $k \in [K]$ is connected to every other node in $[K]\backslash\{k\}$ via a bus link that allows a noise-free broadcast between the nodes. A *distributed database* of W across nodes indexed by $[K]$ is a collection of subsets $\mathcal{D} = \{D_k \subseteq \{W_i : i \in [N]\} \ : \ k \in [K]\}$, such that $\bigcup_{k\in[K]} D_k = \{W_i : i \in [N]\}$, where D_k denotes the set of segments stored at node k. We will denote the set of nodes where W_i is stored as S_i. The *replication factor* of the segment W_i is then $|S_i|$. The distributed database $\mathcal{D}$ is said to be r*-balanced*, if $|S_i| = r, \forall i$, and $|D_k| = \frac{rN}{K}, \forall k$. That is, each segment is stored in r nodes, and each node stores an equal $\frac{r}{K}$-fraction of the N segments. We may assume without loss of generality that $2 \leq r \leq K-1$, since if $r = 1$ no rebalancing is possible from node removal, and if $r = K$ no rebalancing is required.

When a node k is removed from the r-balanced database, the replication factor of the segments D_k stored in node k drops by one, thus disturbing the 'balanced' state of the database. If a new empty node $K+1$ is added to the database, once again, the new database is not balanced. To reinstate the balanced state, a rebalancing scheme is initiated. Formally, a rebalancing scheme is a collection of transmissions between the nodes present, such that upon decoding the transmissions, the final database denoted by $\mathcal{D}'$ (on nodes $[K] \setminus \{k\}$ in case of node removal, or on nodes $[K+1]$ in case of node addition) is another r-balanced database.

Definition 1. *A distributed database is an r-balanced cyclic database if $N = K$ and a segment labelled W_i is stored precisely in the nodes $S_i = \{i \boxplus_K \langle r-1 \rangle\}$.*

Fig. 2.2 depicts an r-balanced cyclic balanced database on K nodes as defined above.

2.4 Main Result: Rebalancing Schemes for Cyclic Databases

As mentioned in the previous section, when a node is removed from an r-balanced database, the balanced state of the database is disturbed. To reinstate the balanced state, transmissions between the existing nodes take place. Let $X_{k'} = \phi_{k'}(D_{k'})$ be the transmission by node k' during the rebalancing phase, where $\phi_{k'}$ represents some encoding function. The communication loads of rebalancing (denoted

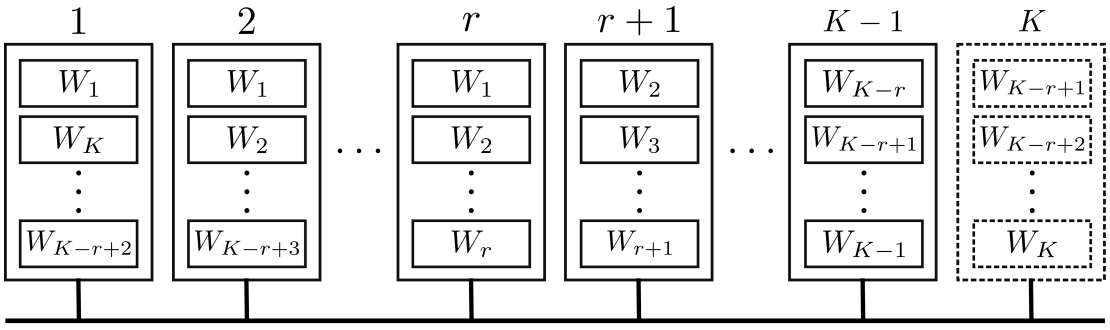

Figure 2.2: r-balanced cyclic database on nodes $[K]$

by $L_{\mathsf{rem}}(r)$ for the case of node removal, $L_{\mathsf{add}}(r)$ for node addition) are then defined as

$$L_{\mathsf{rem}}(r) = \frac{\sum_{k' \in [K] \setminus k} |X_{k'}|}{T},$$
$$L_{\mathsf{add}}(r) = \frac{\sum_{k \in [K]} |X_k|}{T}.$$

The *normalized* communication loads are then defined as $\mathfrak{L}_{rem}(r) = L_{\mathsf{rem}}(r)/N$ and $\mathfrak{L}_{add}(r) = L_{\mathsf{add}}(r)/N$. The optimal normalized communication loads for the node removal and addition scenarios are denoted by $\mathfrak{L}^*_{rem}(r)$ and $\mathfrak{L}^*_{add}(r)$ respectively. Here, the optimality is by minimization across all possible initial and target (final) databases, and all possible rebalancing schemes. In [1], it was shown[1] that $\mathfrak{L}^*_{rem}(r) \geq \frac{r}{K(r-1)}$ and $\mathfrak{L}^*_{add}(r) \geq \frac{r}{K+1}$. Further, schemes for rebalancing were presented for node removal and addition for a carefully designed database which required $N = \frac{(K+1)!}{r!}$ and $T = r - 1$, which achieve these optimal loads. Observe that, in these achievable schemes, the file size NT grows (at least) exponentially in K as K grows, for any fixed replication factor r, which is one of the main drawbacks of this result. Therefore, our interest lies in databases where N and T are small. Towards this end, we now present our main result.

Theorem 2. *For an r-balanced cyclic database having K nodes and $r \in \{3, \ldots, K-1\}$, if the segment size T is divisible by $2(K^2 - 1)$, then rebalancing schemes for node removal and addition exist which achieve the respective communication loads*

$$L_{\mathsf{rem}}(r) = \frac{K - r}{(K - 1)} + \min\left(L_1(r), L_2(r)\right),$$
$$L_{\mathsf{add}}(r) = \frac{rK}{K + 1},$$

[1]The size of the file in the present work is NT bits; whereas in [1], the notation N represents the file size in bits, thus absorbing both the segmentation and the size of each segment. The definitions of communication loads in [1] are also slightly different, involving a normalization by the storage size of the removed (or added) node. The results of [1] are presented here according to our current notations and definitions.

where, $L_1(r) = \frac{(K-r)(2r-1)}{(K-1)}$ *and* $L_2(r) = \frac{1}{2(K-1)}\left(K(r-1) + \lceil \frac{r^2-2r}{2} \rceil\right)$*. Also, the following relationship holds between* $L_{\mathsf{rem}}(r)$ *and the load* $L_u(r)$ *of the uncoded rebalancing scheme for node removal*

$$\frac{L_{\mathsf{rem}}(r)}{L_u(r)} < \min\left(2\left(1 - \frac{r-1}{K-1}\right), \left(\frac{1}{2} + \frac{1}{2r} + \frac{r}{4(K-1)}\right)\right),$$

where the RHS term is strictly smaller than 1*. Further, the rebalancing scheme for node addition is optimal (i.e.,* $L_{\mathsf{add}}(r)/N = \mathfrak{L}^*_{add}(r) = \frac{r}{K+1}$*).*

Remark 1. *In the proof of the node removal part of Theorem 2, we assume that the target database is also cyclic. For the case of* $r = 2$*, with this target database, our rebalancing scheme does not apply, as coding opportunities do not arise. Hence, we restrict our result to the scenario of* $r \in \{3, \ldots, K-1\}$*.*

We observe that the minimum file size required in the case of cyclic databases in conjunction with the above schemes is $NT = 2(K^2-1)K$, i.e., it is cubic in K and thus much smaller than the file size requirement for the schemes of [1].

2.5 Rebalancing Schemes for Single Node Removal in Cyclic Databases

In this section, we prove the result in Theorem 2 regarding the node removal scenario. We present a rebalancing algorithm, the core of which is a transmission phase in which coded subsegments are communicated between the nodes. In the transmission phase, the algorithm chooses between two schemes, Scheme 1 and Scheme 2. Scheme 1 has the communication load $\frac{K-r}{K-1} + L_1(r)$ and Scheme 2 has the load $\frac{K-r}{K-1} + L_2(r)$. We identify a threshold value for r, denoted r_{th}, beyond which Scheme 1 is found to be performing better than Scheme 2, as shown by the following claim; and thus the rebalancing algorithm chooses between the two schemes based on whether $r \geq r_{\mathsf{th}}$ or otherwise. The proof of the below claim is in Appendix A.

Claim 1. *Let* $r_{\mathsf{th}} = \lceil \frac{2K+2}{3} \rceil$*. If* $r \geq r_{\mathsf{th}}$*, then* $\min(L_1(r), L_2(r)) = L_1(r)$ *(thus, Scheme 1 has a smaller load) and if* $r < r_{\mathsf{th}}$*, we have* $\min(L_1(r), L_2(r)) = L_2(r)$ *(thus, Scheme 2 has a lower communication load).*

The organization of this section is as follows. In Subsection 2.5.1, we provide some intuition for the rebalancing algorithm, which covers both the two transmission schemes. Scheme 1 and Scheme 2. Then, in Subsection 2.5.2, we describe how the algorithm works for two example parameters (one for Scheme 1, and another Scheme 2). In Subsection 2.5.3, we formally describe the complete details of the rebalancing algorithm. In Subsection 2.5.4, we prove the correctness of two transmission schemes and the rebalancing algorithm. In Subsection 2.5.5, we calculate the communication loads of our schemes. Finally, in Subsection 2.5.6, we bound the advantage of our schemes over the uncoded scheme, and show that our schemes perform strictly better, thus completing the arguments for the node-removal part of Theorem 2.

Remark 2. *Note that throughout this section, we describe the scheme when the node K is removed from the system. A scheme for the removal of a general node can be extrapolated easily by permuting the labels of the subsegments. Further details are provided in Subsection 2.5.3 (see Remark 3).*

2.5.1 Intuition for the Rebalancing Schemes

Consider a r-balanced cyclic database as shown in Fig. 2.2. Without loss of generality, consider that the node K is removed. Now the segments that were present in node K, i.e., $D_K = \{W_{K-r+1}, \ldots, W_K\}$, no longer have replication factor r. In order to restore the replication factor of these segments, we must reinstate each bit in these segments via rebalancing into a node where it was not present before. We fix the target database post-rebalancing to also be an r-balanced cyclic database. Recall that $S_i = \{i \boxplus_K \langle r-1 \rangle\}$ represents the nodes where W_i was placed in the initial database. We represent the $K-1$ file segments in this target cyclic database as $\tilde{W}_i : i \in [K-1]$, and the nodes where $\tilde{W}_i$ would be placed is denoted as $\tilde{S}_i = \{i \boxplus_{K-1} \langle r-1 \rangle\}$. This target database is depicted in Fig. 2.3.

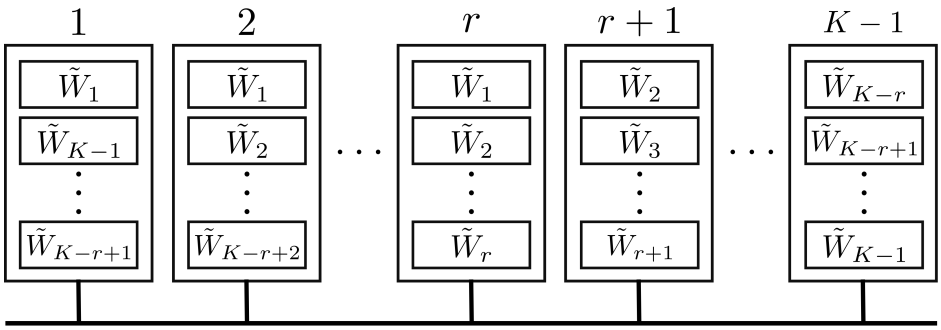

Figure 2.3: Target cyclic database on nodes $[K-1]$

Our rebalancing algorithm involves three phases: (a) a *splitting phase* where the segments in D_K are split into subsegments, (b) a *transmission phase* in which coded subsegments are transmitted, and (c) a *merge phase*, where the decoded subsegments are merged with existing segments, and appropriate deletions are carried out, to create the target database. Further, the algorithm will choose one of two transmission schemes, Scheme 1 and Scheme 2, in the transmission phase. Our discussion here pertains to both these schemes.

The design of the rebalancing algorithm is driven by two natural motives: (a) move the subsegments as minimally as possible, and (b) exploit the available coding opportunities. Based on this, we give the three generic principles below.

- *Principle 1:* The splitting and merging phases are unavoidable for maintaining the balanced nature of the target database and reducing the communication load. In our merging phase, the target segment $\tilde{W}_j : j \in [K-1]$ is constructed by merging, (a) either a subsegment of W_j or the complete W_j, along with (b) some other subsegments of the segments in D_K.

- *Principle 2:* Particularly, for each $W_i \in D_K$, we seek to split W_i into subsegments and merge these into those $\tilde{W}_j : j \in [K-1]$ such that $|\tilde{S}_j \cap S_i|$ is as large as possible, while trying to ensure the balanced condition of the target database. Observe that the maximum cardinality of such intersection is $r-1$. We denote the subsegment of segment W_i which is to be merged into $\tilde{W}_j$, and thus to be placed in the nodes $\tilde{S}_j \setminus S_i$, as $W_i^{\tilde{S}_j \setminus S_i}$. As making $|\tilde{S}_j \cap S_i|$ large reduces $|\tilde{S}_j \setminus S_i|$, we see that this principle reduces the movement of subsegments during rebalancing.

- *Principle 3:* Because of the structure of the cyclic placement, there exist 'nice' subsets of nodes whose indices are separated (cyclically) by $K-r$, which provide coding opportunity. In other words, there is a set of subsegments of segments in D_K, each of which is present in all-but-one of the nodes in any such 'nice' subset, and is to be delivered to the remaining node. Transmitting the XOR-coding of these subsegments ensures successful decoding at the respective nodes they are set to be delivered to (given by the subsegments' superscripts), because of this 'nice' structure.

We shall illustrate the third principle, which guides the design of our transmission schemes, via the examples and the algorithm itself. We now elaborate on how the first two principles are reflected in our algorithms. Consider the segment W_{K-r+1}. We call this a *corner* segment of the removed node K. Following Principles 1 and 2, this segment W_{K-r+1} will be split into $\lceil \frac{K-r+2}{2} \rceil$ subsegments, out of which one large subsegment is to be merged into $\tilde{W}_{K-r+1}$ (as $|S_{K-r+1} \cap \tilde{S}_{K-r+1}| = r-1$). In order to maintain a balanced database, the remaining $\lceil \frac{K-r+2}{2} \rceil - 1$ are to be merged into the $\lfloor \frac{K-r}{2} \rfloor$ target segments $\tilde{W}_{\lceil \frac{K-r}{2} \rceil + 1}, \ldots, \tilde{W}_{K-r}$, and additionally into the segment $\tilde{W}_{\frac{K-r+1}{2}}$ if $K-r$ is odd. The other *corner* segment of K is W_K, for which a similar splitting is followed. One large subsegment of W_K will be merged into $\tilde{W}_{K-1}$ (again, as $|S_K \cap \tilde{S}_{K-1}| = r-1$) and the remaining $\lceil \frac{K-r+2}{2} \rceil - 1$ will be merged into $\lfloor \frac{K-r}{2} \rfloor$ target segments $\tilde{W}_1, \ldots, \tilde{W}_{\lfloor \frac{K-r}{2} \rfloor}$, and additionally into the segment $\tilde{W}_{\frac{K-r+1}{2}}$ if $K-r$ is odd.

Now, consider the segment W_{K-r+2}. This is not a corner segment, hence we refer to this as a *middle* segment. This was available in the nodes $S_{K-r+2} = \{K-r+2, \ldots, K, 1\}$. Following Principles 1 and 2, this segment W_{K-r+2} will split into two: one to be merged into $\tilde{W}_{K-r+1}$ (for which $\tilde{S}_{K-r+1} = \{K-r+1, \ldots, K-1, 1\}$) and $\tilde{W}_{K-r+2}$ (for which $\tilde{S}_{K-r+2} = \{K-r+2, \ldots, K-1, 2\}$). Observe that $|S_{K-r+2} \cap \tilde{S}_{K-r+1}| = r-1$ and $|S_{K-r+2} \cap \tilde{S}_{K-r+2}| = r-1$. In the same way, each middle segment $W_{K-r+i+1} : i \in [r-2]$ is split into two subsegments, and will be merged into $\tilde{W}_{K-r+i}$ and $\tilde{W}_{K-r+i+1}$ respectively.

