CODES FOR DISTRIBUTED MACHINE LEARNING

A Thesis

by

ADARSH MUTHUVEERU SUBRAMANIAM

Submitted to the Office of Graduate and Professional Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Chair of Committee,	Krishna Narayanan
Committee Members,	Alex Sprintson
	Chao Tian
	Frank Sottile
Head of Department,	Miroslav M. Begovic

May 2020

Major Subject: Electrical Engineering

Copyright 2020 Adarsh Muthuveeru Subramaniam

ABSTRACT

The problem considered is that of distributing machine learning operations of matrix multiplication and multivariate polynomial evaluation among computer nodes a.k.a worker nodes some of whom don't return their outputs or return erroneous outputs. The thesis can be divided into three major parts.

In the first part of the thesis, a fault tolerant setup where t worker nodes return erroneous values is considered. For an additive random Gaussian error model, it is shown that for all $t < N - K$, errors can be corrected with probability 1 for polynomial codes.

In the second part of the thesis, a class of codes called random Khatri-Rao-Product (RKRP) codes for distributed matrix multiplication in the presence of stragglers is proposed. The main advantage of the proposed codes is that decoding of RKRP codes is highly numerically stable in comparison to decoding of Polynomial codes [67] and decoding of the recently proposed OrthoPoly codes [18]. It is shown that RKRP codes are maximum distance separable with probability 1.

In the third part of the thesis, the problem of distributed multivariate polynomial evaluation (DPME) is considered, where Lagrange Coded Computing (LCC) [66] was proposed as a coded computation scheme to provide resilience against stragglers for the DPME problem. A variant of the LCC scheme, termed Product Lagrange Coded Computing (PLCC) is proposed by combining ideas from classical product codes and LCC. The main advantage of PLCC is that they are more numerically stable than LCC;

DEDICATION

To Amma and Appa

ACKNOWLEDGMENTS

First and foremost I would like to thank my advisor, Prof.Krishna Narayanan for his immense help in writing this thesis. Throughout my Masters program, he has been a source of guidance and constructive criticism. I would like to thank him for introducing me to the problem of 'coded computing' and guiding me to publish 3 papers.

I would also like to immensely thank Professor Anoosheh Heidarzadeh for his constant support and valuable advice. I have enjoyed the various discussions I have had with him over the course of my Masters degree.

Thanks are also due, to several of my fellow graduate students Asit, Vamsi, Priyanka, Nagaraj, Arman, Desik, Kaushik and many others at Texas A&M university for their valuable advise and cheerful talk.

TABLE OF CONTENTS

Page

LIST OF FIGURES

FIGURE Page

LIST OF TABLES

TABLE Page

1. INTRODUCTION

1.1 Overview of the Dissertation

In the data-centric age, distributed large scale machine learning has become a paradigm of prime importance[1]. Distributed machine learning involves distributing the machine learning task among various computer nodes, commonly called worker nodes, and combining the outputs of the worker nodes at a central/master node to get the original machine learning task's output.

Recent work in the area of distributed machine learning that use techniques from coding theory to encode the data input to the machine learning algorithm has shown remarkable improvement in performance in terms of straggler tolerance compared to traditional distributed computing methods. A key parameter in distributed machine learning is the minimum fraction of workers that have to return their outputs for the master node to construct the solution to the original problem. Smaller the fraction, better is the coding scheme since it can tolerate a large number of failing/slow worker nodes (stragglers). Techniques based on coding theory have been shown to have much better performance compared to traditional distributed computing schemes using repetition coding, in terms of the fraction of workers that have to return in order to compute the final solution. Even though some of these schemes have optimal theoretical properties, practical implementation poses some challenges. Since the arithmetic on computers have a finite bit-precision, practical implementations suffer from numerical instability while trying to reconstruct the solution at the master node.

This thesis studies two important machine learning problems. i) distributed matrix multiplication, and ii) distributed polynomial evaluation. Distributed matrix multiplication, which involves multiplying two large matrices, is very important in machine learning. Gradient computation at each layer in a Neural Network as part of the Backprop-

[1]The contents of this section have been presented in verbatim as part of the thesis proposal of the author

agation algorithm and computation of the output of a layer of the Neural Network can be considered as matrix multiplication problems. Important machine learning problems like linear regression and tensor product computation can be considered as polynomial evaluation problems. The thesis constructs new codes that can implement solutions to the aforementioned problems in a numerically stable and distributed manner. The thesis also proposes a new decoding algorithm called “Collaborative Peterson’s algorithm" to existing polynomial-based distributed computing codes which will substantially increase the resilience of distributed matrix multiplication in the presence of random additive errors in the computation.

1.2 Motivation for the thesis

A typical distributed computing scenario consists of a master node which distributes its work to several worker nodes [9,72]. There are three important problems that have to be addressed in a distributed algorithm for such a scenario.

1. Straggler resiliency - resilience to straggling workers where each worker takes a random amount of time to respond.

2. Fault/Adversarial tolerance - Worker nodes introducing random or Byzantine errors respectively.

3. Scalability - The algorithm must be numerically stable when the number of workers increases, which is an issue with existing algorithms.

This thesis considers two important distributed computation tasks;

1. Matrix multiplication [17,27,70] which is essential in computing the gradient [53] of a Neural Network which is part of the backpropagation algorithm and in evaluating the output of a Neural Network.

2. Multivariate polynomial evaluation on massive data-sets. Machine learning problems like linear regression and tensor computations are essentially multivariate poly-

nomial evaluations [66].