2.5.2 Examples

We now provide two examples illustrating our rebalancing algorithm, one corresponding to each of the two transmissions schemes.

Example illustrating Scheme 1: Consider a database with $K = 8$ nodes satisfying the r-balanced cyclic storage condition with replication factor $r = 6$. A file W is thus split into segments $W_1, \ldots, W_8$ such that the segment W_1 is stored in nodes $\{1 \boxplus_8 \langle 5 \rangle\}$, W_2 in nodes $\{2 \boxplus_8 \langle 5 \rangle\}$, W_3 in nodes $\{3 \boxplus_8 \langle 5 \rangle\}$, W_4 in nodes $\{4 \boxplus_8 \langle 5 \rangle\}$, and W_5 in nodes $\{5 \boxplus_8 \langle 5 \rangle\}$, and so on.

Node 8 is removed from the system and its contents, namely W_3, W_4, W_5, W_6, W_7, W_8, must be restored. The rebalancing algorithm performs the following steps.

Splitting: The splitting is guided by Principles 1 and 2. Each node splits the segments it contains into subsegments as follows:

- W_3 is a corner segment with respect to the removed node 8. Thus, it is split into two subsegments. The larger is labelled $W_3^{\{1\}}$ and is of size $\frac{12T}{14}$. This is to be merged into $\tilde{W}_3$ since $|S_3 \cap \tilde{S}_3| = 5 = (r-1)$. The other segment is labelled $W_3^{\{2\}}$ and is of size $\frac{2T}{14}$. As before, the idea is to merge this into $\tilde{W}_2$ to maintain a balanced database.

- The other corner segment W_8 is handled similarly. It is split into two subsegments labelled $W_8^{\{7\}}$ and $W_8^{\{6\}}$ of sizes $\frac{12T}{14}$ and $\frac{2T}{14}$ respectively.

- W_4 is a middle segment for node 8. It is split into two subsegments labelled $W_4^{\{2\}}$ and $W_4^{\{3\}}$ of sizes $\frac{10T}{14}$ and $\frac{4T}{14}$ respectively. The intent once again is to merge $W_4^{\{2\}}$ into $\tilde{W}_4$ and $W_4^{\{3\}}$ into $\tilde{W}_3$ since both $|S_4 \cap \tilde{S}_3| = |S_4 \cap \tilde{S}_4| = 5 = (r-1)$. The remaining middle segments are treated similarly.

- W_5 into two subsegments labelled $W_5^{\{3\}}$ and $W_5^{\{4\}}$ of sizes $\frac{8T}{14}$ and $\frac{6T}{14}$ respectively.

- W_6 into two subsegments labelled $W_6^{\{4\}}$ and $W_6^{\{5\}}$ of sizes $\frac{6T}{14}$ and $\frac{8T}{14}$ respectively.

- W_7 into two subsegments labelled $W_7^{\{5\}}$ and $W_7^{\{6\}}$ of sizes $\frac{4T}{14}$ and $\frac{10T}{14}$ respectively.

The superscript represents the set of nodes to which the subsegment is to be delivered.

Coding and Transmission: Now, to deliver these subsegments, nodes make use of coded broadcasts. The design of these broadcasts are guided by Principle 3. We elucidate the existence of the 'nice' subsets given in Principle 3 using a matrix form (referred to as matrix M) in Figure 2.4.

We note that this representation is similar to the combinatorial structure defined for coded caching in [22], called a placement delivery array. Consider a submatrix of M described by distinct rows $i_1, \ldots, i_l$ and distinct columns $j_1, \ldots, j_{l+1}$. If this submatrix is equivalent to the $l \times (l+1)$ matrix

$$\begin{bmatrix} s & * & \ldots & * & * \\ * & s & \ldots & * & * \\ \vdots & \vdots & \ddots & * & * \\ * & * & \ldots & s & * \end{bmatrix} \tag{2.1}$$

up to some row/column permutation, then each of the nodes $j_1, \ldots, j_l$ can decode their required subsegment from the XOR of the $i_1^{\text{th}}, \ldots, i_l^{\text{th}}$ subsegments which can be broadcasted by the $(l+1)^{\text{th}}$ node. Our rebalancing algorithm makes use of this property to design the transmissions. To denote the submatrices we make use of shapes enclosing each requirement (represented using an 's') in the matrix. For each shape, the row and column corresponding to each 's' result in a XOR-coded transmission. Before such XOR-coding, padding the 'shorter' subsegments with 0s to match the length of the longest subsegment

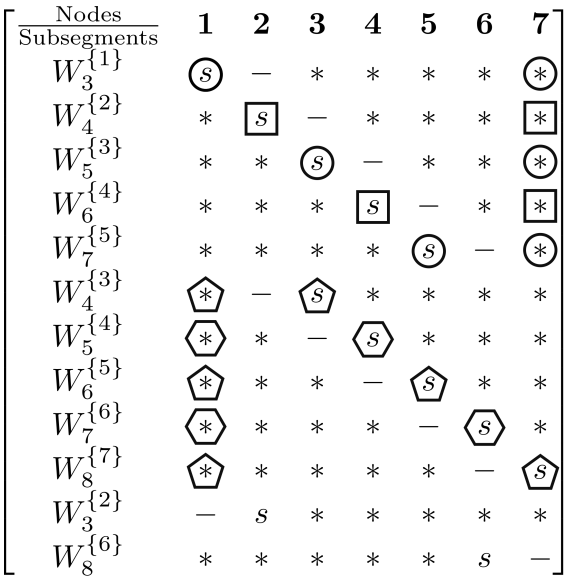

Figure 2.4: The matrix M for $K = 8$, $r = 6$. The rows correspond to subsegments and the columns correspond to nodes. Entry $M_{i,j}$ = '$*$' if the i^{th}subsegment is contained in the j^{th}node. $M_{i,j}$ = 's' if the i^{th}subsegment must be delivered to the j^{th}node. For each shape enclosing an entry, the row and column corresponding each entry with that shape gives a valid XOR-coded transmission.

would be required. There are other s entries in the matrix M which are not covered by matrices of type (2.1). These will correspond to uncoded broadcasts. Thus, we get the following transmissions from the matrix

- Node 1 pads $W_6^{\{5\}}$ and $W_4^{\{3\}}$ to size $\frac{12T}{14}$ and broadcasts $W_8^{\{7\}} \oplus W_6^{\{5\}} \oplus W_4^{\{3\}}$.
- Similarly, Node 7 pads $W_5^{\{3\}}$ and $W_7^{\{5\}}$ and broadcasts $W_3^{\{1\}} \oplus W_5^{\{3\}} \oplus W_7^{\{5\}}$.
- Node 1 pads $W_5^{\{4\}}$ to size $\frac{10T}{14}$ and broadcasts $W_5^{\{4\}} \oplus W_7^{\{6\}}$.
- Similarly, Node 7 pads $W_6^{\{4\}}$ and broadcasts $W_4^{\{2\}} \oplus W_6^{\{4\}}$.
- Finally, Node 1 broadcasts $W_8^{\{6\}}$ and Node 7 broadcasts $W_3^{\{2\}}$.

The total communication load incurred in performing these broadcasts is $\frac{1}{T}\left(2.\frac{12T}{14} + 2.\frac{10T}{14} + 2.\frac{2T}{14}\right) = \frac{24}{7}$.

Decoding: The uncoded subsegments are directly received by the respective nodes. The nodes present in the superscript of the XORed subsegment proceed to decode their respective required subsegment as follows.

- From the transmission $W_8^{\{7\}} \oplus W_6^{\{5\}} \oplus W_4^{\{3\}}$, Node 7 contains W_6, W_4 and can hence recover $W_8^{\{7\}}$ by XORing away the other subsegments. Similarly, Nodes 3 and 5 can recover $W_4^{\{3\}}$ and $W_6^{\{5\}}$ respectively.

- From the transmission $W_3^{\{1\}} \oplus W_5^{\{3\}} \oplus W_7^{\{5\}}$, Node 1 contains W_5, W_7 and can recover $W_3^{\{1\}}$. Similarly, Nodes 3 and 5 can recover $W_5^{\{3\}}$ and $W_7^{\{5\}}$ respectively.
- From the broadcast $W_5^{\{4\}} \oplus W_7^{\{6\}}$, Node 4 contains W_7 and can hence recover $W_5^{\{4\}}$. Similarly, Node 6 can recover $W_7^{\{6\}}$.
- From the broadcast $W_4^{\{2\}} \oplus W_6^{\{4\}}$. Node 2 contains W_6 and can hence recover $W_4^{\{2\}}$. Similarly, Node 4 can recover $W_6^{\{4\}}$.

Merging and Relabelling: To restore the cyclic storage condition, each node $k \in [K-1]$ merges and relabels all segments that must be stored in it in the final database. These are $\tilde{W}_j$ for $j \in \{k \boxminus_7 \langle 5 \rangle\}$.

- $\tilde{W}_1 = W_1 | W_8^{\{6\}}$ of size $1 + \frac{2T}{14} = \frac{8T}{7}$ is obtained at nodes $\{1, 2, 3, 4, 5, 6\}$.
- $\tilde{W}_2 = W_2 | W_3^{\{2\}}$ of size $1 + \frac{2T}{14} = \frac{8T}{7}$ is obtained at nodes $\{2, 3, 4, 5, 6, 7\}$.
- $\tilde{W}_3 = W_3^{\{1\}} | W_4^{\{3\}}$ of size $\frac{12T}{14} + \frac{4T}{14} = \frac{8T}{7}$ is obtained at nodes $\{3, 4, 5, 6, 7, 1\}$.
- $\tilde{W}_4 = W_4^{\{2\}} | W_5^{\{4\}}$ of size $\frac{10T}{14} + \frac{6T}{14} = \frac{8T}{7}$ is obtained at nodes $\{4, 5, 6, 7, 1, 2\}$.
- $\tilde{W}_5 = W_5^{\{3\}} | W_6^{\{5\}}$ of size $\frac{8T}{14} + \frac{8T}{14} = \frac{8T}{7}$ is obtained at nodes $\{5, 6, 7, 1, 2, 3\}$.
- $\tilde{W}_6 = W_6^{\{4\}} | W_7^{\{6\}}$ of size $\frac{6T}{14} + \frac{10T}{14} = \frac{8T}{7}$ is obtained at nodes $\{6, 7, 1, 2, 3, 4\}$.
- $\tilde{W}_7 = W_7^{\{5\}} | W_8^{\{7\}}$ of size $\frac{4T}{14} + \frac{12T}{14} = \frac{8T}{7}$ is obtained at nodes $\{7, 1, 2, 3, 4, 5\}$.

After merging and relabelling, each node keeps only the required segments mentioned previously and discards any extra data present. Since each node now stores 6 segments each of size $\frac{8T}{7}$, the total data stored is still $48T = rNT$. Thus, the cyclic storage condition is satisfied.

Example illustrating Scheme 2: Consider a database with $K = 6$ nodes satisfying the r-balanced cyclic storage condition with replication factor $r = 3$. A file W is split into segments $W_1, \ldots, W_6$ such that the segment W_1 is stored in nodes $1, 2$ and 3, W_2 in nodes $2, 3$ and 4, W_3 in nodes $3, 4$ and 5, W_4 in nodes $4, 5$ and 6, W_5 in nodes $5, 6$ and 1, and W_6 in nodes $6, 1$ and 2.

Suppose the node 6 is removed from the system. The contents of node 6, namely W_4, W_5, W_6 must be restored. To do so, the rebalancing algorithm performs the following steps.

Splitting: Again, each node that contains these segments splits them into subsegments as per Principles 1 and 2.

- W_4 is a *corner* segment for the removed node 6. This is split into three subsegments. The largest is labelled $W_4^{\{1\}}$ and is of size $\frac{7T}{10}$. This subsegment is to be merged into $\tilde{W}_4$ since $|S_4 \cap \tilde{S}_4| = 2 = (r-1)$. Observe that the superscript of $W_4^{\{1\}}$ represents the set of nodes to which the subsegment is to be delivered, i.e., $\tilde{S}_4 \setminus S_4$. The remaining 2 subsegments are labelled $W_4^{\{3\}}$ and $W_4^{\{2,3\}}$ and are of sizes $\frac{2T}{10}$, and $\frac{T}{10}$ respectively. In order to maintain a balanced database, these are to be merged into $\tilde{W}_3$ and $\tilde{W}_2$ respectively.

- Similarly, the other *corner* segment W_6 is split into three subsegments labelled $W_6^{\{5\}}, W_6^{\{3\}}$ and $W_6^{\{3,4\}}$ of sizes $\frac{7T}{10}$, $\frac{2T}{10}$, and $\frac{T}{10}$, and are to be merged with $\tilde{W}_5$, $\tilde{W}_1$, and $\tilde{W}_2$ respectively.
- W_5 is a middle segment of node 6. It is split into two subsegments labelled $W_5^{\{1\}}$ and $W_5^{\{4\}}$ of size $\frac{5T}{10}$ each. $W_5^{\{1\}}$ is to be merged into $\tilde{W}_5$, and $W_5^{\{4\}}$ into $\tilde{W}_4$, since both $|S_5 \cap \tilde{S}_5| = |S_5 \cap \tilde{S}_4| = 2 = (r-1)$.

Nodes / Subsegments	**1**	**2**	**3**	**4**	**5**
$W_4^{\{1\}}$	(s)	–	–	*	(*)
$W_5^{\{2\}}$	[*]	[s]	–	–	*
$W_5^{\{4\}}$	*	–	–	(s)	(*)
$W_6^{\{5\}}$	[*]	*	–	–	[s]
$W_4^{\{2,3\}}$	–	s	s	*	*
$W_4^{\{3\}}$	–	–	s	*	*
$W_6^{\{3\}}$	*	*	s	–	–
$W_6^{\{3,4\}}$	*	*	s	s	–

Figure 2.5: Matrix M for $K = 6, r = 3$ case. The rows of this matrix M correspond to subsegments and the columns correspond to nodes. The $M_{i,j}$ = '$*$' if the i^{th}subsegment is contained in the j^{th}node. $M_{i,j}$ = 's' if the i^{th}subsegment must be delivered to the j^{th}node. For each shape enclosing an entry, the row and column corresponding each entry with that shape lead to a XOR-coded transmission.

Coding and Transmission: As before, we make use of the placement matrix shown in Fig. 2.5 to explain how nodes perform coded broadcasts as per Principle 3. Consider the submatrix denoted by the circles in Figure 2.5. This submatrix described by columns $1, 4, 5$ and rows $1, 3$ means that each of the subsegments corresponding to these rows are present in all but one of the nodes corresponding to these columns. Further, node 5 contains both of these subsegments, and thus node 5 can broadcast the XOR of them and each of the nodes $1, 4$ can recover the respective subsegments denoted by the rows. Further, those s entries in the matrix not covered by any shape lead to uncoded broadcasts. Following these ideas, we get the following transmissions.

- Node 1 pads $W_5^{\{2\}}$ to size $\frac{7T}{10}$ and broadcasts $W_5^{\{2\}} \oplus W_6^{\{5\}}$.
- Similarly, Node 5 pads $W_5^{\{4\}}$ and broadcasts $W_4^{\{1\}} \oplus W_5^{\{4\}}$.
- Finally, Node 1 broadcasts $W_6^{\{3\}}, W_6^{\{3,4\}}$ and Node 5 broadcasts $W_4^{\{3\}}, W_4^{\{2,3\}}$.

The total communication load incurred in performing these broadcasts is $\frac{1}{T}\left(2.\frac{7T}{10} + 2.\frac{T}{10} + 2.\frac{2T}{10}\right) = 2$.

Decoding: The uncoded broadcast subsegments are received as-is by the superscript nodes. With respect to any XOR-coded subsegment, the nodes present in the superscript of the subsegment can

decode the subsegment, due to the careful design of the broadcasts as per Principle 3. For this example, we have the following.

- From the transmission $W_5^{\{2\}} \oplus W_6^{\{5\}}$, node 2 can decode $W_5^{\{2\}}$ by XORing away $W_6^{\{5\}}$ and similarly node 5 can decode $W_6^{\{5\}}$.
- Similarly, from $W_4^{\{1\}} \oplus W_5^{\{4\}}$, nodes 1 and 4 can recover $W_4^{\{1\}}$ and $W_5^{\{4\}}$ respectively.

Merging and Relabelling: To restore the cyclic storage condition, we merge and relabel the subsegments to form $\tilde{W}_1, \ldots, \tilde{W}_5$.

Each node $k \in [K-1]$ merges and relabels all segments that must be stored in it in the final database. These are $\tilde{W}_j : j \in \{k \boxminus_5 \langle 2 \rangle\}$.

- Observe that W_1 and $W_6^{\{3\}}$ are available at nodes $\{1,2,3\}$, from either the prior storage or due to decoding. Thus, the segment $\tilde{W}_1 = W_1 | W_6^{\{3\}}$ of size $1 + \frac{2T}{10} = \frac{6T}{5}$ is obtained and stored at nodes $\{1,2,3\}$. Similarly, we have the other merge operations as follows.
- $\tilde{W}_2 = W_2 | W_4^{\{2,3\}} | W_6^{\{3,4\}}$ of size $1 + \frac{T}{10} + \frac{T}{10} = \frac{6T}{5}$ is obtained at nodes $\{2,3,4\}$.
- $\tilde{W}_3 = W_3 | W_4^{\{3\}}$ of size $1 + \frac{2T}{10} = \frac{6T}{5}$ is obtained at nodes $\{3,4,5\}$.
- $\tilde{W}_4 = W_4^{\{1\}} | W_5^{\{4\}}$ of size $\frac{7T}{10} + \frac{5T}{10} = \frac{6T}{5}$ is obtained at nodes $\{4,5,1\}$.
- $\tilde{W}_5 = W_5^{\{2\}} | W_6^{\{5\}}$ of size $\frac{5T}{10} + \frac{7T}{10} = \frac{6T}{5}$ is obtained at nodes $\{5,1,2\}$.

After merging and relabelling, each node keeps only the required segments mentioned previously and discards any extra data present. Since each node now stores 3 segments each of size $\frac{6T}{5}$, the total data stored is still $18T = rNT$. Thus, the cyclic storage condition is satisfied.

2.5.3 Algorithm

In this section, following the intuition built in Subsection 2.5.1, we give our complete rebalancing algorithm (Algorithm 1) for removal of a node from an r-balanced cyclic database on K nodes (with $r \leq \{2, \ldots, K-1\}$). We thus prove the node-removal result in Theorem 2. Algorithm 1 initially invokes the SPLIT routine (described in Algorithm 2) which gives the procedure to split segments into subsegments. Each subsegment's size is assumed to be an integral multiple of $\frac{1}{2(K-1)}$. This is without loss of generality, by the condition on the size T of each segment as in Theorem 2. This splitting scheme is also illustrated in the figures Fig. 2.6-2.8. Guided by Claim 1, based on the value of r, Algorithm 1 selects between two routines that correspond to the two transmission schemes: SCHEME 1 and SCHEME 2. These schemes are given in Algorithm 3 and 4. We note that, since the sizes of the subsegments may not be the same after splitting, appropriate zero-padding (up to the size of the larger subsegment) is done before the XOR operations are performed in the two schemes. Finally, the MERGE routine (given in Algorithm 5) is run at the end of Algorithm 1. This merges the subsegments and relabels the merged segments as the target segments, thus resulting in the target r-balanced cyclic database on $K-1$ nodes.

Remark 3. *Note that, for ease of understanding, we describe the algorithms for the removal of node K from the system. The scheme for the removal of a general node i can be obtained as follows. Consider the set of permutations $\phi_i : [K] \to [K]$, where $\phi_i(j) = j \boxminus_K (K - i)$, for $i \in [K]$. If a node i is removed instead of K, we replace each label j in the subscript and superscript of the subsegments in our Algorithms 2-5 with $\phi_i(j)$. Due to the cyclic nature of both the input and target databases, we naturally obtain the rebalancing scheme for the removal of node i.*

Algorithm 1 Rebalancing Scheme for Node Removal from Cyclic Database

```
1: procedure TRANSMIT
2:     SPLIT()                                          ▷ Call SPLIT
3:     if r ≥ r_th = ⌈(2K+2)/3⌉ then
4:         SCHEME 1()                                   ▷ Call SCHEME 1
5:     else
6:         SCHEME 2()                                   ▷ Call SCHEME 2
7:     end if
8:     MERGE()                                          ▷ Call MERGE
9: end procedure
```

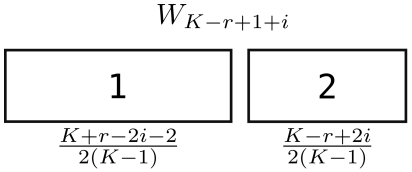

Figure 2.6: For each $i \in [r-2]$, $W_{K-r+1+i}$ is split into two parts labelled $W_{K-r+1+i}^{\{i+1\}}$ and $W_{K-r+1+i}^{\{i+K-r\}}$ of sizes $\frac{K+r-2i-2}{2(K-1)}$ and $\frac{K-r+2i}{2(K-1)}$ respectively.

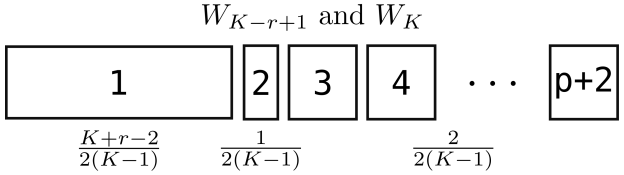

Figure 2.7: Let $p = \lfloor \frac{K-r}{2} \rfloor$. When $K - r$ is odd, W_{K-r+1} is split into $p + 2$ parts labelled $W_{K-r+1}^{\{1\}}$, $W_{K-r+1}^{\{(K-r-p)\boxplus_{K-1}\langle \min(r,p+1)\rangle\}}$, and $W_{K-r+1}^{\{(K-r+1-j)\boxplus_{K-1}\langle \min(r,j)\rangle\}}$ for $j = 1, \ldots, p$; of sizes $\frac{K+r-2}{2(K-1)}, \frac{1}{2(K-1)}, \frac{2}{2(K-1)}, \frac{2}{2(K-1)}, \ldots, \frac{2}{2(K-1)}$. Similarly, W_K is split into $p + 2$ parts labelled $W_K^{\{K-1\}}$, $W_K^{\{(r+p)\boxminus_{K-1}\langle \min(r,p+1)\rangle\}}$, and $W_K^{\{(r-1+j)\boxminus_{K-1}\langle \min(r,j)\rangle\}}$ for $j = 1, \ldots, p$; of sizes $\frac{K+r-2}{2(K-1)}, \frac{1}{2(K-1)}, \frac{2}{2(K-1)}, \frac{2}{2(K-1)}, \ldots, \frac{2}{2(K-1)}$ respectively.

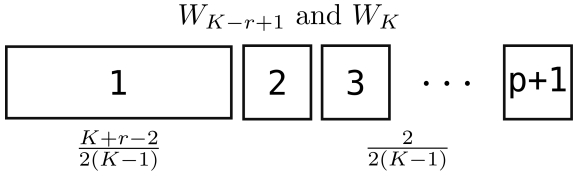

Figure 2.8: Let $p = \lfloor \frac{K-r}{2} \rfloor$. When $K-r$ is even, W_{K-r+1} is split into $p+1$ parts labelled $W_{K-r+1}^{\{1\}}$, and $W_{K-r+1}^{\{(K-r+1-j)\boxplus_{K-1}\langle \min(r,j)\rangle\}}$ for $j = 1, \ldots, p$; of sizes $\frac{K+r-2}{2(K-1)}, \frac{2}{2(K-1)}, \frac{2}{2(K-1)}, \ldots, \frac{2}{2(K-1)}$. Similarly, W_K is split into $p + 1$ parts labelled $W_K^{\{K-1\}}$, and $W_K^{\{(r-1+j)\boxminus_{K-1}\langle \min(r,j)\rangle\}}$ for $j = 1, \ldots, p$; of sizes $\frac{K+r-2}{2(K-1)}, \frac{2}{2(K-1)}, \frac{2}{2(K-1)}, \ldots, \frac{2}{2(K-1)}$ respectively.

Algorithm 2 Splitting Scheme

procedure SPLIT

  for each $i \in [r-2]$ **do**

    Split $W_{K-r+1+i}$ into subsegments labelled $W^B_{K-r+1+i}$ for $B \in \{\{i+1\}, \{i+K-r\}\}$, where the size of the subsegment is $\begin{cases} \frac{K+r-2i-2}{2(K-1)}, & \text{if } B = \{i+1\} \\ \frac{K-r+2i}{2(K-1)}, & \text{if } B = \{i+K-r\} \end{cases}$

  end for

  if $K-r$ is odd **then**

    Let $p = \lfloor \frac{K-r}{2} \rfloor$

    Split W_{K-r+1} into $p+2$ subsegments labelled W^B_{K-r+1}, for $B \in \{\{1\}, \{(K-r-p) \boxplus_{K-1} \langle \min(r, p+1) \rangle\}\} \cup \{\{(K-r+1-j) \boxplus_{K-1} \langle \min(r,j) \rangle\} : j \in [p]\}$ where the size of the subsegment is $\begin{cases} \frac{K+r-2}{2(K-1)}, & \text{if } B = \{1\} \\ \frac{1}{2(K-1)}, & \text{if } B = \{(K-r-p) \boxplus_{K-1} \langle \min(r, p+1) \rangle\} \\ \frac{2}{2(K-1)}, & \text{otherwise} \end{cases}$

    Split W_K into $p+2$ subsegments labelled W^B_K, for $B \in \{\{K-1\}, \{(r+p) \boxminus_{K-1} \langle \min(r, p+1) \rangle\}\} \cup \{\{(r-1+j) \boxminus_{K-1} \langle \min(r,j) \rangle\} : j \in [p]\}$ where the size of the subsegment is $\begin{cases} \frac{K+r-2}{2(K-1)}, & \text{if } B = \{K-1\} \\ \frac{1}{2(K-1)}, & \text{if } B = \{(r+p) \boxminus_{K-1} \langle \min(r, p+1) \rangle\} \\ \frac{2}{2(K-1)}, & \text{otherwise} \end{cases}$

  else

    Let $p = \frac{K-r}{2}$

    Split W_{K-r+1} into $p+1$ subsegments labelled W^B_{K-r+1}, for $B \in \{1\} \cup \{\{(K-r+1-j) \boxplus_{K-1} \langle \min(r,j) \rangle\} : j \in [p]\}$ where the size of the subsegment is $\begin{cases} \frac{K+r-2}{2(K-1)}, & \text{if } B = \{1\} \\ \frac{2}{2(K-1)}, & \text{otherwise} \end{cases}$

    Split W_K into $p+1$ subsegments labelled W^B_K, for $B \in \{K-1\} \cup \{\{(r-1+j) \boxminus_{K-1} \langle \min(r,j) \rangle\} : j \in [p]\}$ where the size of the subsegment is $\begin{cases} \frac{K+r-2}{2(K-1)}, & \text{if } B = \{K-1\} \\ \frac{2}{2(K-1)}, & \text{otherwise} \end{cases}$

  end if

end procedure

Algorithm 3 Transmission Scheme 1

procedure SCHEME 1

  for each $i = 1, \ldots, K - r$ **do**

    Node 1 broadcasts $\bigoplus_{j=0}^{\lfloor \frac{r-1-i}{K-r} \rfloor} W_{K+1-i-j(K-r)}^{\{K-i-j(K-r)\}}$.

    Node $K-1$ broadcasts $\bigoplus_{j=0}^{\lfloor \frac{r-1-i}{K-r} \rfloor} W_{K-r+i+j(K-r)}^{\{i+j(K-r)\}}$

  end for

  Node 1 broadcasts all subsegments of W_K except $W_K^{\{K-1\}}$.

  Node $K-1$ broadcasts all subsegments of W_{K-r+1} except $W_{K-r+1}^{\{1\}}$.

end procedure

Algorithm 4 Transmission Scheme 2

procedure SCHEME 2

  for each $i = 2, \ldots, r - 1$ **do**

    Node 1 broadcasts $W_{K-r+i}^{\{i\}} \oplus W_{K-r+i+1}^{\{K-r+i\}}$.

  end for

  Node $K-1$ broadcasts $W_{K-r+1}^{\{1\}} \oplus W_{K-r+2}^{\{K-r+1\}}$.

  Node 1 broadcasts all subsegments of W_K except $W_K^{\{K-1\}}$.

  Node $K-1$ broadcasts all subsegments of W_{K-r+1} except $W_{K-r+1}^{\{1\}}$.

end procedure

Algorithm 5 Merging and Relabelling

procedure MERGE

  for each $i = 1, \ldots, r-1$ **do**

    Each node in $\{(K-r+i) \boxplus_{K-1} \langle r \rangle\}$ performs the concatenation $\tilde{W}_{K-r+i} = W_{K-r+i}^{\{i\}} | W_{K-r+i+1}^{\{K-r+i\}}$.

  end for

  if $K-r$ is even **then**

    for each $i = 1, \ldots, \frac{K-r}{2}$ **do**

      Each node in $\{i \boxplus_{K-1} \langle r \rangle\}$ performs the concatenation $\tilde{W}_i = W_i | W_K^{\{(r-1+i) \boxminus_{K-1} \langle min(r,i) \rangle\}}$.

    end for

    for each $i = \frac{K-r}{2} + 1, \ldots, K-r$ **do**

      Each node in $\{i \boxplus_{K-1} \langle r \rangle\}$ performs the concatenation $\tilde{W}_i = W_i | W_{K-r+1}^{\{i \boxplus_{K-1} \langle min(r,K-r-i+1) \rangle\}}$.

    end for

  else

    for each $i = 1, \ldots, \frac{K-r-1}{2}$ **do**

      Each node in $\{i \boxplus_{K-1} \langle r \rangle\}$ performs the concatenation $\tilde{W}_i = W_i | W_K^{\{(r-1+i) \boxminus_{K-1} \langle min(r,i) \rangle\}}$.

    end for

    for each $i = \frac{K-r+1}{2} + 1, \ldots, K-r$ **do**

      Each node in $\{i \boxplus_{K-1} \langle r \rangle\}$ performs the concatenation $\tilde{W}_i = W_i | W_{K-r+1}^{\{i \boxplus_{K-1} \langle min(r,K-r-i+1) \rangle\}}$.

    end for

    Each node in $\left\{\left(\frac{K-r+1}{2}\right) \boxplus_{K-1} \langle r \rangle\right\}$ performs the concatenation $\tilde{W}_{\frac{K-r+1}{2}} = W_{\frac{K-r+1}{2}} | W_K^{\{(r+p) \boxminus_{K-1} \langle \min(r,p+1) \rangle\}} | W_{K-r+1}^{\{(K-r-p) \boxplus_{K-1} \langle \min(r,p+1) \rangle\}}$, where $p = \lfloor \frac{K-r}{2} \rfloor$.

  end if

end procedure

Note: Once the target segments $\tilde{W}_1, \ldots, \tilde{W}_{K-1}$ are recovered at the required nodes, any extra bits present at the node are discarded.