Recently, coding theoretic techniques applied to the above problems to eliminate stragglers have shown significant improvements over traditional Distributed computing literature which doesn't take into account the particular type of operation performed. This has led to a number of papers in this area of *coded distributed computing* [1,2,4,6–8,10,11,15–17, 19,21,23–25,28,31,34–37,39,40,42,45,47,48,50,54,59–62,64–66,69,70]. Polynomial based codes (Reed-Solomon type) have been shown to be optimal for the problem of matrix multiplication [17,69,70] in terms of number of stragglers that can be tolerated. Further the work of Lagrange coded computing [66] has been shown to be optimal in terms of straggler resiliency for the multivariate polynomial evaluation problem. However, these polynomial based codes are highly unstable as the error in recovering the message from the encoded polynomial evaluations in the Real Field involves inverting a Vandermonde matrix. The condition number of Vandermonde matrices grows exponentially in the size of the matrix [43] (size of the matrix depends on the size of the code) and hence the operation is *numerically unstable*. Implementing even a small distributed system with 54 worker nodes and 5 stragglers results in a serious loss of precision when trying to decode the the Polynomial code [69] (see empirical results presented in Fig. 4.3 on Page 19).

1.3 Thesis Contributions

This thesis proposes solutions to counter the numerical stability issues of polynomial based codes. This effort has resulted in the publication of two papers [55,57]. The thesis deals with the following problems in the area of coded computation

1. Collaborative Decoding of Polynomial Codes: It is shown in [69] that an (N, K) Polynomial code/Reed Solomon code can correct upto $\lfloor \frac{N-K}{2} \rfloor$ errors. We have proved that polynomial codes used for the matrix multiplication problem are Generalized Interleaved Reed Solomon codes and hence can be collaboratively decoded. This results in an increase in the decoding radius of upto $N - K - 1$ errors and re-

markably, we have empirically shown that the condition number of the underlying Vandermonde matrices decreases with increase in the number of codes that are collaboratively decoded.

2. Random Khatri-Rao Product Codes: We propose a simple class of codes based on Random linear codes as an alternative to polynomial codes. These codes have a higher degree of freedom since the choice of evaluation points is random and not constrained by any parameter. We prove that these codes are MDS (Maximum Distance Separable) with probability 1. These codes have been shown to be of several magnitudes better in terms of error in the reconstructed message [57].

3. Product Lagrange-Coded Computing: We propose a product coded variant of Lagrange codes [66]. For Polynomial codes with degree of the message upto 20, it is possible to find evaluation points [55] such that there is no error in the reconstruction of the message. But to the best of our knowledge there is no literature that can specify evaluation points for messages with a higher degree. We utilize this observation to break the message in 2 dimensions. This converts a message of degree 400 to a series of messages with degree $20 = \sqrt{400}$. This results in a huge gain in numerical stability.

1.4 Organization of the Thesis

The remainder of the thesis is organized as follows. Chapter 2 gives a short description of previous works in the field of coded computing. Chapter 3 describes the Collaborative decoding problem in the real field and its application to distributed matrix multiplication. Chapter 4 elaborates on the numerical instability issues in existing codes proposed for the distributed matrix multiplication problem and proposes a novel random coding scheme called Random-Khatri Rao product codes to alleviate the instability problem. Chapter 5 describes numerical instability issues of existing coding schemes for distributed polynomial computation and proposes product lagrange codes which are much stabler in terms

of reconstruction error. Finally, we conclude the thesis in chapter 6 and mention some open problems in the field of coded computation.

2. BACKGROUND

2.1 Distributed Matrix Multiplication problem

We consider the problem of computing $\mathbf{A}^\mathsf{T}\mathbf{B}$ for two matrices $\mathbf{A} \in \mathbb{F}^{s\times r}$ and $\mathbf{B} \in \mathbb{F}^{s\times r'}$ (for an arbitrary field $\mathbb{F}$) in a distributed fashion with N worker nodes using a coded matrix multiplication scheme [12, 13, 29, 30, 32, 33, 49, 63, 67, 68, 71]. To compute this distributed matrix multiplication operation, we assume that the matrices $\mathbf{A}$ and $\mathbf{B}$ are split into m subblocks and n subblocks, respectively. These subblocks are encoded for example, using a Polynomial code [68]. Each worker node performs a matrix multiplication and returns a matrix with a total of $L = \frac{rr'}{mn}$ elements (from $\mathbb{F}$) to the master node.

2.2 Polynomial based Codes for Distributed Matrix Multiplication

2.2.1 Notation

Throughout the thesis, we denote matrices by boldface capital letters, e.g., $\mathbf{A}$, and denote vectors by boldface small letters, e.g., $\mathbf{a}$. Occasionally, we use underlined variables to represent vectors, e.g., $\underline{a}$. For an integer $i \geq 1$, we denote $\{1, \ldots, i\}$ by $[i]$, and for two integers i and j such that $i < j$, we denote $\{i, i+1, \ldots, j\}$ by $[i, j]$. We use the short notation $((f(i, j))_{i\in[m],j\in[n]})$ to represent an $m\times n$ matrix whose entry (i, j) is $f(i, j)$, where $f(i, j)$ is a function of i and j. We occasionally use the compact notation $(\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_n)$ to represent an $m \times n$ matrix whose columns are the column-vectors $\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_n$, each of length m. Similarly, sometimes we use the compact notation $(\mathbf{a}_1; \mathbf{a}_2; \ldots; \mathbf{a}_m)$ to represent an $m \times n$ matrix whose rows are the row-vectors $\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_m$, each of length n. We also denote by $\mathbf{A}(i, :)$ and $\mathbf{A}(:, j)$ the ith row and the jth column of a matrix $\mathbf{A}$, respectively. If $\mathcal{S}_1 \subset \mathbb{Z}^+$ and $\mathcal{S}_2 \subset \mathbb{Z}^+$ are two subsets of positive integers, then the submatrix of $\mathbf{A}$ corresponding to the rows from $\mathcal{S}_1$ and columns from $\mathcal{S}_2$ is given by $[\mathbf{A}](\mathcal{S}_1, \mathcal{S}_2)$. We denote the set of integers from i to j, inclusive of i and j by $i : j$ and we denote the set of integers from 1 to i by $[i]$. Also, for a vector $\underline{v}$, we denote the part of vector $\underline{v}$ between indices i and j

as $\underline{v}_{i:j}$. We will assume that vectors without transposes are column vectors unless stated otherwise. Random variables will be denoted by capital letters and their realizations will be denoted by lower case letters. A vector or a matrix with a $\wedge$ above is an estimate.