2.5.4 Correctness

Before we proceed, we recall the following notations. For a non-negative integer n, we use $[n]$ to denote the set $\{1, \ldots, n\}$. We also define $[0] \triangleq \phi$. Similarly, for a positive integer n, $\langle n \rangle$ denotes the set $\{0, 1, \ldots, n-1\}$. To describe operations involving wrap-arounds, we define two operators. For positive integers i, j, K such that $i, j \leq K$, $i \boxplus_K j = \begin{cases} i+j, & \text{if } i+j \leq K \\ i+j-K, & \text{if } i+j > K \end{cases}$. Similarly, $i \boxminus_K j = \begin{cases} i-j, & \text{if } i-j > 0 \\ i-j+K, & \text{if } i-j \leq 0 \end{cases}$. We also extend these operations to sets. For $A \subseteq [K]$ and $i \in [K]$, we use $\{i \boxplus_K A\}$ to denote the set $\{i \boxplus_K a \ : \ a \in A\}$. Similarly, $\{i \boxminus_K A\}$ denotes $\{i \boxminus_K a \ : \ a \in A\}$. For a binary vector X, we use $|X|$ to denote its size. The concatenation of two binary vectors, X_1 and X_2, is denoted by $X_1|X_2$. We also recall that, for a segment W_i and its subsegment W_i^J, J represents the set of nodes where this subsegment needs to be sent.

To check the correctness of the scheme, we have to check the correctness of the encoding, decoding, and the merging. It is straightforward to check that the nodes that broadcast any transmission, whether coded or uncoded subsegments, contain all respective subsegments according to the design of the initial storage. Thus, the XOR-coding and broadcasts given in the transmissions schemes are correct. For checking the decoding, we must check that each subsegment can be decoded at the corresponding 'superscript' nodes where it is meant to be delivered. We must also check that the merging scheme is successful, i.e., at any node, all the subsegments to be merged into a target segment are available at that node. Finally, we check that the target database is the cyclic database on $K-1$ nodes.

Now, we focus on checking the decoding of the transmissions in both Scheme 1 and Scheme 2. Clearly, all uncoded transmissions are directly received. Thus, we now check only the decoding involved for XOR-coded transmissions, for the two schemes.

- **Decoding for Scheme 1:** For each $i \in [K-r]$, two broadcasts $\bigoplus_{j=0}^{\lfloor \frac{r-1-i}{K-r} \rfloor} W_{K+1-i-j(K-r)}^{\{K-i-j(K-r)\}}$ and $\bigoplus_{j=0}^{\lfloor \frac{r-1-i}{K-r} \rfloor} W_{K-r+i+j(K-r)}^{\{i+j(K-r)\}}$ are made. Consider the first broadcast. Let $J = \{0, \ldots, \lfloor \frac{r-1-i}{K-r} \rfloor\}$. For some $j \in J$, consider the segment $W_{K+1-i-j(K-r)}$. For any $j' \in J \backslash \{j\}$, we claim that node $K-i-j'(K-r)$ contains the segment $W_{K+1-i-j(K-r)}$. Going through all possible j, j' would then mean that all the segments in this first XOR-coded broadcast can be decoded at the respective superscript-nodes. Now, for node $K-i-j'(K-r)$ to contain the segment $W_{K+1-i-j(K-r)}$, the following condition must be satisfied:

  - Condition (A): $K-i-j'(K-r) \in S_{K+1-i-j(K-r)} = \{(K+1-i-j(K-r)) \boxplus_K \langle r \rangle\}$.

  To remove the wrap-around, we simplify Condition (A) into two cases based on the relation between j and j'. For Condition (A) to hold, it is easy to check that one of the following pairs of inequalities must hold.

  1. if $j < j'$, $K+1-i-j(K-r) \leq 2K-i-j'(K-r) \leq K+1-i-j(K-r)+r-1$

2. if $j > j'$, $K+1-i-j(K-r) \leq K-i-j'(K-r) \leq K+1-i-j(K-r)+r-1$.

Consider the first inequality. First we prove that when $j < j'$, $K+1-i-j(K-r) \leq 2K-i-j'(K-r)$. To show this, we consider the following sequence of equations,

$$\begin{aligned}(2K-i-j'(K-r))-(K+1-i-j(K-r)) &= K-1-(j'-j)(K-r).\\ &\overset{(a)}{\geq} K-1-\left(\frac{r-1-i}{K-r}\right)(K-r).\\ &\geq K-r+i\\ &\overset{(b)}{\geq} K-r+1\\ &\overset{(c)}{\geq} K-(K-1)+1\\ &\geq 0,\end{aligned}$$

where (a) holds as the maximum value of $j'-j$ is equal to $\lfloor \frac{r-1-i}{K-r} \rfloor$, (b) holds as the minimum value of i is 1, and (c) holds as the maximum value of r is $K-1$.

Similarly,

$$\begin{aligned}(K+1-i-j(K-r)+r-1)-(2K-i-j'(K-r)) &= r-K+(j'-j)(K-r)\\ &\overset{(a)}{\geq} r-K+(K-r) \geq 0,\end{aligned}$$

where (a) holds as the minimum value of $j'-j$ is equal to 1.

Now, for the second inequality, we first prove that when $j > j'$, $K+1-i-j(K-r) \leq K-i-j'(K-r)$. To show this, we consider the following sequence of equations

$$\begin{aligned}(K-i-j'(K-r))-(K+1-i-j(K-r)) &= (j-j')(K-r)-1\\ &\overset{(a)}{\geq} K-r-1\\ &\overset{(b)}{\geq} K-(K-1)-1\\ &\geq 0,\end{aligned}$$

where (a) holds as the minimum value of $j-j'$ is equal to 1 and (b) holds as the maximum value of r is $K-1$.. Similarly,

$$\begin{aligned}(K+1-i-j(K-r)+r-1)-(K-i-j'(K-r)) &= r-(j-j')(K-r)\\ &\overset{(a)}{\geq} r-\left(\frac{r-1-i}{K-r}\right)(K-r)\\ &\geq 1+i\\ &\overset{(b)}{\geq} 0,\end{aligned}$$

where (a) holds as the maximum value of $j - j'$ is equal to $\lfloor \frac{r-1-i}{K-r} \rfloor$ and (b) holds as the minimum value of i is 1.

Hence, all the inequalities for both the cases are true. Similar arguments hold for the second broadcast as well.

- **Decoding for Scheme 2:** For each $i \in [r]$, a broadcast $W^{\{i\}}_{K-r+i} \oplus W^{\{K-r+i\}}_{K-r+i+1}$ is made (c.f. Lines 3, 5 in Algorithm 4). Now, node i contains the subsegment $W_{K-r+i+1}$ since $(K-r+i+1) \boxplus_K (r-1) = i$. Similarly, node $K-r+i$ clearly contains the subsegment W_{K-r+i}. Thus, node i can decode $W^{\{i\}}_{K-r+i}$ and node $K-r+i$ can decode $W^{\{K-r+i\}}_{K-r+i+1}$. Thus, we have verified the correctness of the transmission schemes.

- **Checking the merging phase:** Initially, for each $i \in [K]$, W_i is stored at the nodes $\{i \boxplus_K \langle r-1 \rangle\}$. After the transmissions are done, $\tilde{W}_i$ is obtained by merging some subsegments with W_i, for each $i \in \{1, \ldots, K\}$. Now, to verify that the merging can be done correctly, we need to show that all these subsegments are present at the nodes $\{i \boxplus_{K-1} \langle r-1 \rangle\}$ after the transmissions are done.

  - For each $i \in [r-1]$, consider the segment $\tilde{W}_{K-r+i}$ which is obtained by merging $W^{\{i\}}_{K-r+i}$ with $W^{\{K-r+i\}}_{K-r+i+1}$, (c.f. Algorithm 5 Line 2 and 3). We observe that $W^{\{i\}}_{K-r+i}$ was present in $\{(K-r+i) \boxplus_{K-1} \langle r \rangle\} \backslash \{i\}$ before rebalancing and was decoded by node i during the rebalancing process. Similarly, $W^{\{K-r+i\}}_{K-r+i+1}$ was present in $\{(K-r+i) \boxplus_{K-1} \langle r \rangle\} \backslash \{K-r+i\}$ before rebalancing and was decoded by node $K-r+i$ during the rebalancing process.
  - For each $i \in \{1, \ldots, \lfloor \frac{K-r}{2} \rfloor\}$, $\tilde{W}_i$ is obtained by merging W_i and $W_K^{\{(r-1+i) \boxminus_{K-1} \langle \min(r,i) \rangle\}}$, (c.f. Algorithm 5 Lines 7 and 14). Each node in $\{i \boxplus_{K-1} \langle r \rangle\}$ that does not contain W_K can obtain $W_K^{\{(r-1+i) \boxminus_{K-1} \langle \min(r,i) \rangle\}}$ using the broadcasts made in Lines 6-7 of Algorithms 3 and 4. Thus, $\tilde{W}_i$ can be obtained at the nodes $\{i \boxplus_{K-1} \langle r \rangle\}$.
  - We use similar arguments the target segments $\tilde{W}_i$ for each $i \in \{\lceil \frac{K-r}{2} \rceil + 1, \ldots, K-r\}$ and $\tilde{W}_{\frac{K-r+1}{2}}$, if $K-r$ is odd.

- **Checking the target database structure:** We first show that the sizes of all the segments after rebalancing are equal. For this, we look at how the new segments are formed by merging some subsegments with the older segment.

  - The size of the target segment $\tilde{W}_{K-r+i}$, for each $i \in \{1, \ldots, r-1\}$, is $\frac{K+r-2(i-1)-2}{2(K-1)} + \frac{K-r+2i}{2(K-1)} = \frac{K}{K-1}$ (c.f. Algorithm 5 - Line 2 and 3).
  - The size of the target segment $\tilde{W}_i$, for each $i \in [\lfloor \frac{K-r}{2} \rfloor]$, is $1 + \frac{2}{2(K-1)} = \frac{K}{K-1}$ (c.f. Algorithm 5 - Line 6, 7, 13, and 14).
  - For even $K-r$, the size of the target segment $\tilde{W}_i$, for each $i \in \{\frac{K-r}{2} + 1, \ldots, K-r\}$, is $1 + \frac{2}{2(K-1)} = \frac{K}{K-1}$ (c.f. Algorithm 5 - Line 9 and 10).
  - For odd $K-r$, the size of the target segment $\tilde{W}_i$, for each $i \in \{\frac{K-r+1}{2} + 1, \ldots, K-r\}$, is $1 + \frac{2}{2(K-1)} = \frac{K}{K-1}$ (c.f. Algorithm 5 - Line 16 and 17).

– The size of the target segment $\tilde{W}_{\frac{K-r+1}{2}}$ is $1 + \frac{1}{2(K-1)} + \frac{1}{2(K-1)} = \frac{K}{K-1}$. (c.f. Algorithm 5 - Line 19).

We can see that the sizes of all the segments after rebalancing are the same, i.e., $\frac{K}{K-1}$. This completes the verification of the correctness of our rebalancing algorithm, which assures that the target database is an r-balanced cyclic database on $K-1$ nodes.

2.5.5 Communication Load

We now calculate the communication loads of the two schemes. For the uncoded broadcasts in both schemes corresponding to lines 6, 7 in both Algorithms 3 and 4, the communication load incurred is $2\left(\frac{K-r-1}{2}.\frac{2}{2(K-1)} + \frac{1}{2(K-1)}\right) = \frac{K-r}{(K-1)}$, when $K-r$ is odd, and $2\left(\frac{K-r}{2}.\frac{2}{2(K-1)}\right) = \frac{K-r}{(K-1)}$ when $K-r$ is even, respectively. Now, we analyse the remainder of the communication loads of the two schemes.

2.5.5.1 Scheme 1

The coded broadcasts made in Scheme 1 are $\bigoplus_{j=0}^{\lfloor\frac{r-1-i}{K-r}\rfloor} W_{K+1-i-j(K-r)}^{\{K-i-j(K-r)\}}$ and $\bigoplus_{j=0}^{\lfloor\frac{r-1-i}{K-r}\rfloor} W_{K-r+i+j(K-r)}^{\{i+j(K-r)\}}$ for each $i \in [K-r]$ (c.f. Algorithm 3 Lines 2-5). Once again, the subsegments involved in the broadcast are padded so that they are of the same size. Consider $\bigoplus_{j=0}^{\lfloor\frac{r-1-i}{K-r}\rfloor} W_{K-r+i+j(K-r)}^{\{i+j(K-r)\}}$. The size of each subsegment involved is $\frac{K+r-2(i+j(K-r)-1)-2}{2(K-1)}$ Thus the subsegment having the maximum size is the one corresponding to $j=0$ having size $\frac{K+r-2i}{2(K-1)}$. Similarly, each subsegment involved in $\bigoplus_{j=0}^{\lfloor\frac{r-1-i}{K-r}\rfloor} W_{K+1-i-j(K-r)}^{\{K-i-j(K-r)\}}$ is of size $\frac{K-r+2(r-i-j(K-r))}{2(K-1)}$. Thus, the maximum size is the one corresponding to $j=0$ having size, $\frac{K+r-2i}{2(K-1)}$. Thus, the communication load is

$$\begin{aligned} L_1(r) &= 2 \cdot \sum_{i=1}^{K-r} \frac{K+r-2i}{2(K-1)} \\ &= \left(\frac{1}{K-1}\right)((K-r)(K+r) - (K-r)(K-r+1)) \\ &= \frac{(K-r)(2r-1)}{(K-1)}. \end{aligned}$$

2.5.5.2 Scheme 2

The coded broadcasts made in Scheme 2 involve $W_{K-r+i}^{\{i\}} \oplus W_{K-r+i+1}^{\{K-r+i\}}$ for each $i \in [r-1]$ (c.f. Algorithm 4 Lines 2-5). Since the smaller subsegment of each pair is padded to match the size of the larger, the cost involved in making the broadcast depends on the larger subsegment. Now, the size of these subsegments are $\frac{K+r-2j-2}{2(K-1)}$ and $\frac{K-r+2(j+1)}{2(K-1)}$ respectively (c.f. Figures 2.6, 2.7, 2.8). For the first subsegment to be larger, $\frac{K+r-2j-2}{2(K-1)} \geq \frac{K-r+2(j+1)}{2(K-1)}$. It is easy to verify that this occurs when $j \leq \frac{r}{2} - 1$. We separate our analysis into cases based on the parity of r.

For odd r, we have,

$$\begin{aligned}
L_2(r) &= \sum_{j=0}^{\frac{r-1}{2}-1} \frac{K+r-2j-2}{2(K-1)} + \sum_{j=\frac{r-1}{2}}^{r-2} \frac{K-r+2(j+1)}{2(K-1)} \\
&\overset{(a)}{=} \sum_{j=0}^{\frac{r-1}{2}-1} \frac{K+r-2j-2}{2(K-1)} + \sum_{j'=0}^{\frac{r-1}{2}-1} \frac{K-r+2(r-2-j'+1)}{2(K-1)} \\
&= 2 \cdot \sum_{j=0}^{\frac{r-1}{2}-1} \frac{K+r-2j-2}{2(K-1)} \\
&= \left(\frac{2}{2(K-1)}\right)\left(\left(\frac{r-1}{2}\right)(K+r-2) - \left(\frac{r-1}{2}-1\right)\left(\frac{r-1}{2}\right)\right) \\
&= \left(\frac{1}{2(K-1)}\right)(r-1)\left(K+\frac{r-1}{2}\right),
\end{aligned}$$

where (a) is obtained by changing the variable $j' = (r-2) - j$.

Similarly, for even r, we have,

$$\begin{aligned}
L_2(r) &= \sum_{j=0}^{\frac{r}{2}-1} \frac{K+r-2j-2}{2(K-1)} + \sum_{j=\frac{r}{2}}^{r-2} \frac{K-r+2(j+1)}{2(K-1)} \\
&= \sum_{j=0}^{\frac{r}{2}-2} \frac{K+r-2j-2}{2(K-1)} + \frac{K+r-2\left(\frac{r}{2}-1\right)-2}{2(K-1)} + \sum_{j=\frac{r}{2}}^{r-2} \frac{K-r+2(j+1)}{2(K-1)} \\
&\overset{(a)}{=} \sum_{j=0}^{\frac{r}{2}-2} \frac{K+r-2j-2}{2(K-1)} + \frac{K}{2(K-1)} + \sum_{j'=0}^{\frac{r}{2}-2} \frac{K-r+2(r-2-j'+1)}{2(K-1)} \\
&= 2 \cdot \sum_{j=0}^{\frac{r}{2}-2} \frac{K+r-2j-2}{2(K-1)} + \frac{K}{2(K-1)} \\
&= \left(\frac{2}{2(K-1)}\right)\left(\left(\frac{r}{2}-1\right)(K+r-2) - \left(\frac{r}{2}-2\right)\left(\frac{r}{2}-1\right)\right) + \frac{K}{2(K-1)} \\
&= \left(\frac{1}{2(K-1)}\right)(r-2)\left(K+\frac{r}{2}\right) + \frac{K}{2(K-1)},
\end{aligned}$$

where (a) is obtained by changing the variable $j' = (r-2) - j$.

The total communication load is therefore $L_{\mathsf{rem}}(r) = \frac{K-r}{(K-1)} + \min(L_1(r), L_2(r))$.

2.5.6 Advantage over the uncoded scheme

In this subsection, we bound the advantage of the rebalancing schemes presented in this work, over the uncoded scheme, in which the nodes simply exchange all the data which was available at the removed

node via uncoded transmissions. We know that the load of the uncoded scheme must thus be r. Consider the ratio of the communication load of Scheme 1 to that of the uncoded scheme. We then have the following sequence of equations.

$$\begin{aligned}\frac{\frac{K-r}{K-1}+L_1(r)}{L_u(r)} &= \frac{1}{r}\left(\frac{K-r}{K-1}+\frac{(K-r)(2r-1)}{K-1}\right)\\ &= \frac{K-r}{r(K-1)}\left(1+2r-1\right)\\ &= \frac{2(K-r)}{K-1} = 2\left(\frac{(K-1)-(r-1)}{K-1}\right)\\ &= 2\left(1-\frac{r-1}{K-1}\right).\end{aligned}$$

Now we do the same for Scheme 2.

$$\begin{aligned}\frac{\frac{K-r}{K-1}+L_2(r)}{L_u(r)} &= \frac{1}{r}\left(\frac{K-r}{K-1}+\frac{r-1}{2(K-1)}\left(K+\frac{r-1}{2}\right)\right)\\ &= \frac{1}{2r(K-1)}\left(2K-2r+rK-K+\frac{(r-1)^2}{2}\right)\\ &= \frac{1}{2r(K-1)}\left(K(r+1)+\frac{(r-1)^2-4r}{2}\right)\\ &< \frac{1}{2r(K-1)}\left(K(r+1)+\frac{r^2-4r}{2}\right)\\ &\leq \frac{(K-1+1)(r+1)}{2r(K-1)}+\frac{r-4}{4(K-1)}\\ &\leq \frac{1}{2}+\frac{r+K}{2r(K-1)}+\frac{r-4}{4(K-1)}\\ &\leq \frac{1}{2}+\frac{1}{2(K-1)}\left(\frac{K-r}{r}+\frac{r}{2}\right) < \frac{1}{2}+\frac{1}{2r}+\frac{r}{4(K-1)},\end{aligned}$$

where the last inequality follows by using the fact that $K-r \leq K-1$. Observe that $\frac{1}{2}+\frac{1}{2r}+\frac{r}{4(K-1)} < 1$, as $r \geq 3$ and $K-1 \geq r$. As the choice of the scheme selected for transmissions is based on the minimum load of Scheme 1 and Scheme 2, we have the result in the theorem. This completes the proof of the node-removal part of Theorem 2.

2.6 Rebalancing Scheme for Single Node Addition in Cyclic Databases

We consider the case of a r-balanced cyclic database system when a new node is added. Let this new empty node be indexed by $K+1$. For this imbalance situation, we present a rebalancing algorithm (Algorithm 6) in which nodes split the existing data segments and broadcast appropriate subsegments, so that the target database which is a r-balanced cyclic database on $K+1$ nodes, can be achieved. Each subsegment's size is assumed to be an integral multiple of $\frac{1}{K+1}$. This is without loss of generality, by the condition on the size T of each segment as in Theorem 2. Since node $K+1$ starts empty, there

are no coding opportunities; hence, the rebalancing scheme uses only uncoded transmissions. We show that this rebalancing scheme achieves a normalized communication load of $\frac{r}{K+1}$, which is known to be optimal from the results of [1]. This proves the node-addition part of Theorem 2.

Algorithm 6 Rebalancing Scheme for Single Node Addition

1: **procedure** DELIVERY SCHEME
2: **for** each $i \in [K]$ **do**
3: Split W_i into two subsegments, labelled $\tilde{W}_i$ of size $\frac{K}{K+1}$ and $W_i^{\{K+1\}\cup[\min(r-1,i-1)]}$ of size $\frac{1}{K+1}$.
4: Node i broadcasts $W_i^{\{K+1\}\cup[\min(r-1,i-1)]}$.
5: **end for**
6: Each node in $\{(K+1) \boxplus_{K+1} \langle r \rangle\}$ initializes the segment $\tilde{W}_{K+1}$ as an empty vector.
7: **for** each $i \in [K]$ **do**
8: Each node in $\{(K+1) \boxplus_{K+1} \langle r \rangle\}$ performs the concatenation $\tilde{W}_{K+1} = \tilde{W}_{K+1} | W_i^{\{K+1\}\cup[\min(r-1,i-1)]}$.
9: **end for**
10: **for** each $i = (K-r+2)$ to K **do**
11: Node i transmits $\tilde{W}_i$ to node $K+1$.
12: **end for**
13: **for** each $i = 1$ to $r-1$ **do**
14: Node i discards $\tilde{W}_{K-r+1+i}$.
15: **end for**
16: **end procedure**

2.6.1 Correctness

We verify the correctness of the rebalancing algorithm, i.e., we check that the target r-balanced cyclic database on $K+1$ nodes is achieved post-rebalancing. To achieve the target database, each target segment $\tilde{W}_i$, for $i \in [K+1]$, must be of size $\frac{K}{K+1}$ and stored exactly in $i \boxplus_{K+1} \langle r-1 \rangle$. Consider the segments $\tilde{W}_i$ for each $i \in [K-r+1]$. Since, these were a part of W_i, they are already present exactly at nodes $\{i \boxplus_{K+1} \langle r-1 \rangle\}$.

Now, consider the segments $\tilde{W}_i : i \in \{(K-r+1) \boxplus_K \langle r \rangle\}$. We recall S_i as the set of nodes where W_i is present in the initial database. Let $\tilde{S}_i$ be the set of nodes where it must be present in the final database. Now, the nodes where $\tilde{W}_i$ is not present and must be delivered to are given by $\tilde{S}_i \backslash S_i = \{K+1\}$. This is performed in lines 10-12 of Algorithm 6. Also, the nodes where W_i is present but $\tilde{W}_i$ must not be

present are given by $S_i \backslash \tilde{S}_i = \{i - K + r - 1\}$. This node discards $\tilde{W}_i$ in lines 13-15 of Algorithm 6. Finally, $\tilde{W}_{K+1}$ must be present in nodes $\{(K+1) \boxplus_{K+1} \langle r \rangle\}$. This can be obtained by the those nodes from the broadcasts made in line 4 in Algorithm 6 and the concatenation performed in lines 7-9.

Finally, It is easy to calculate that each target segment $\tilde{W}_i : i \in [K+1]$ is of size $\frac{K}{K+1}$. Thus, the final database satisfies the cyclic storage condition.

Note: Once the target segments $\tilde{W}_1, \ldots, \tilde{W}_{K+1}$ are recovered at the required nodes, any extra bits present at the node are discarded.

2.6.2 Communication Load

For broadcasting each $W_i^{\{K+1\} \cup [\min(r-1, i-1)]}$ of size $\frac{1}{K+1}$, $\forall i \in [K]$, the communication load incurred is $K.\frac{1}{K+1} = \frac{K}{K+1}$. Now, to transmit $\tilde{W}_{K-r+2} \ldots \tilde{W}_K$ to node $K+1$, the communication load incurred is $(r-1).\frac{K}{K+1}$. Hence, the total communication load is $L_{\mathsf{add}}(r) = \frac{rK}{K+1}$.

2.7 Comparisons between the existing schemes for single node removal

A comparison of the discussed schemes for single node removal is shown in Fig. 2.9 for the case of $K = 15$ (as r varies), along with the load $L_u(r)$ of the uncoded rebalancing scheme, and the lower bound based on the results of [1] ($\mathfrak{L}^*_{rem}(r)N = \frac{rK}{K(r-1)} = \frac{r}{r-1}$).

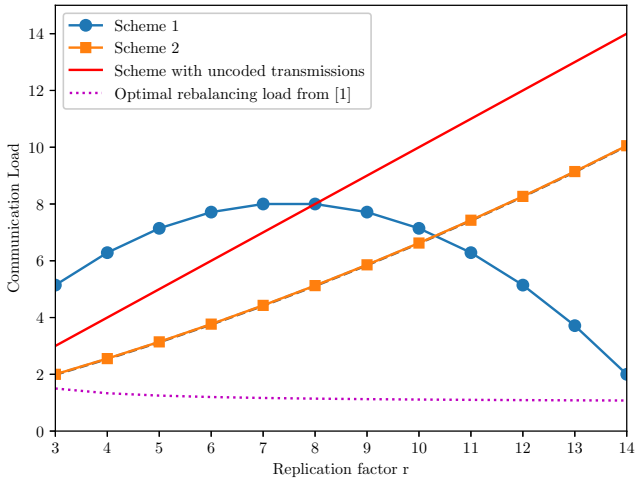

Figure 2.9: For $K = 15$, the figure shows comparisons of communication loads of Scheme 1 and Scheme 2 with the load of uncoded transmission scheme and the optimal load achieved by the scheme in [1], for varying r. Note that all curves are relevant only for $r \in \{3, \ldots, K-1\}$. We see that the minimum of the loads of the two schemes is always less than the uncoded load. Further, for any integer value of $r \geq 11$, we see that Scheme 1 has smaller load than Scheme 2, and the reverse is true otherwise.

Chapter 3

A New Low Complexity Distributed Computing Scheme via Subspace Designs

3.1 Introduction

A distributed computing framework is a software framework that enables the coordination of multiple computers in a distributed computing environment. The framework provides tools and APIs for programmers to write distributed applications that can be executed on a cluster of computers. There are many distributed computing frameworks available, each with its own set of features and capabilities. One of the most popular such frameworks is MapReduce [7]. MapReduce is a programming framework for processing large amounts of data in a distributed environment. It provides careful parallelization and distribution of the work across the network of computers.

In the MapReduce framework, the available data file is divided into smaller chunks and these chunks are assigned to different computing nodes. The overall computation in MapReduce is divided into two main phases, namely Map and Reduce. In the Map phase, the computing nodes process parts of the data that are assigned to them locally and generate some intermediate values (IVAs) using the Map functions. Following that, the nodes exchange the generated IVAs with each other to compute the outputs using the Reduce functions in a distributed fashion. This corresponds to the Shuffle Phase of the framework. This phase involves movement of high volumes of data between the nodes and the total amount of data shuffled (i.e., exchanged among the nodes) is called the communication load in the literature. The data shuffling problem in the context of distributed computing is widely studied, as it limits the performance of distributed computing applications. For instance, in a Facebook's Hadoop Cluster, about one-third of the total execution time is spent on data shuffling [23]. In order to address this issue, Coded MapReduce was introduced in [2] that exploits a specific form of coding in order to reduce the communication load involved in the data shuffling phase. Further, in [3], the authors proposed a Coded Distributed Computing Scheme (CDC). In particular, they have characterized a fundamental tradeoff between the computation load in the Map Phase and the communication load in the Shuffle Phase. The communication load achieved by the scheme in [2, 3] is shown to be optimal in [3]. In [24, 25, 26], coded MapReduce schemes were considered in which it was not required to compute

IVAs of all the stored subfiles. Furthermore, in [25, 26], tradeoffs between storage, computation, and communication were presented along with an optimal scheme that meets this tradeoff. In [27, 28], coded distributed computing schemes limited to linear computing functions in the presence of stragglers are discussed. This restriction was lifted in the works [29, 30], where they extend the model in [3] to arbitrary computing functions in the presence of stragglers.

Though some of the discussed results above are shown to be optimal in terms of communication load, the file complexity for these schemes is too large, i.e., exponential in the number of computing nodes. In [4, 5], the authors have presented distributed computing schemes using a binary matrix framework. They have shown how such binary matrices which produce distributed computing schemes arise out of combinatorial designs. The schemes presented require low file complexity, at the cost of higher local storage and communication load.

In this work, we present a scheme based on the principles of [4] (and its extended version [5]). In particular, we use the q-analogs of combinatorial designs, known as subspace designs, and present a distributed computing scheme based on them.

3.1.1 Contributions and Organization

This chapter presents a distributed computing scheme via subspace designs, which has low file complexity. The organization and contributions of this chapter are as follows. Previous works and results on Coded MapReduce are discussed in Section 3.2. We first review the Coded MapReduce setup [2, 3] in subsection 3.2.1. Following that, we give an introduction to the notion of binary matrices for distributed computing [4] and an example illustrating a distributed computing scheme via binary matrices in subsection 3.2.2. In Section 3.3, we review the important terminologies and constructions related to combinatorial designs and their q-analogs (subspace designs). In Section 3.4, we discuss the subspace designs based scheme and give its parameters. Finally, in Section 3.5, we conclude the chapter with numerical comparisons of our scheme with some existing baseline schemes.

3.2 Background and Previous Results

In this section, we first review the system model of Coded MapReduce presented in [2, 3] and discuss the main results of [3]. Further, we discuss the notion of binary matrices for distributed computing [4].

3.2.1 System Model for Coded MapReduce Distributed Computing

The framework of Coded MapReduce was introduced in [2, 3] and the formal system model was presented in [3]. We briefly review the same in this section.

The goal of this framework is to compute Q output functions on a file using K distributed computing nodes (servers). These servers are indexed by a set $\mathcal{K}$. The file on which the output functions are to be computed is divided into F subfiles, for $F \geq K$. These subfiles are indexed by a set $\mathcal{F}$. In the context of coded distributed computing, the parameter F is known as the file complexity. We now define the computation load as in [3].

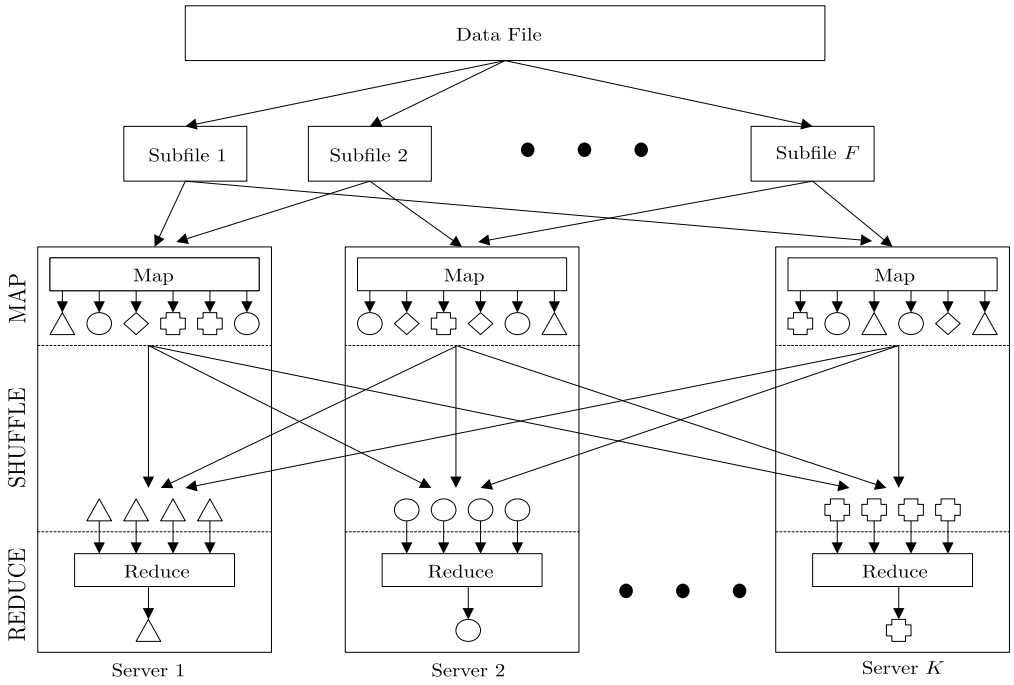

Figure 3.1: Workflow of a generic MapReduce framework on K servers. The three phases, namely Map, Shuffle, and Reduce phase are separated by a dotted line. The data file is first divided into F subfiles, which are then assigned to different nodes. In the Map phase, the nodes use Map functions to generate IVAs (shapes), which are then exchanged among the nodes in the Shuffle phase (shown using arrows) in order to compute the outputs using the Reduce functions in the Reduce phase.