2.2.2 Polynomial Codes

In this section, we review the Polynomial codes of Yu, Maddah-Ali and Avestimehr [67] for distributed matrix multiplication. Consider the problem of computing $\mathbf{A}^\mathsf{T}\mathbf{B}$ in a distributed fashion for two matrices $\mathbf{A} \in \mathbb{F}^{s\times r}$ and $\mathbf{B} \in \mathbb{F}^{s\times r'}$ for an arbitrary field $\mathbb{F}$. In the scheme of Polynomial codes in [67], the master node distributes the task of matrix multiplication among N worker nodes as follows.

The columns of $\mathbf{A}$ and $\mathbf{B}$ are first partitioned into m partitions $\mathbf{A}_0, \mathbf{A}_1, \ldots, \mathbf{A}_{m-1}$ of equal size $\frac{r}{m}$ and n partitions $\mathbf{B}_0, \mathbf{B}_1, \ldots, \mathbf{B}_{n-1}$ of equal size $\frac{r'}{n}$, respectively,

$$\mathbf{A} = [\mathbf{A}_0\ \mathbf{A}_1 \cdots \mathbf{A}_{m-1}],\ \ \mathbf{B} = [\mathbf{B}_0\ \mathbf{B}_1 \cdots \mathbf{B}_{n-1}].$$

Let $x_1, x_2, \ldots, x_N$ be N distinct elements in $\mathbb{F}$. For two parameters $\alpha, \beta \in [N]$, let $\tilde{\mathbf{A}}_i$ and $\tilde{\mathbf{B}}_i$ be matrices defined by,

$$\tilde{\mathbf{A}}_i = \sum_{j=0}^{m-1} \mathbf{A}_j x_i^{j\alpha},\ \ \tilde{\mathbf{B}}_i = \sum_{j=0}^{n-1} \mathbf{B}_j x_i^{j\beta}.$$

The dimensions of the matrices $\tilde{\mathbf{A}}_i$ and $\tilde{\mathbf{B}}_i$ are $s \times \frac{r}{m}$ and $s \times \frac{r'}{n}$, respectively.

The ith worker node computes the smaller matrix product $\tilde{\mathbf{C}}_i$ given the values of $\tilde{\mathbf{A}}_i$ and $\tilde{\mathbf{B}}_i$,

$$\tilde{\mathbf{C}}_i = \tilde{\mathbf{A}}_i^\mathsf{T}\tilde{\mathbf{B}}_i = \sum_{j=0}^{m-1}\sum_{k=0}^{n-1} \mathbf{A}_j^\mathsf{T}\mathbf{B}_k\, x_i^{j\alpha+k\beta}. \tag{2.1}$$

The parameters α and β are chosen carefully such that for each pair (j,k) the corresponding exponent of x_i (i.e., $j\alpha + k\beta$) is distinct. For instance, one such choice for α and β is $\alpha = 1$ and $\beta = m$. In this case, the ith worker node essentially evaluates $\mathbf{P}(x)$ at $x = x_i$

and returns $\mathbf{P}(x_i)$, where

$$\mathbf{P}(x) = \sum_{j=0}^{m-1} \sum_{k=0}^{n-1} \mathbf{A}_j^\mathsf{T} \mathbf{B}_k \, x^{j+km}. \tag{2.2}$$

The coefficients in the polynomial $\mathbf{P}(x)$ are the mn uncoded symbols of the product $\tilde{\mathbf{C}}_i$ in (2.1). The crux of the Polynomial code is that the vector of coded symbols $(\mathbf{P}(x_1), \ldots, \mathbf{P}(x_N)) = (\tilde{\mathbf{C}}_1, \tilde{\mathbf{C}}_2, \cdots, \tilde{\mathbf{C}}_N)$ can be considered as a codeword of a Reed-Solomon (RS) code. If N worker nodes are available in the distributed system, a Polynomial code essentially evaluates the polynomial $\mathbf{P}(x)$ at N points of the field $\mathbb{F}$; any mn of which can recover the coefficients which can be put together to recover the matrix product. The minimum number of worker nodes that need to compute and return the correct evaluations of $\mathbf{P}(x)$ for the master node to be able to successfully recover the matrix product $\mathbf{A}^\mathsf{T}\mathbf{B}$ is called the *recovery threshold*. Viewing the recovery process of a Polynomial code as a polynomial interpolation operation, it can be seen that the recovery threshold of the Polynomial code is mn [67].

2.2.3 OrthoPoly codes

One important drawback of Polynomial codes is that the process of recovering $\mathbf{A}^\mathsf{T}\mathbf{B}$ from the results of the worker nodes (the decoding process) involves explicitly or implicitly inverting a Vandermonde matrix, which is well known to be highly numerically unstable even for moderate values of $K \triangleq mn$. Very recently, Fahim and Cadambe [18] proposed a very interesting polynomial code called OrthoPoly code which uses an orthogonal polynomial basis resulting in a Chebyshev-Vandermonde structure for the generator matrix. OrthoPoly codes are also MDS codes, i.e., have optimal recovery threshold; however, they afford better numerical stability than Polynomial codes. In particular, when there are S stragglers among N nodes, i.e., $N = K + S$, the condition number of the matrix that needs to be inverted grows only polynomially in N.