Definition 2 (Computation Load [3])**.** *We define the computation load, denoted by* r*,* $1 \leq r \leq K$*, as the total number of Map functions computed across the* K *nodes, normalized by the number of subfiles (file complexity* F*). The computation load* r *can also be interpreted as the average number of nodes that map each subfile.*

By the above definition, each subfile is assigned to r nodes, $r \geq 1$. We denote the set of subfiles assigned to node k ($k \in \mathcal{K}$) as $\mathcal{M}_k \subseteq \mathcal{F}$. Now, we define the computing functions below.

- The Q output functions are denoted as $\phi_1, \ldots, \phi_Q$. Each ϕ_q maps all the input files to a fixed length binary stream $u_q = \phi_q(\{\forall f \in \mathcal{F}\})$.
- The map function $g_{q,f}, \forall q \in [Q], \forall f \in \mathcal{F}$ maps the input subfile $f \in \mathcal{F}$ into Q length-T intermediate values (IVAs), denoted as $\{v_{1,f}, \ldots, v_{Q,f}\}$. Each $v_{q,f} \triangleq g_{q,f}(f),\ q \in [Q],\ f \in \mathcal{F}$ is an IVA corresponding to the subfile f and the q^{th} map function.

- The reduce function $h_q, q \in [Q]$ maps the IVAs $v_{q,f} : \forall f \in \mathcal{F}$ into the output value u_q. Thus, $u_q = \phi_q(\{\forall f \in \mathcal{F}\}) = h_q(\{v_{q,f} : \forall f \in \mathcal{F}\}) = h_q(\{g_{q,f}(f) : \forall f \in \mathcal{F}\})$.

The computation in a distributed computing scheme in the MapReduce framework proceeds in three phases: *Map, Shuffle*, and *Reduce*, which we describe as follows:

1. *Map Phase:* Each node $k \in \mathcal{K}$ uses the map functions to compute all the IVAs of the subfiles in $\mathcal{M}_k$, i.e., the node k computes $g_{q,f}(f) : \forall f \in \mathcal{M}_k, \forall q$. Thus, the node k will have $\{v_{q,f} : \forall q \in [Q], \forall f \in \mathcal{M}_k\}$ after the map phase.

2. *Shuffle Phase:* A distinct set of $\beta = \frac{Q}{K}$ functions of the Q functions are assigned to each node. In order to compute the output of a reduce function, a node needs to have the IVAs of that output function for all the subfiles. Each node $k \in \mathcal{K}$ already has the IVAs corresponding to the subfiles in $\mathcal{M}_k$ (Map Phase). Thus, to reduce the functions assigned to it, it requires $\{v_{q,f} : \forall q \in \mathcal{W}_k, \forall f \notin \mathcal{M}_k\}$. Therefore, the nodes send broadcast transmissions to each other so that each node has all the IVAs it requires to reduce the functions assigned to it. This exchange of data (data shuffling) is known as the shuffle phase in the MapReduce framework.

3. *Reduce Phase:* Each node k computes $h_q(\{v_{q,f} : \forall f \in \mathcal{F}\})$ for each $q \in \mathcal{W}_k$, using the IVAs computed in the map phase and the received IVAs in the shuffle phase. This results in computing the value of the $\phi_q : \forall q \in \mathcal{W}_k$ on the input file.

Definition 3 (Communication load [3])**.** *Let T be the size of each IVA in bits. The communication load, denoted by L, $0 \leq L \leq 1$, is defined as the (normalized) total number of bits communicated by the K computing nodes during the Shuffle phase and can be calculated using the following.*

$$L \triangleq \frac{\textit{Total number of bits transmitted in shuffle phase}}{QFT}.$$

Let us consider the uncoded scheme, in which each node receives the required IVAs sent by other nodes without any coding. It is clear that QF IVAs are needed across the K nodes and $rF \cdot \frac{Q}{K} = \frac{rQF}{K}$ of them are already available (Map Phase). Thus, the communication load achieved by the uncoded scheme is

$$L_{\text{uncoded}} = \frac{(QFT - \frac{rQFT}{K})}{QFT} = 1 - \frac{r}{K}.$$

In [3], the authors have proposed Coded Distributed Computing (CDC). CDC uses a specific strategy to assign the computations of Map and Reduce functions across the K computing nodes, which enables coding opportunities during the shuffle phase. From the definition of computation load r, each subfile is mapped by a total of r nodes. The authors carefully design a mapping of the subfiles at r distinct nodes to enable maximal coding opportunities. This results in a communication load of

$$L = \frac{1}{r}(1 - \frac{r}{K}).$$

Observe that using CDC, the communication load is reduced exactly by a factor of r, when compared to the uncoded scheme.

The authors also apply CDC to a more general framework of distributed computing, where each output function is computed by $s \in \{1, \ldots, K\}$ nodes. This is done to provide better fault-tolerance.

The main result of [3] are given by the following theorem.

Theorem 3 ([3]). *The computation-communication function of the distributed computing framework, $L^*(r)$ is given by*

$$L^*(r) = L_{coded}(r) \triangleq \frac{1}{r} \cdot (1 - \frac{r}{K}), r \in \{1, \ldots, K\},$$

for sufficiently large T. For general $1 \leq r \leq K$, $L^*(r)$ is the lower convex envelope of the above points $\{(r, \frac{1}{r} \cdot (1 - \frac{r}{K})) : r \in \{1, \ldots, K\}\}$.

We refer the reader to Section V of [3] for the proof of achievability of the theorem above. The authors have also proved the converse of the theorem in Section VI of [3].

3.2.2 Binary Matrices and Distributed Computing

In [31] (and its extended version [32]), coded caching schemes were constructed using binary matrices, while in [4] (and its extended version [5]), distributed computing schemes were constructed using binary matrices. In this section, we review how a distributed computing scheme can be constructed using a binary matrix with constant column weight. We also provide an example illustrating a distributed computing scheme via binary matrices arising out of a specific combinatorial design.

Definition 4 (Binary Computing Matrix [4, 5]). *Consider a binary matrix C with rows indexed by a K-sized set $\mathcal{K}$ and columns indexed by a F-sized set $\mathcal{F}$ such that the number of 0's in any column is constant (say r). Then the matrix C defines a distributed computing scheme with K users (indexed by $\mathcal{K}$), file complexity F (subfiles indexed by $\mathcal{F}$) and computation load $= r$ as follows:*

- *Server $k \in \mathcal{K}$ maps subfile $f : \forall f \in \mathcal{F}$ if $C(k, f) = 0$ and does not map it if $C(k, f) = 1$.*

We then call the matrix C as a (K, F, r)-computing matrix.

The lemma below describes a single round of two transmissions using an identity submatrix of the computing matrix.

Lemma 1 ([4]). *Consider an identity submatrix of C given by rows $\{k_1, k_2, .., k_l : k_i \in \mathcal{K}\}$ and columns $\{f_1, f_2, .., f_l : f_i \in \mathcal{F}\}$, such that $C(k_i, f_i) = 1, \forall i \in [l]$, while $C(k_i, f_j) = 0, \forall i, j \in [l]$ where $i \neq j$. Then there exists two transmissions of length $\beta T = \frac{QT}{K}$ bits each, one coded and one uncoded, done by*

any two different servers k_i and $k_j : i, j \in [l], i \neq j$ such that each server $k_i : i \in [l]$ can recover the missing IVAs, $\{v_{q_{(k_i,b)},f_i} : \forall b \in [\beta]\}$, from these two coded transmissions.

Using lemma 1, the following theorem was presented in [4].

Theorem 4 ([4])**.** *Consider a computing matrix C of size $K \times F$ with a non-overlapping identity submatrix cover $\mathfrak{C} = \{C_1, C_2, .., C_S\}$ where the size of each identity submatrix is $g \geq 2$. Then, there exists a distributed computing scheme with K nodes, attaining computation load r and communication load $L = \frac{2}{g}\big(1 - \frac{r}{K}\big)$, with file complexity F.*

We refer the reader to Section III of [4] and [5] for the proof of Lemma 1 and Theorem 4 respectively. Given below is an example illustrating the new scheme using an identity submatrix cover of a given computing matrix.

Example 1. *Consider a set system $(\mathcal{K}, \mathcal{F})$ given by $\mathcal{K} = \{1, 2, 3, 4, 5, 6, 7\}$ and $\mathcal{F} = \{127, 145, 136, 467, 256, 357, 234\}$. The incidence matrix C for this set system is*

$$C = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \end{array} \begin{array}{c} \begin{array}{ccccccc} 127 & 145 & 136 & 467 & 256 & 357 & 234 \end{array} \\ \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 \end{pmatrix} \end{array}.$$

We can see that the above matrix is a $(7, 7, 4)$-computing matrix. The distributed computing system corresponding to this matrix has $K = 7$ nodes and $F = 7$ subfiles. Each subfile is stored in $r = 4$ nodes. For instance, the subfile indexed by 145 is stored in nodes $2, 3, 6$, and 7.

The identity submatrices (using 7 different shapes), each of size 3, of the above matrix form an identity submatrix cover, which consists of 7 non-overlapping identity submatrices.

We will now describe one round of transmissions, which consists of two transmissions corresponding to one identity submatrix. Consider the identity submatrix denoted as C_1, where

$$C_1 = \begin{array}{c} \\ 1 \\ 6 \\ 7 \end{array} \begin{array}{c} \begin{array}{ccc} 145 & 256 & 357 \end{array} \\ \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \end{array}.$$

$$C = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \end{array} \begin{array}{c} \begin{array}{ccccccc} 127 & 145 & 136 & 467 & 256 & 357 & 234 \end{array} \\ \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 \end{pmatrix} \end{array}$$

In C_1, the row indices $\{1, 6, 7\}$ represent the servers and the column indices $\{145, 256, 357\}$ represent the subfiles. One round of transmissions will consist of a coded transmission and an uncoded transmission, done by two different servers respectively. In this example, let server 1 do the coded transmission and server 6 do the uncoded transmission.

Let $Q = 14$ (i.e., $\beta = 2$) and $\mathcal{W}_1 = \{1, 8\}$, $\mathcal{W}_6 = \{6, 13\}$, $\mathcal{W}_7 = \{7, 14\}$. As $145 \notin \mathcal{M}_1$, server 1 will be missing the IVAs $\{v_{1,145},\ v_{8,145}\}$. Similarly, server 6 and 7 will be missing the IVAs $\{v_{6,256},\ v_{13,256}\}$ and $\{v_{7,357},\ v_{14,357}\}$ respectively. Now, server 1 will send the coded transmission $\{v_{7,357} \oplus v_{6,256},\ v_{14,357} \oplus v_{13,256}\}$. Observe that server 6 can decode $v_{6,256}$ and $v_{13,256}$ since it already has $v_{7,357}$ and $v_{14,357}$. Similarly, server 7 can decode $v_{7,357}$ and $v_{14,357}$. Server 6 sends the uncoded transmission $\{v_{1,145},\ v_{8,145}\}$, which is received by server 1.

All the other transmissions corresponding to the remaining identity submatrices are done in a similar manner and the decoding of the required IVAs at the respective servers is done successfully, according to Theorem 4 and finally, rate = $\frac{2}{3}(1 - \frac{4}{7}) = \frac{2}{7}$.

As mentioned in the previous section, the scheme in [3] achieves a communication load of $\frac{1}{r} \cdot (1 - \frac{r}{K})$, which was shown to be optimal. However, the file complexity required for this scheme is $\binom{K}{r}$. The distributed storage scheme corresponding to the Map Phase of the scheme in [3] can be represented using a computing matrix. This is captured in the following corollary to Theorem 4.

Corollary 1 ([4]). *For any positive integers K and $r \in [K]$, there exists a $(K, \binom{K}{r}, r)$-computing matrix, from which we get a distributed computing scheme on K nodes with computation load r and communication load $L = \frac{2}{r+1}\left(1 - \frac{r}{K}\right)$, with file complexity $F = \binom{K}{r}$. Further, this load $L < 2L^*(r)$, where $L^*(r)$ [3]is the optimal rate for a given computation load r.*

The proof of the above corollary can be found in Section X of [33].

3.2.2.1 Extensions to the Straggler Scenarios

Straggling nodes or stragglers are the nodes that are slower than the other nodes. The presence of straggling nodes is one of the practical issues in the distributed computing framework. In [4, 33], the authors have extended their schemes to the full and partial straggler scenarios. Full stragglers are the nodes that are unable to complete any map tasks completely. Thus, they can be considered identical to failed nodes and are not involved in any phase of the MapReduce framework. Partial stragglers are also slower than the other nodes by some factor, but they are not considered as failed nodes.

The following two theorems state the main results presented in [4, 33] with respect to the full and partial straggler scenarios, respectively. In both cases, the schemes presented are robust up to $g-2$ stragglers, where g is the size of any identity submatrix. We refer the reader to Section XI of [33] for the proofs of both the theorems given below.

Theorem 5 ([4, 33]). *Consider a computing matrix of size $K \times F$ with a non-overlapping identity submatrix cover $\mathfrak{C} = \{C_1, C_2, .., C_S\}$ where the size of each identity submatrix is $g \geq 2$. Then, there exists a distributed computing scheme with K nodes that is robust for $K-\kappa \in [0 : g-2]$ full stragglers, attaining computation load $= r$ and communication load $L(\kappa) = \frac{2}{g}\left(\frac{K}{\kappa} - \frac{r}{\kappa}\right)$, with file complexity F.*

Theorem 6 ([4, 33]). *Consider a computing matrix of size $K \times F$ with a non-overlapping identity submatrix cover $\mathfrak{C} = \{C_1, C_2, .., C_S\}$ where the size of each identity submatrix is $g \geq 2$. Then, there exists a distributed computing scheme with K nodes that is robust for $K - \kappa' \in [0 : g-2]$ partial stragglers, attaining computation load $= r$ and communication load $L(\kappa') = \frac{2}{g}\left(1 - \frac{r}{K}\right)$, with file complexity F.*

In [29] (and its extended version [34]), a similar setting was assumed for the full straggler scenario. The presented scheme is robust $K - \kappa \leq r - 1$ and achieves the following communication load

$$L^*(\kappa) = \left(1 - \frac{r}{K}\right) \sum_{i=r+\kappa-K}^{min\{r,\kappa-1\}} \frac{1}{i} \frac{\binom{r}{i}\binom{K-r-1}{\kappa-i-1}}{\binom{K-1}{\kappa-1}}.$$

The numerical comparisons with the above scheme can be found in Table VII of [33].

3.3 Background on Subspace Designs

We first review some of the basic definitions related to combinatorial designs and their constructions. For more details regarding the combinatorial designs, the reader is referred to [35, 36]. Later, we review some relevant aspects of subspace designs, which are the q-analogs of combinatorial designs.

Definition 5 (Design $(\mathcal{X}, \mathcal{A})$ [31]). *A design is a pair $(\mathcal{X}, \mathcal{A})$ such that the following properties are satisfied:*

(D1). $\mathcal{X}$ is a set of elements called points, and

(D2). $\mathcal{A}$ is a collection (i.e., multiset) of nonempty subsets of $\mathcal{X}$ called blocks.

We now define t-designs.

Definition 6 (t-designs [31])**.** *Let v, k, λ, and t be positive integers such that $v > k \geq t$. A t-(v, k, λ)-design (or simply t-design) is a design $(\mathcal{X}, \mathcal{A})$ such that the following properties are satisfied:*

(T1). $|\mathcal{X}| = v$,

(T2). Each block contains exactly k points, and

(T3). Every set of t distinct points is contained in exactly λ blocks.