In this section, we will briefly review OrthoPoly codes for the sake of completeness. Details can be found in [18]. The encoding scheme consists of the master node dividing

the matrices $\mathbf{A}$ and $\mathbf{B}$ as in Section 4.2 and subsequently computing

$$\tilde{\mathbf{A}}_i^\mathsf{T} = \sum_{j=1}^{m-1} T_j(x_i)\mathbf{A}_j^\mathsf{T}, \quad \tilde{\mathbf{B}}_i = \sum_{j=1}^{n-1} T_{jm}(x_i)\mathbf{B}_j$$

where $T_r(x) = \cos(r(\cos^{-1}(x)))$ and $x_i = \cos(\frac{(2i-1)\pi}{2N})$, and sending $\tilde{\mathbf{A}}_i^\mathsf{T}$ and $\tilde{\mathbf{B}}_i$ to the ith worker node. The ith worker node computes $\tilde{\mathbf{A}}_i^\mathsf{T}\tilde{\mathbf{B}}_i$ and sends the result back the master node for decoding. Let us assume that worker nodes $i = 1, \ldots, K$ (K is defined as $K \triangleq mn$) return their outputs. As before, we focus on the recovery of $[\mathbf{A}_i^\mathsf{T}\mathbf{B}_j](1,1)$. Let $y_i = [\tilde{\mathbf{A}}_i^\mathsf{T}\tilde{\mathbf{B}}_i](1,1)$ denote the $(1,1)$th entry in the matrix product computed by the ith non-straggler worker node and let $z_{j,l} = [\mathbf{A}_j^\mathsf{T}\mathbf{B}_l](1,1)$. For $j \in [mn]$, let $j' = \lceil j/n \rceil$ and $j'' = ((j-1) \mod n) + 1$ and let $w_j = z_{j',j''}$. The computed values y_i's are related to the unknown values w_j's according to

$$\underbrace{\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_i \\ \vdots \\ y_K \end{bmatrix}}_{\underline{y}} = \underbrace{\begin{bmatrix} T_0(x_1) & \cdots & T_{K-1}(x_1) \\ \vdots & \ddots & \vdots \\ T_0(x_K) & \cdots & T_{K-1}(x_K) \end{bmatrix}}_{\mathbf{G}_O} \mathbf{H} \underbrace{\begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_j \\ \vdots \\ w_K \end{bmatrix}}_{\underline{w}} \tag{2.3}$$

where $\mathbf{H}$ is a $K \times K$ matrix such that

$$\mathbf{H}((r,(i-1)+(j-1)m+1)) = \begin{cases} 1, & r=(i-1)+(j-1)m+1, \\ & i=1, j \in [n] \\ \frac{1}{2}, & r=(i-1)+(j-1)m+1, \\ & i \neq 1, i \in [m], j \in [n] \\ \frac{1}{2}, & r=|(i-1)-(j-1)m|+1, \\ & i \neq 1, i \in [m], j \in [n] \\ 0, & \text{otherwise.} \end{cases}$$

An estimate of $\underline{w}$ ($\underline{w}_i = [\mathbf{A}_j \mathbf{B}_l](1,1)$ such that $i = r + lm + 1$, for $0 \leq r \leq m-1$ and $0 \leq l \leq n-1$) is then obtained according to

$$\underline{\hat{w}} = \mathbf{H}^{-1} \mathbf{G}_O^{-1} \underline{y}. \tag{2.4}$$

2.3 Distributed Multivariate Polynomial Evaluation problem and Lagrange codes

Polynomial based codes, despite being unstable have been used to construct solutions to the Distributed Multivariate Polynomial Evaluation problem (DPME). One such solution would be the Lagrange codes [66]. In this section, we look at the DPME problem and the Lagrange code solution.

The goal is to compute K evaluations $\{f(\mathbf{X}_k)\}_{1 \leq k \leq K}$ using N workers where $\{\mathbf{X}_k\}_{1 \leq k \leq K}$ are K matrices, each of size $r \times d$ with entries from the real field, and f is a matrix function of the form $f(\mathbf{X}_k) = [f_{i,j}(\mathbf{X}_k)]_{1 \leq i \leq a, 1 \leq j \leq b}$ where $f_{i,j}(\mathbf{X}_k)$ is a multivariate polynomial whose variables are the entries of the matrix $\mathbf{X}_k$. We refer to this problem as the *Distributed Multivariate Polynomial Evaluation (DPME)*. We briefly explain the main ideas of the Lagrange Coded Computing (LCC) scheme of [66] via an example.

Consider $K = 2$ matrices $\mathbf{X}_1$ and $\mathbf{X}_2$ with real entries, each of size 3×2 (i.e., $r = 3$

and $d = 2$). Suppose that we wish to compute $f(\mathbf{X}_1)$ and $f(\mathbf{X}_2)$ distributedly using $N = 4$ workers among which at most $S = 1$ worker node is a straggler, where $f(\mathbf{X}_k) = \mathbf{X}_k^\mathsf{T}\mathbf{X}_k\mathbf{w}$ for $\mathbf{w} = [1, 1]^\mathsf{T}$. Note that $f : \mathbb{R}^{3\times 2} \to \mathbb{R}^{2\times 1}$ is a matrix function of the form

$$f(\mathbf{X}_k) = \begin{bmatrix} f_{1,1}(\mathbf{X}_k) \\ f_{2,1}(\mathbf{X}_k) \end{bmatrix}$$
$$= \begin{bmatrix} x_{1,1}^2 + x_{2,1}^2 + x_{3,1}^2 + x_{1,1}x_{1,2} + x_{2,1}x_{2,2} + x_{3,1}x_{3,2} \\ x_{1,1}x_{1,2} + x_{2,1}x_{2,2} + x_{3,1}x_{3,2} + x_{1,2}^2 + x_{2,2}^2 + x_{3,2}^2 \end{bmatrix}$$

where $x_{i,j}$ is the (i, j)th entry of the matrix $\mathbf{X}_k$. Note that $\deg(f) = 2$ because $f_{1,1}$ and $f_{2,1}$ have total degree 2.

First, the master node encodes $\mathbf{X}_1$ and $\mathbf{X}_2$ using a Lagrange interpolation polynomial as follows:

$$u(z) = \mathbf{X}_1 \cdot \frac{z - \beta_2}{\beta_1 - \beta_2} + \mathbf{X}_2 \cdot \frac{z - \beta_1}{\beta_2 - \beta_1},$$

where β_1, β_2 are $K = 2$ distinct elements from $\mathbb{R}$. Noting that $u(\beta_1) = \mathbf{X}_1$ and $u(\beta_2) = \mathbf{X}_2$, computing $f(\mathbf{X}_1)$ and $f(\mathbf{X}_2)$ is equivalent to computing $f(u(\beta_1))$ and $f(u(\beta_2))$, respectively. Let $\alpha_1, \ldots, \alpha_4$ be $N = 4$ distinct elements from $\mathbb{R}$. The master node requests the ith worker node to compute $f(u(\alpha_i))$.

From the construction, it is easy to see that $f(u(\alpha_i))$ is the evaluation of the composition polynomial $f(u(z))$ at $z = \alpha_i$. Since the degree of the polynomial $u(z)$ is $K-1 = 1$, the degree of the (univariate) polynomial $f(u(z))$ (in variable z) is at most $(K - 1)\deg(f) = 2$. Thus, the master node is able to recover the polynomial $f(u(z))$ from any $(K-1)\deg(f)+1 = 3$ out of $N = 4$ evaluations $f(u(\alpha_1)), \ldots, f(u(\alpha_4))$, using polynomial interpolation. Since the evaluations $f(u(\alpha_1)), \ldots, f(u(\alpha_4))$ are the results of the computations by the worker nodes, any $(K - 1)\deg(f) + 1 = 3$ worker nodes suffice for the master node to recover the polynomial $f(u(z))$. Followed by the recovery of the polynomial $f(u(z))$, the master node can readily recover $f(\mathbf{X}_1)$ and $f(\mathbf{X}_2)$ by evaluating $f(u(z))$ at $z = \beta_1$ and

$z = \beta_2$, respectively.

In general, the worst-case recovery threshold of LCC is $(K-1)\deg(f)+1$ [66]. It should be noted that the average-case recovery threshold of LCC is also the same.

3. COLLABORATIVE DECODING OF POLYNOMIAL CODES [1]

3.1 Introduction and Main Result

In this chapter, we consider the problem of computing $\mathbf{A}^\mathsf{T}\mathbf{B}$ for two matrices $\mathbf{A} \in \mathbb{F}^{s\times r}$ and $\mathbf{B} \in \mathbb{F}^{s\times r'}$ (for an arbitrary field $\mathbb{F}$)[2] in a distributed fashion with N worker nodes using a coded matrix multiplication scheme [12, 13, 29, 30, 32, 33, 49, 63, 67, 68, 71] To keep the presentation clear, we will focus on one class of codes, namely Polynomial codes, and explain our results in relation to the Polynomial codes [67]; notwithstanding, our results also apply to Entangled Polynomial codes [68] and PolyDot codes [12]. We assume that the matrices $\mathbf{A}$ and $\mathbf{B}$ are split into m subblocks and n subblocks, respectively. These subblocks are encoded using a Polynomial code [68]. Each worker node performs a matrix multiplication and returns a matrix with a total of $L = \frac{rr'}{mn}$ elements (from $\mathbb{F}$) to the master node.

Our main interest is in the fault-tolerant setup where some of the N worker nodes return erroneous values. We say that an error pattern of Hamming weight t has occurred if t worker nodes return matrices that contain some erroneous values. The main idea in the Polynomial codes, Entangled Polynomial codes and PolyDot codes is to encode the subblocks of $\mathbf{A}$ and $\mathbf{B}$ in a clever way such that the matrix product returned by the worker nodes are symbols of a codeword of a Reed-Solomon (RS) code over $\mathbb{F}$. The properties of an RS code are then used to obtain bounds on the error-correction capability of the scheme.

The main contribution of this work relies on the observation that Polynomial codes, Entangled Polynomial codes, and PolyDot codes are not just RS codes, but an Interleaved Reed-Solomon (IRS) code which consists of several RS codes that can be collaboratively decoded (see Section 3.2 or [52] for a formal definition). This additional structure provides

[1] The contents of this section have been presented in verbatim as part of the paper "Collaborative decoding of polynomial codes" at the Information Theory Workshop 2019 [55] for which I was the first author

[2] Some results in this chapter will apply to specific fields and this will be clarified later.

the opportunity for collaborative decoding of multiple RS codes involved in such coded matrix multiplication schemes. Such a collaborative decoding, for which efficient multi-sequence shift-register (MSSR) based decoding algorithms exist [26], provides a practical decoder with quadratic complexity in t, while potentially nearly doubling the decoding radius.

The main results of this chapter and their relation to the existing results are as follows. In [68], it is shown that any error pattern with Hamming weight t can be corrected if $t \leq \lfloor \frac{N-K}{2} \rfloor$ where $K = mn$ is the effective dimension of the Polynomial code. Very recently, Dutta *et al.* in [12] showed that when $\mathbb{F} = \mathbb{R}$ (the real field) and error values are randomly distributed according to a Gaussian distribution, with probability 1 all error patterns of Hamming weight $t \leq N - K - 1$ can be corrected. To attain this bound, [12] uses a decoding algorithm which is similar in spirit to exhaustive maximum likelihood decoding with a complexity that is $O\big(LN^{\min\{t,N-t\}}\big)$. This can be prohibitive for many practical values of N and t. In [12], it is suggested that in practice, the performance of ML decoding can be approximated by algorithms with polynomial complexity in N such as the ℓ_1-minimization algorithm [3]. However, there is no proof (nor evidence) that such algorithms can correct all error patterns of Hamming weight up to $N - K - 1$ with probability 1. Indeed, as we will show in this work, the standard ℓ_1-minimization based decoding algorithm [3] fails to correct all error patterns of Hamming weight up to $N - K - 1$ with a non-zero probability.