Definition 7 (Subspace Designs)**.** *Let $\mathcal{V}$ be a vector space over the finite field $\mathbb{F}_q$ of dimension v. Let the subspaces with dimension k be called as k-dim subspaces. The q-analog of a design is defined as follows. Let $0 \leq t \leq k \leq v$ be integers and λ be a non-negative integer. A pair $\mathcal{D} = (\mathcal{V}, \mathcal{A})$, where $\mathcal{A}$ is s a collection of k-dim subspaces (blocks) of $\mathcal{V}$, is called a t-$(v, k, \lambda)_q$-subspace design on $\mathcal{V}$ if each t-dim subspace of $\mathcal{V}$ is contained in exactly λ blocks.*

A t-$(v, k, 1)$ subspace design is also referred to as a q-analog of an equivalent t-$(v, k, 1)$ design. We now recount some known constructions of subspace designs.

Example 2. *Few constructions of subspace designs known from literature are recollected here.*

1. *For any $0 \leq t \leq k \leq v$, and any q being a prime power, the collection of all k-dimensional subspaces of $\mathbb{F}_q^v$ forms a $t-(v, k, \langle v-t, k-t \rangle)_q$ subspace design. Specifically, when $t = k$, we get a $k-(v, k, 1)_q$ subspace design.*

2. *For any prime power q, it was shown in [37] that there exists a $1-(v, k, 1)_q$ design if and only if k divides v.*

3. *In [38, 39], a construction of nontrivial $2-(v, 3, q^2+q+1)_q$ designs was presented, for all q being a prime power and for all $v \geq 7$ such that v and the integer 24 are coprime.*

4. *The authors in [40] showed the existence of subspace designs over $\mathbb{F}_q$ for any t and any $k > 12(t+1)$, when n is sufficiently large.*

5. *Many other individual constructions for specific parameters are available in [37], some of which we use in this work to present numerical examples of the schemes obtained from subspace designs.*

3.4 Low File Complexity Scheme based on Binary Matrices from Subspace Designs

In Section VIII of [33], a coded caching scheme is presented via binary matrices arising out of subspace designs. We utilize the designed binary matrix as a scheme for MapReduce framework.

Let $(\mathcal{V}, \mathcal{A})$ denote a t-$(v,k,1)_q$-subspace design for $t \geq 2$. Let the blocks in this design be denoted by $\mathcal{A} = \{B_1, B_2, \ldots, B_b\}$. Let T denote the set of all 1-dim subspaces of $\mathcal{V}$, H denote the set of all t-dim subspaces of $\mathcal{V}$, and R denote the set of all $(t-1)$-dim subspaces of $\mathcal{V}$.

We construct a binary matrix C as follows. Let the rows of C be indexed by the set R. Let the columns be indexed by $\{(y, B) : y \in T, y \subset B, B \in \mathcal{A}\}$. The number of rows in matrix C is $\big\langle v, t-1 \big\rangle$. The number of columns in matrix C is $b\big\langle k, 1 \big\rangle = \frac{\big\langle v,t \big\rangle \big\langle k,1 \big\rangle}{\big\langle k,t \big\rangle}$. For some $D \in R$, the matrix $C = (C(D, (y, B)))$ is defined by the rule,

$$C(D,(y,B)) = \begin{cases} 1, & \text{if } D \bigoplus y \in H, D \bigoplus y \subset B \\ 0, & \text{otherwise.} \end{cases}$$

Remark 4. *The number of 1's in each row of C is $\big\langle v-t+1, 1 \big\rangle \cdot q^{t-1}$. We see this by noting that the distinct r-dim subspaces of $\mathbb{F}_q^v$ intersecting a fixed s-dim subspace in some l-dim subspace is $q^{(r-l)(s-l)}\big\langle v-s, r-l \big\rangle \big\langle s, l \big\rangle$ (see [41] for a proof). To count the number of 1's in each row, since each t-dim subspace is included precisely in one block of the design, we only have to count the number of t-dim subspaces of $\mathbb{F}_q^v$ intersecting a fixed $(t-1)$-dim subspace (identified by the row) in a $(t-1)$-dim subspace. This turns out to be precisely $\big\langle v-t+1, 1 \big\rangle \cdot q^{t-1}$. Similarly, we can obtain that the number of 1's in each column is $\big\langle k-1, t-1 \big\rangle \cdot q^{t-1}$ by the expression above. Hence, the binary computing matrix C defined above is also a constant row and column weight matrix.*

In the matrix C, the rows represent the set of servers, while the columns represent file complexity F. Thus, the matrix C gives us a $\left(\big\langle v, t-1 \big\rangle, \frac{\big\langle v,t \big\rangle \big\langle k,1 \big\rangle}{\big\langle k,t \big\rangle}, \big\langle v, t-1 \big\rangle - \big\langle k-1, t-1 \big\rangle q^{t-1} \right)$-computing matrix. For some $y \in T$, let $\mathcal{B}_y$ be the set of blocks containing y. Now,

$$|\mathcal{B}_y| = \lambda_1 = \frac{\big\langle v-1, t-1 \big\rangle}{\big\langle k-1, t-1 \big\rangle}.$$

Denote $\mathcal{B}_y$ by $\mathcal{B}_y = \{B_1, B_2, \ldots, B_{\lambda_1}\}$. For any B_i, denote by $D_i = \{D_{i,j} : j \in \big[\big\langle k-1, t-1 \big\rangle \cdot q^{t-1}\big]\}$ the set of all $(t-1)$-subspaces of B_i that do not contain y. In the next lemma, we describe an identity submatrix of the matrix C.

Lemma 2. *For some $j \in \big[\big\langle k-1, t-1 \big\rangle \cdot q^{t-1}\big]$ and $y \in T$, consider the submatrix $C_{y,j}$ of C defined as follows: the rows of $C_{y,j}$ are indexed by $\{D_{i,j} : \forall i \in [\lambda_1]\}$ and the columns are indexed by $\{(y, B_i) : B_i \in \mathcal{B}_y\}$. Then $C_{y,j}$ is an identity submatrix of C of size λ_1.*

The next two lemmas state that there is no overlap between the identity sub-matrices in the collection $\{C_{y,j} : \forall y \in T,\ \forall j \in \left[\left\langle k-1, t-1\right\rangle \cdot q^{t-1}\right]\}$ and that these identity sub-matrices will cover all the entries where $C(D,(y,B)) = 1$ in matrix C.

Lemma 3. *Any $C(D,(y,B)) = 1$ such that $y \in T, B \in \mathcal{A}, D \in R$ will be covered by exactly one identity submatrix of C (as defined in Lemma 2).*

Lemma 4. *The set of matrices $\{C_{y,j} : \forall y \in T,\ \forall j \in \left[\left\langle k-1, t-1\right\rangle \cdot q^{t-1}\right]\}$ forms a non-overlapping identity submatrix cover of C.*

We refer the reader to Section VIII of [33] for the proofs of lemmas 2, 3, and 4. Given below are the parameters for the distributed computing scheme based on subspace designs.

Number of servers K	**File Complexity** F	**Computation Load** r	**Communication Load for non/partial straggler case**	**Communication Load for** $K-\kappa$ **full straggler case**
$\left\langle v, t-1\right\rangle$	$\frac{\left\langle v,t\right\rangle \left\langle k,1\right\rangle}{\left\langle k,t\right\rangle}$	$\left\langle v, t-1\right\rangle - \left\langle k-1, t-1\right\rangle q^{t-1}$	$\frac{2\left\langle k-1,t-1\right\rangle^2 q^{t-1}}{\left\langle v,t-1\right\rangle \left\langle v-1,t-1\right\rangle}$	$\frac{2\left\langle k-1,t-1\right\rangle^2 q^{t-1}}{\kappa\left\langle v-1,t-1\right\rangle}$

Table 3.1: Parameters of distributed computing schemes based on subspace designs.

Example 3. *We consider the case in which $t = k$ (Example 2). This leads to a k-$(v,k,1)_q$ subspace design. For $v = 3, k = t = 2$, and $q = 4$, we get a distributed computing scheme with parameters $K = 21, F = 105$, and $r = 17$. Thus, we have a $(21, 105, 17)$-computing matrix.*

The identity submatrix cover of this matrix will consist of 84 identity submatrices. Each identity submatrix in this matrix will be of size 5 and thus the rate is $\frac{2}{5}(1 - \frac{17}{21}) = 0.076$.

3.5 Numerical Comparisons

In Table 3.2, we provide some numerical examples of our subspace design based construction for coded computing schemes. We provide the parameters of the design used and the resulting parameters of the coded computing scheme, the number of servers K, the file complexity F, the computation load r, along with the communication load for the no-straggler scenario as given by Theorem 4 (which matches the partial straggler load as given by Theorem 6), and the full-straggler load (as given by Theorem 5) for $K - \kappa \in \{1, 2\}$ stragglers. Finally, in Table 3.3, we compare our scheme with that from [29]. We see that our scheme has advantages in file-complexity, with increased communication loads.

Subspace Design Parameters	Number of servers K	File Complexity F	Computation Load r	L for non/partial straggler case	L for $K-\kappa=1$	L for $K-\kappa=2$
$2-(3,2,1)_2$	7	21	5	0.19	0.22	0.27
$4-(5,4,1)_2$	155	465	147	0.00688	0.00693	0.00697
$3-(4,3,1)_3$	130	520	121	0.01065	0.01073	0.01082
$4-(5,4,1)_3$	1210	4840	1183	0.0011157	0.0011166	0.0011175

Table 3.2: Numerical comparisons of communication loads of our schemes in non/partial straggler case and straggler case (applying the expressions in Table 3.1 to specific constructions of Section 3.3).

Number of servers K	Computation Load r	File Complexity F	File Complexity F in [29]	Number of non stragglers κ	Communication Load in Theorem 5	Optimal Communication Load in [29]
13	10	52	286	13	0.115	0.023
13	10	52	286	11	0.136	0.028
130	121	520	2.2×10^{13}	130	0.0106	0.00057
130	121	520	2.2×10^{13}	128	0.0108	0.00058
1210	1183	4840	1.18×10^{55}	1210	0.001115	1.887×10^{-5}
1210	1183	4840	1.18×10^{55}	1208	0.001117	1.889×10^{-5}

Table 3.3: Numerical comparisons between the scheme from [29] and subspace designs based computing schemes presented in this work. The load expression for the scheme in [29] is given in Section 3.2.2.1, while those of our schemes are compiled in Table 3.1 (in conjunction with specific constructions of Section 3.3).

Chapter 4

Future Work and Conclusions

We have addressed the problem of large file-size requirement in two of the known cache-enabled multi-receiver communication problems, namely Data Rebalancing in distributed file systems and Distributed Computing. We provide communication schemes for both settings with low file-size requirement, with a relatively high communication load as compared to the existing optimal schemes.

In Chapter 2, we have presented a XOR-based coded rebalancing scheme for the case of node removal and node addition in cyclic databases. Our scheme only requires the file size to be only cubic in the number of nodes in the system. For the node removal case, we present a coded rebalancing algorithm that chooses between the better of two coded transmission schemes in order to reduce the communication load incurred in rebalancing. We showed that the communication load of this rebalancing algorithm is always smaller than that of the uncoded scheme. For the node addition case, we present a simple uncoded scheme that achieves the optimal load.

We give a few comments regarding other future directions. Constructing good converse arguments in the cyclic database setting for the minimum communication load required for rebalancing from node-removal seems to be a challenging problem, due to the freedom involved in choosing the target database, the necessity of the target database to be balanced, and also because of the low file size requirement. Fig. 2.9 shows a numerical comparison of the loads of our schemes with the converse from [1]. However, the conditions of having a constrained file size or balanced target database are not used to show this converse, thus this bound is possibly quite loose for our cyclic placement setting. It would be certainly worthwhile to construct a tight converse for our specific setting. As we do not have a tight converse, it is also quite possible that rebalancing schemes for node removal exist with lower communication load for the cyclic placement setting. Designing such schemes would be an interesting future direction.

In Chapter 3, we have presented a novel distributed computing scheme via binary matrices arising out of subspace designs. The scheme has the advantage of low file complexity at the cost of a higher rate, as compared to the optimal scheme. Further, this scheme can further be extended to the full and partial straggler scenarios as well. Numerical comparisons with some existing baseline schemes are also shown, to illustrate the advantage of our scheme in terms of file complexity.

It is important to note that the presented scheme is primarily useful for the large local storage scenario. A possible future direction would be to consider wider classes of subspace designs to address the issue of large local storage requirement. In this work, we have used a t-$(v, k, 1)_q$-subspace design to produce a distributed computing scheme. Using higher values of λ is likely to address this issue. However, designing the communication schemes for the same appears to be difficult.

Appendix A

Proof of Claim 1

First, we can assume $K \geq 4$ without loss of generality, as our replication factor r lies between $\{3, \ldots, K-1\}$. Consider the expressions in Theorem 2 for $L_1(r)$ and $L_2(r)$ as continuous functions of r. Also, consider $L_{2,o}(r) = \frac{K-r}{(K-1)} + \frac{r-1}{2(K-1)}\left(K + \frac{r-1}{2}\right)$ be the continuous function of r which matches with $\frac{K-r}{K-1} + L_2(r)$ in Theorem 2 for odd values of $r \in \{3, \ldots, K-1\}$. Similarly, let $L_{2,e}(r) = \frac{K-r}{(K-1)} + \frac{r-2}{2(K-1)}\left(K + \frac{r}{2}\right) + \frac{K}{2(K-1)}$ be the continuous function of r which matches with $\frac{K-r}{K-1} + L_2(r)$ in Theorem 2 for even values of $r \in \{3, \ldots, K-1\}$.

Let r_o be a real number in the interval $[3 : K-1]$ such that $L_{2,o}(r_o) = L_1(r_o)$. Similarly, let r_e be a real number r in the interval $[3 : K-1]$ such that $L_{2,e}(r_e) = L_1(r_e)$.

With these quantities set up, the proof then proceeds according to the following steps.

1. Firstly, we show that $L_{2,o}(r) > L_{2,e}(r)$ for $K \geq 4$.

2. We then find the values of r_o and r_e, which turn out to be unique. We also show that $r_e > r_o$ and that $\lceil r_e \rceil = \lceil \frac{2K+2}{3} \rceil = \lfloor r_o \rfloor + 1$.

3. Then, we shall show that for any integer $r > r_e$, we have $L_1(r) < L_{2,e}(r)$. Further, we will also show that for any integer $r < r_o$, we have $L_{2,o}(r) < L_1(r)$.

It then follows from the above steps that the threshold value is precisely $r_{\mathsf{th}} = \lceil r_e \rceil = \lceil \frac{2K+2}{3} \rceil$. We now show the above steps one by one.

1) Proof of $L_{2,o}(r) > L_{2,e}(r)$ for $K \geq 4$: We have that

$$\begin{aligned}
L_{2,o}(r) - L_{2,e}(r) &= \frac{r-1}{2(K-1)}\left(K + \frac{r-1}{2}\right) - \frac{r-2}{2(K-1)}\left(K + \frac{r}{2}\right) - \frac{K}{2(K-1)} \\
&= \frac{(r-1)(2K+r-1) - (r-2)(2K+r) - 2K}{4(K-1)} \\
&= \frac{2rK - 2K + r^2 - 2r + 1 - 2rK + 4K - r^2 + 2r - 2K}{4(K-1)} \\
&= \frac{1}{4(K-1)} > 0,
\end{aligned}$$

which holds as $K \geq 4$. Hence, $L_{2,o}(r) > L_{2,e}(r)$ for $K \geq 4$.