In this work, we show that we can use the MSSR decoding algorithm of [26] for decoding Polynomial codes with the complexity of $O\big(Lt^2 + N\big)$. For this algorithm, we will show that when $\mathbb{F} = \mathbb{F}_q$ (a finite field with q elements), for $\lfloor \frac{N-K}{2} \rfloor < t \leq \frac{L}{L+1}(N - K)$, all but a fraction $\gamma(t)$ of the error patterns of Hamming weight t can be corrected where $\gamma(t) \to 0$ as $q \to \infty$. In particular, the convergence of $\gamma(t)$ to zero is exponentially fast in L, i.e., $\gamma(t) = q^{-\Omega(L)}$, for $\lfloor \frac{N-K}{2} \rfloor < t \leq \frac{L}{L+1}(N - K)$. In addition, when $\mathbb{F} = \mathbb{R}$, by extending the results of [26] and [51] to the real field and using the results of [12], we will show that

for $L \geq N-K-1$ and $\lfloor \frac{N-K}{2} \rfloor < t \leq N-K-1$, all error patterns of Hamming weight t can be corrected with probability 1, under the random Gaussian error model previously considered in [12].

In a nutshell, our results show that with a probability arbitrarily close to 1 (or respectively, with probability 1), all error patterns of Hamming weight up to $\frac{L}{L+1}(N-K)$, which can be made arbitrarily close to $N-K-1$ for sufficiently large L, can be corrected for sufficiently large finite fields (or respectively, the real field). Not only does this indicate a substantial increase in the error-correction radius with provable guarantees when compared to the results in [68], but it also shows that the Dutta *et al.*'s upper bound in [12] can be achieved with a practical decoder with a quadratic complexity in the number of faulty worker nodes (t). This improvement in complexity is the result of collaboratively decoding the IRS code instead of separately decoding the RS codes using a maximum likelihood decoder as is done in [12].

3.2 Polynomial Codes are Interleaved Reed-Solomon Codes

Definition 1. *Generalized Reed-Solomon (RS) Codes: Let* $\mathbf{m} = (m_0, m_1, \ldots, m_{K-1})$ *and let the associated polynomial* $m(x)$ *be defined as* $m(x) := m_0 + m_1 x + \ldots + m_{K-1}x^{K-1}$. *Further, let* $\mathbf{c} = (c_0, c_1, \ldots, c_{N-1})$, $\boldsymbol{\alpha} = (\alpha_0, \alpha_1, \ldots, \alpha_{N-1})$ *and* $\mathbf{v} = (v_0, v_1, \ldots, v_{N-1})$ *be three row vectors such that* $c_i, \alpha_i, v_i \in \mathbb{F}$, $v_i \neq 0$, *and* $\alpha_i \neq \alpha_j$. *A Generalized Reed-Solomon (GRS) code* $\mathcal{C}$ *over* $\mathbb{F}$ *of length* N, *dimension* K, *evaluation points* $\boldsymbol{\alpha}$, *weight vectors* $\mathbf{v}$, *denoted by* $\text{GRS}(\mathbb{F}, N, K, \boldsymbol{\alpha}, \mathbf{v})$, *is the set of all row-vectors (codewords)* $\mathbf{c} = (v_0 m(\alpha_0), v_1 m(\alpha_1), \ldots, v_{N-1} m(\alpha_{N-1}))$, *i.e.,* $c_i = v_i m(\alpha_i)$. *Equivalently, a GRS code is also the set of codewords* $\mathbf{c}$ *such that for all* $i \in [0, N-K-1]$, $\sum_{j=0}^{N-1} u_j c_j (\alpha_i)^j = 0$, *where* $u_i^{-1} = v_i \prod_{j \neq i} (\alpha_i - \alpha_j)$. *The minimum distance of such a GRS code is* $d_{\min} = N-K+1$.

Reed-Solomon (RS) codes are a special case of GRS codes with $v_i = 1, u_i = 1, \forall i \in [0, N-1]$. For finite fields and the complex field, an $\boldsymbol{\alpha}$ exists such that $v_i = 1$ and $u_i = 1$, $i \in [0, N-1]$. However for the real field, u_i and v_i cannot be simultaneously set to 1 and, hence, it is required to consider GRS codes.

Definition 2. *Interleaved Generalized Reed-Solomon (IRS) Codes [52]: Let* $\{\mathcal{C}^{(l)} = \mathrm{RS}(\mathbb{F}, N, K^{(l)}, \boldsymbol{\alpha}, \boldsymbol{u})\}_{l\in[L]}$ *be a collection of* L *GRS codes, each of length* N *over a field* $\mathbb{F}$*, where the dimension and minimum distance of the* l*th RS code are* $K^{(l)}$ *and* $d^{(l)}$*, respectively. Then, an Interleaved Reed-Solomon (IGRS) code* $\mathcal{C}_{\mathrm{IGRS}}$ *is the set of all* $L \times N$ *matrices* $(\mathbf{c}^{(1)}; \mathbf{c}^{(2)}; \ldots; \mathbf{c}^{(L)})$ *where* $\mathbf{c}^{(l)} \in \mathcal{C}^{(l)}$ *for* $l \in [L]$ *[26]. If all the* L *GRS codes* $\mathcal{C}^{(l)}$ *are equivalent, i.e.,* $\mathcal{C}^{(l)} = \mathcal{C}$ *for all* $l \in [L]$*, the IGRS code* $\mathcal{C}_{\mathrm{IRS}}$ *is called* homogeneous*.*

The chief observation in this work is that the Polynomial codes, Entangled Polynomial codes, and PolyDot codes are IGRS codes. Here, we formally prove this observation for the Polynomial codes. We shall henceforth refer to GRS codes as RS codes.

Theorem 3. *A Polynomial code is an IGRS code.*

Proof: Let $\mathbf{W}$ be an $a \times b$ matrix with entries from $\mathbb{F}$, and let $\Gamma : \mathbb{F}^{a\times b} \to \mathbb{F}^{ab}$ denote a vectorizing operator which reshapes a matrix $\mathbf{W}$ into a column-vector $\mathbf{w} = (w_1, \ldots, w_{ab})^{\mathsf{T}}$, i.e., $\Gamma(\mathbf{W}) = \mathbf{w}$, such that $w_{(i-1)b+j} = \mathbf{W}(i, j)$, where $\mathbf{W}(i, j)$ is the element (i, j) of $\mathbf{W}$.

Let $\tilde{\mathbf{C}}_i(p, q)$ be the element (p, q) of the matrix $\tilde{\mathbf{C}}_i$,

$$\tilde{\mathbf{C}}_i(p, q) \triangleq \sum_{j=0}^{m-1} \sum_{k=0}^{n-1} [\mathbf{A}_j^{\mathsf{T}} \mathbf{B}_k](p, q) x_i^{j+km}. \tag{3.1}$$

Consider the $\frac{rr'}{mn} \times N$ matrix $\mathbf{D} \triangleq (\Gamma(\tilde{\mathbf{C}}_1), \Gamma(\tilde{\mathbf{C}}_2), \ldots, \Gamma(\tilde{\mathbf{C}}_N))$, where the ith column of $\mathbf{D}$, namely $\Gamma(\tilde{\mathbf{C}}_i)$, is obtained by applying the vectorizing operator Γ to $\tilde{\mathbf{C}}_i$. Let (p_i, q_i) be the unique pair (p, q) such that $i = (p - 1)\frac{r'}{n} + q$. Then, the element (i, j) of $\mathbf{D}$ is $\tilde{\mathbf{C}}_j(p_i, q_i)$, and accordingly, the ith row of $\mathbf{D}$ is given by $[\tilde{\mathbf{C}}_1(p_i, q_i), \tilde{\mathbf{C}}_2(p_i, q_i), \ldots, \tilde{\mathbf{C}}_N(p_i, q_i)]$, which is a codeword of an RS code. Thus the matrix $\mathbf{D}$ is a codeword of an IRS code with $L = \frac{rr'}{mn}$. In particular, the ith worker node computes $\tilde{\mathbf{C}}_i$ that has dimension $\frac{r}{m} \times \frac{r'}{n}$. It is evident from (3.1) that the element (p, q) of $\tilde{\mathbf{C}}_i$ is the message polynomial $\sum_{j=0}^{m-1} \sum_{k=0}^{n-1} [\mathbf{A}_j^{\mathsf{T}} \mathbf{B}_k](p, q) x^{j+km}$ evaluated at x_i. Thus, $\tilde{\mathbf{C}}_i$ contains $\frac{rr'}{mn}$ RS codes evaluated at x_i by the ith worker node. That is, the computations returned by the ith worker node constitute the ith column of an

IGRS code with N being the number of worker nodes and $L = \frac{rr'}{mn}$ being the number of RS codes. This shows that a Polynomial code is a homogeneous IGRS code with $K^{(l)} = mn$ for $l \in [L]$. $\square$

3.2.1 Error Matrix and Error Models

We consider the case when the worker nodes introduce additive errors in their computation. Let $\mathbf{E}_i \in \mathbb{F}^{\frac{r}{m} \times \frac{r'}{n}}$ denote the error matrix introduced by the ith worker node. Then the master node receives the set of matrices $\tilde{\mathbf{R}}_i$, for $i \in [N]$ where $\tilde{\mathbf{R}}_i = \tilde{\mathbf{C}}_i \oplus \tilde{\mathbf{E}}_i$. Let $\mathbf{R}$ be the $\frac{rr'}{mn} \times N$ matrix of values received by the master node where the ith column of $\mathbf{R}$ is given by $\Gamma(\tilde{\mathbf{R}}_i)$, and let $\mathbf{E}$, referred to as the *error matrix*, be the $\frac{rr'}{mn} \times N$ matrix of error values where the ith column of $\mathbf{E}$ is given by $\Gamma(\tilde{\mathbf{E}}_i)$. Then, $\mathbf{R} = \mathbf{D} \oplus \mathbf{E}$ where $\mathbf{D}$ is a codeword of an IRS code. If the ith worker node returns erroneous values, then the ith column of $\mathbf{R}$ will contain errors. Thus, the original problem of fault-tolerant distributed matrix multiplication reduces to the problem of decoding $\mathbf{D}$ from $\mathbf{R}$.

Definition 4. *The Hamming weight of the matrix* $\mathbf{E}$ *denoted by* $W_H(\mathbf{E})$ *is defined as the number of non-zero columns in* $\mathbf{E}$*.*

We consider two different error models. First, we consider the Uniform Random Error for Finite Fields (UREF) model where the non-zero columns of the error matrix $\mathbf{E}$ are assumed to be uniformly distributed over all the non-zero vectors in $\mathbb{F}_q^L$ for a finite field $\mathbb{F}_q$. We further extend this model to the real field $\mathbb{R}$ where each non-zero entry in the error matrix $\mathbf{E}$ is assumed to be an independently and identically distributed Gaussian random variable (with arbitrary mean and variance). This model is referred to as the Gaussian Random Error (GRE) model.

3.2.2 Decoding and Error Events

Let $\psi : \mathbb{F}^{L \times N} \to \{\mathcal{C}_{\text{IRS}}, F\}$ be the decoding function, where F is a symbol that denotes decoding failure. A *decoding error* is said to have occurred if $\psi(\mathbf{R}) \neq \mathbf{D}$. An *undetected*

decoding error is said to have occurred if $\psi(\mathbf{R}) \neq \mathbf{D}$ and $\psi(\mathbf{R}) \neq F$, whereas a *decoding failure* is said to have occurred if $\psi(\mathbf{R}) = F$.

3.3 Collaborative Decoding of Interleaved Reed-Solomon Codes

Simultaneous decoding of all the RS codes in an IRS code is known as *collaborative decoding*. As shown in [52] and [26], collaborative decoding of IRS codes has certain advantages. In particular, when burst errors occur, they occur on the same column of the IRS code. Hence, multiple RS codewords share the same error positions. Note that an IRS code is actually a set of RS codes stacked together, each of which yields a set of syndrome equations. Intuitively, when burst errors occur, the error locator polynomials are more or less the same for all the RS codes but the number of syndrome equations increases with the number of stacked RS codes. This implies that a much larger set of errors can be corrected. This is because the rank of the stacked syndrome matrix is greater than or equal to the rank of the individual syndrome matrices, thus giving rise to the possibility of a greater decoding radius than the unique decoding bound of $\frac{1-R}{2}$, where R is the code rate. More specifically, it was shown by Schmidt *et al.* in [52] that when a set of L RS codes are collaboratively decoded, except for a small probability of failure and a small probability of error (discussed in Section 3.5), the fraction of errors that can be corrected can be as large as $\frac{L}{L+1}(1-R)$.