2) Finding r_o and r_e and their relationship: Calculating $L_{2,o}(r) - L_1(r)$, we get the following

$$\begin{aligned} L_{2,o}(r) - L_1(r) &= \frac{r-1}{2(K-1)}\left(K + \frac{r-1}{2}\right) - \frac{(K-r)(2r-1)}{(K-1)} \\ &= \frac{(r-1)(2K+r-1) - 4(K-r)(2r-1)}{4(K-1)} \\ &= \frac{9r^2 - 6r(K+1) + 2K + 1}{4(K-1)}. \end{aligned}$$

Solving for r from $L_{2,o}(r) - L_1(r) = 0$ with the condition that $r \geq 3$, we get

$$r = \frac{2K+1}{3}.$$

It is easy to see that $\frac{2K+1}{3} > 2$ for $K \geq 4$. Also, we can check that $\frac{2K+1}{3} < (K-1)$, when $K \geq 4$. Thus, we have that $r_o = \frac{2K+1}{3}$. By similar calculations for $L_{2,e}(r) - L_1(r)$, we see that $r_e = \frac{K+1+\sqrt{K^2+1}}{3}$. Further, we observe that $r_e - r_o = \frac{\sqrt{K^2+1}-K}{3}$, for $K \geq 4$. Hence, $r_e > r_o$ for $K \geq 4$. Also observe that $\lceil r_e \rceil \leq \lceil \frac{2K+2}{3} \rceil = \lfloor \frac{2K+1}{3} \rfloor + 1 = \lfloor r_o \rfloor + 1$, where the first equality holds as $K \geq 4$. Now, as $r_e > r_o$ we have that $\lceil r_e \rceil > \lfloor r_o \rfloor$. Thus, we see that $\lceil r_e \rceil = \lceil \frac{2K+2}{3} \rceil = \lfloor r_o \rfloor + 1$.

Proof of 3): We now prove that for any integer r if $r > r_e$, then $L_1(r) < L_{2,e}(r)$. Let $r = r_e + a$, for some real number $a > 0$ such that $r_e + a$ is an integer. We know that $L_1(r_e) = L_{2,e}(r_e)$. Consider the following sequence of equations.

$$\begin{aligned} L_1(r) &= L_1(r_e + a) \\ &= \frac{K - (r_e + a)}{(K-1)} + \frac{(K - (r_e + a))(2(r_e + a) - 1)}{(K-1)} \\ &= \frac{K - r_e}{K-1} - \frac{a}{K-1} + \frac{(K - r_e - a)\,(2r_e - 1 + 2a)}{K-1} \\ &= \frac{K - r_e}{K-1} + \frac{(K - r_e)(2r_e - 1)}{K-1} + \frac{-a - a(2r_e - 1 + 2a) + 2a(K - r_e)}{K-1} \\ &= L_1(r_e) + \frac{2aK - 2a^2 - 4ar_e}{K-1} \\ &= L_{2,e}(r_e) + \frac{2aK - 2a^2 - 4ar_e}{K-1}. \end{aligned}$$

Similarly, consider the following.

$$\begin{aligned} L_{2,e}(r) &= L_{2,e}(r_e + a) \\ &= \frac{K - (r_e + a)}{(K-1)} + \frac{(r_e + a) - 2}{2(K-1)}\left(K + \frac{r_e + a}{2}\right) \\ &= \frac{K - r_e}{(K-1)} - \frac{a}{K-1} + \frac{(r_e - 2) + a}{2(K-1)}\left(K + \frac{(_e + a}{2}\right) \\ &= \frac{K - r_e}{(K-1)} + \frac{(r_e - 2)\left(K + \frac{r_e}{2}\right)}{2(K-1)} + \frac{-6a + a^2 + 2aK + 2ar_e}{4(K-1)} \end{aligned}$$

$$= L_{2,e}(r_e) + \frac{-6a + a^2 + 2aK + 2ar_e}{4(K-1)}.$$

$$\text{Now, } L_{2,e}(r) - L_1(r) = \frac{18ar_e - 6aK + 9a^2 - 6a}{4(K-1)}$$
$$= \frac{a(18r_e - 6K + 9a - 6)}{4(K-1)}$$
$$\overset{(a)}{>} \frac{a(6\sqrt{K^2+1} + 9a)}{4(K-1)}$$
$$\overset{(b)}{>} 0,$$

where (a) holds on substituting $r_e = \frac{K+1+\sqrt{K^2+1}}{3}$, (b) holds as $K, a > 0$. Therefore, when $r > r_e$, $L_1(r) < L_{2,e}(r)$.

We now prove that for any integer r if $r < r_o$, then $L_{2,o}(r) < L_1(r)$. Let $r = r_o - a$, for some real number $a < r_o$ such that $r_o - a$ is an integer. We know that $L_1(r_o) = L_{2,o}(r_o)$. Consider the following sequence of equations.

$$\begin{aligned}
L_1(r) &= L_1(r_o - a) \\
&= \frac{K - (r_o - a)}{(K-1)} + \frac{(K - (r_o - a))(2(r_o - a) - 1)}{(K-1)} \\
&= \frac{K - r_o}{K-1} + \frac{a}{K-1} + \frac{(K - r_o + a)\,(2r_o - 1 - 2a)}{K-1} \\
&= \frac{K - r_o}{K-1} + \frac{(K - r_o)(2r_o - 1)}{K-1} + \frac{a + a(2r_o - 1 - 2a) - 2a(K - r_o)}{K-1} \\
&= L_1(r_o) + \frac{4ar_o - 2aK - 2a^2}{K-1} \\
&= L_{2,o}(r_o) + \frac{4ar_o - 2aK - 2a^2}{K-1}.
\end{aligned}$$

Similarly, consider the following.

$$\begin{aligned}
L_{2,o}(r) &= L_{2,o}(r_o - a) \\
&= \frac{K - (r_o - a)}{(K-1)} + \frac{(r_o - a) - 1}{2(K-1)}\left(K + \frac{(r_o - a) - 1}{2}\right) \\
&= \frac{K - r_o}{(K-1)} + \frac{a}{K-1} + \frac{(r_o - 1) - a}{2(K-1)}\left(K + \frac{(r_o - 1) - a}{2}\right) \\
&= \frac{K - r_o}{(K-1)} + \frac{(r_o - 1)\left(K + \frac{(r_o - 1)}{2}\right)}{2(K-1)} + \frac{6a - 2ar_o + a^2 - 2aK}{4(K-1)} \\
&= L_{2,o}(r_o) + \frac{6a - 2ar_o + a^2 - 2aK}{4(K-1)}.
\end{aligned}$$

$$\text{Now, } L_1(r) - L_{2,o}(r) = \frac{18ar_o - 6aK - 9a^2 - 6a}{4(K-1)}$$

$$
\begin{aligned}
&= \frac{a(18r_o - 6K - 9a - 6)}{4(K-1)} \\
&= \frac{3a(3r_o - 3a + 3r_o - 2K - 2)}{4(K-1)} \\
&\overset{(a)}{\geq} \frac{3a(3 + 3r_o - 2K - 2)}{4(K-1)} \\
&\geq \frac{3a(3r_o - 2K + 1)}{4(K-1)} \\
&\overset{(b)}{\geq} \frac{3a(1+1)}{4(K-1)} \\
&> 0,
\end{aligned}
$$

where (a) holds because $r_o - a \geq 1$ and (b) holds on substituting $r_o = \frac{2K+1}{3}$. Therefore, when $r < r_o$, $L_{2,o}(r) < L_1(r)$. The proof is now complete.

Related Publications

Conferences

- Athreya Chandramouli, **Abhinav Vaishya**, Prasad Krishnan. "Coded data rebalancing for distributed data storage systems with cyclic storage." In 2022 IEEE Information Theory Workshop (ITW), pp. 618-623. IEEE, 2022.

Journals (Under Review)

- Shailja Agrawal, K V Sushena Sree, Prasad Krishnan, **Abhinav Vaishya**, Srikar Kale. "Cache-Aided Communication Schemes via Combinatorial Designs and their q-analogs." arXiv preprint arXiv:2302.03452 (2023). [*Submitted to IEEE Journal on Selected Areas in Information Theory (JSAIT), 2023.*]

Preprints

- Athreya Chandramouli, **Abhinav Vaishya**, and Prasad Krishnan. "Coded Data Rebalancing for Distributed Data Storage Systems with Cyclic Storage." arXiv preprint arXiv:2205.06257 (2022).

Bibliography

[1] Prasad Krishnan, V Lalitha, and Lakshmi Natarajan. Coded data rebalancing: Fundamental limits and constructions. In *2020 IEEE International Symposium on Information Theory (ISIT)*, pages 640–645. IEEE, 2020.

[2] Songze Li, Mohammad Ali Maddah-Ali, and A Salman Avestimehr. Coded mapreduce. In *2015 53rd Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pages 964–971. IEEE, 2015.

[3] Songze Li, Mohammad Ali Maddah-Ali, Qian Yu, and A Salman Avestimehr. A fundamental tradeoff between computation and communication in distributed computing. *IEEE Transactions on Information Theory*, 64(1):109–128, 2017.

[4] Shailja Agrawal and Prasad Krishnan. Low complexity distributed computing via binary matrices with extension to stragglers. In *2020 IEEE International Symposium on Information Theory (ISIT)*, pages 162–167. IEEE, 2020.

[5] S. Agrawal and P. Krishnan. Low complexity distributed computing via binary matrices with extension to stragglers. *ArXiv*, abs/2001.05438, 2020.

[6] Dhruba Borthakur. The hadoop distributed file system: Architecture and design. *Hadoop Project Website*, 11(2007):21, 2007.

[7] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[8] Mohammad Ali Maddah-Ali and Urs Niesen. Fundamental limits of caching. *IEEE Transactions on Information Theory*, 60(5):2856–2867, 2014.

[9] Kangwook Lee, Maximilian Lam, Ramtin Pedarsani, Dimitris Papailiopoulos, and Kannan Ramchandran. Speeding up distributed machine learning using codes. *IEEE Transactions on Information Theory*, 64(3):1514–1529, 2017.

[10] Kai Wan, Daniela Tuninetti, Mingyue Ji, Giuseppe Caire, and Pablo Piantanida. Fundamental limits of decentralized data shuffling. *IEEE Transactions on Information Theory*, 66(6):3616–3637, 2020.

[11] Adel Elmahdy and Soheil Mohajer. On the fundamental limits of coded data shuffling for distributed machine learning. *IEEE Transactions on Information Theory*, 66(5):3098–3131, 2020.

[12] Data rebalancing in apache ignite (apache ignite documentation). `https://ignite.apache.org/docs/latest/data-rebalancing`.

[13] Data rebalancing in apache hadoop (apache hadoop documentation). `https://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/HdfsUserGuide.html#Balancer`.

[14] Eugene Kirpichov and Malo Denielou. No shard left behind: dynamic work rebalancing in google cloud dataflow. `https://cloud.google.com/blog/products/gcp/no-shard-left-behind-dynamic-work-rebalancing-in-google-cloud-dataflow`, 2016.

[15] Rebalancing in ceph (ceph architecture). `https://docs.ceph.com/en/latest/rados/operations/balancer`.

[16] Mohammad Ali Maddah-Ali and Urs Niesen. Fundamental limits of caching. *IEEE Transactions on information theory*, 60(5):2856–2867, 2014.

[17] KV Sushena Sree and Prasad Krishnan. Coded data rebalancing for decentralized distributed databases. In *2020 IEEE Information Theory Workshop (ITW)*, pages 1–5. IEEE, 2021.

[18] K. Renuga, S.S. Tan, Y.Q. Zhu, T.C. Low, and Y.H. Wang. Balanced and efficient data placement and replication strategy for distributed backup storage systems. In *2009 International Conference on Computational Science and Engineering*, volume 1, pages 87–94, 2009.

[19] R. Marcelin-Jimenez, S. Rajsbaum, and B. Stevens. Cyclic storage for fault-tolerant distributed executions. *IEEE Transactions on Parallel and Distributed Systems*, 17(9):1028–1036, 2006.

[20] Nicholas Woolsey, Rong-Rong Chen, and Mingyue Ji. Uncoded placement with linear submessages for private information retrieval from storage constrained databases. *IEEE Transactions on Communications*, 68(10):6039–6053, 2020.

[21] Mingyue Ji, Xiang Zhang, and Kai Wan. A new design framework for heterogeneous uncoded storage elastic computing. *CoRR*, abs/2107.09657, 2021.

[22] Qifa Yan, Minquan Cheng, Xiaohu Tang, and Qingchun Chen. On the placement delivery array design for centralized coded caching scheme. *IEEE Transactions on Information Theory*, 63(9):5821–5833, 2017.

[23] Mosharaf Chowdhury, Matei Zaharia, Justin Ma, Michael I Jordan, and Ion Stoica. Managing data transfers in computer clusters with orchestra. *ACM SIGCOMM computer communication review*, 41(4):98–109, 2011.

[24] Y. H. Ezzeldin, M. Karmoose, and C. Fragouli. Communication vs distributed computation: An alternative trade-off curve. In *2017 IEEE Information Theory Workshop (ITW)*, pages 279–283, Nov 2017.

[25] Q. Yan, S. Yang, and M. Wigger. A storage-computation-communication tradeoff for distributed computing. In *2018 15th International Symposium on Wireless Communication Systems (ISWCS)*, pages 1–5, Aug 2018.

[26] Qifa Yan, Sheng Yang, and Michele Wigger. Storage, computation, and communication: a fundamental tradeoff in distributed computing. In *2018 IEEE Information Theory Workshop (ITW)*, pages 1–5. IEEE, 2018.

[27] K. Lee, M. Lam, R. Pedarsani, D. Papailiopoulos, and K. Ramchandran. Speeding up distributed machine learning using codes. *IEEE Transactions on Information Theory*, 64(3):1514–1529, March 2018.

[28] S. Li, M. A. Maddah-Ali, and A. S. Avestimehr. A unified coding framework for distributed computing with straggling servers. In *2016 IEEE Globecom Workshops (GC Wkshps)*, pages 1–6, Dec 2016.

[29] Q. Yan, M. Wigger, S. Yang, and X. Tang. A fundamental storage-communication tradeoff in distributed computing with straggling nodes. In *2019 IEEE International Symposium on Information Theory (ISIT)*, pages 2803–2807, July 2019.

[30] V. Ramkumar and P. V. Kumar. Coded mapreduce schemes based on placement delivery array. In *2019 IEEE International Symposium on Information Theory (ISIT)*, pages 3087–3091, July 2019.

[31] S. Agrawal, K. V. Sushena Sree, and P. Krishnan. Coded caching based on combinatorial designs. In *2019 IEEE International Symposium on Information Theory (ISIT)*, pages 1227–1231, July 2019.

[32] Shailja Agrawal, K V Sushena Sree, and Prasad Krishnan. Coded caching based on combinatorial designs. *ArXiv*, abs/1901.06383, 2019.

[33] Shailja Agrawal, KV Sree, Prasad Krishnan, Abhinav Vaishya, and Srikar Kale. Cache-aided communication schemes via combinatorial designs and their q-analogs. *arXiv preprint arXiv:2302.03452*, 2023.

[34] Qifa Yan, Michèle Wigger, Sheng Yang, and Xiaohu Tang. A fundamental storage-communication tradeoff for distributed computing with straggling nodes. *IEEE Transactions on Communications*, 68(12):7311–7327, 2020.

[35] Douglas R Stinson. *Combinatorial designs: constructions and analysis*. Springer Science & Business Media, 2007.

[36] Charles J Colbourn and Jeffrey H Dinitz. *Handbook of combinatorial designs*. CRC press, 2006.

[37] Michael Braun, Michael Kiermaier, and Alfred Wassermann. *q-Analogs of Designs: Subspace Designs*, pages 171–211. Springer International Publishing, Cham, 2018.

[38] Hiroshi Suzuki. 2-designs over $gf(2^m)$. *Graphs and Combinatorics*, 6(3):293–296, Sep 1990.

[39] Hiroshi Suzuki. 2-designs over gf(q). *Graphs and Combinatorics*, 8(4):381–389, Dec 1992.

[40] Arman Fazeli, Shachar Lovett, and Alexander Vardy. Nontrivial t-designs over finite fields exist for all t. *Journal of Combinatorial Theory, Series A*, 127:149–160, 2014.

[41] JWP Hirschfeld. *Projective Geometries Over Finite Fields. Oxford Mathematical Monographs*. Oxford University Press New York, 1998.