3.4 Decoding Algorithms

3.4.1 Collaborative Peterson's Algorithm

In this section, we propose a collaborative version of the Peterson's algorithm [41] to correct up to $t \leq t_{\max} \triangleq \frac{L}{L+1}(N-K)$ errors.

Consider t non-zero errors in columns $j_1, j_2, \ldots, j_t$ of the matrix $\mathbf{R}$ (i.e., the indices of the non-zero columns of the error matrix $\mathbf{E}$ are $j_1, j_2, \ldots, j_t$). Let $r^{(l)}(z) \triangleq \sum_{j=0}^{N-1} u_j \mathbf{R}(l,j) z^{j-1}$ be the modified (multiplying component-wise by u_j) received polynomial for the lth RS code, where $\mathbf{R}(l,j)$ is the element (l,j) of the matrix $\mathbf{R}$. Then, the syndrome sequence

for the lth RS code is given by $S^{(l)} \triangleq \{S_i^{(l)}\}_{i=0}^{N-K-1}$, where $S_i^{(l)} \triangleq \sum_{j=0}^{N-1} u_j \mathbf{R}(l,j)\alpha_j^i$ for $i \in [0, N-K-1]$. Define the error locator polynomial $\Lambda(z)$ as

$$\Lambda(z) \triangleq \prod_{i=1}^{t}(1 - z\alpha_{j_i}) = 1 + \lambda_1 z + \cdots + \lambda_t z^t$$

and let $\boldsymbol{\lambda}(t) = (\lambda_t, \lambda_{t-1}, \ldots, \lambda_1)^{\mathsf{T}}$ be the error locator vector associated with the error locator polynomial $\Lambda(z)$. When t errors occur $\Lambda(z)$ has a degree of t. The syndrome matrix $\mathbf{S}^{(l)}(t)$ and a vector $\mathbf{a}^{(l)}(t)$ for the lth RS code are given by

$$\mathbf{S}^{(l)}(t) \triangleq \begin{pmatrix} S_0^{(l)} & S_1^{(l)} & \cdots & S_{t-1}^{(l)} \\ S_1^{(l)} & S_2^{(l)} & \cdots & S_t^{(l)} \\ \vdots & \vdots & & \vdots \\ S_{N-K-t-1}^{(l)} & S_{N-K-t}^{(l)} & \cdots & S_{N-K-2}^{(l)} \end{pmatrix}, \ \mathbf{a}^{(l)}(t) \triangleq \begin{pmatrix} -S_t^{(l)} \\ -S_{t+1}^{(l)} \\ \vdots \\ -S_{N-K-1}^{(l)} \end{pmatrix} \tag{3.2}$$

Now we can write the following consistent linear system of equations for the IRS code,

$$\underbrace{\begin{pmatrix} \mathbf{S}^{(1)}(t) \\ \mathbf{S}^{(2)}(t) \\ \vdots \\ \mathbf{S}^{(L)}(t) \end{pmatrix}}_{\mathbf{S}_L(t)} \underbrace{\begin{pmatrix} \lambda_t \\ \lambda_{t-1} \\ \vdots \\ \lambda_1 \end{pmatrix}}_{\boldsymbol{\lambda}(t)} = \underbrace{\begin{pmatrix} \mathbf{a}^{(1)}(t) \\ \mathbf{a}^{(2)}(t) \\ \vdots \\ \mathbf{a}^{(L)}(t) \end{pmatrix}}_{\mathbf{a}_L(t)} \tag{3.3}$$

where $\mathbf{S}_L(t)$, the syndrome matrix for the IRS code, is the stacked matrix of $\mathbf{S}^{(l)}(t)$ for $l \in [L]$, and $\mathbf{a}_L(t)$, a vector for the IRS code, is the stacked vector of $\mathbf{a}^{(l)}(t)$ for $l \in [L]$. If t columns of the matrix $\mathbf{R}$ are in error, then the error locator vector $\boldsymbol{\lambda}(t)$ can be obtained by the collaborative Peterson's algorithm, described in Algorithm 1. The complexity of computing the rank of $\mathrm{rank}(\mathbf{S}_L(\tau))$ is $O(L\tau^3)$; computing $\hat{\boldsymbol{\lambda}}$ requires $O(\tau^3)$ operations if the structure of $\mathbf{S}_L(\tau)$ is not exploited, and the Chien search has a complexity of $O(N)$. Since we have to consider all values of $\tau \in [t_{\max}]$, the overall complexity is $O(Lt_{\max}^4 + N)$.

Definition 5. *(t-valid polynomial $\Lambda(z)$): A polynomial $\Lambda(z)$ over $\mathbb{F}$ is called t-valid if it is a polynomial of degree t and possesses exactly t distinct roots in $\mathbb{F}$.*

Algorithm 1 Collaborative Peterson's algorithm for IRS Decoding

Input: $S^{(l)} = \{S_i^{(l)}\}_{i=0}^{N-K-1} \; \forall l \in [L]$
Output: $\hat{\mathbf{D}} \in \{\mathbb{F}^{L\times N}, F \text{ (decoding failure)}\}$

$\hat{\mathbf{D}} = F$
if $\mathbf{S}_L(t) = \mathbf{0}$ **then**
  $\hat{\mathbf{D}} = \mathbf{R}$
else
  for each t from 1 to $t_{\max}$ **do**
    if $\mathrm{rank}(\mathbf{S}_L^{\mathsf{T}}(t)\mathbf{S}_L(t)) = t$ **then**
      $\hat{\boldsymbol{\lambda}} = (\mathbf{S}_L^{\mathsf{T}}(t)\mathbf{S}_L(t))^{-1}\mathbf{S}_L^{\mathsf{T}}(t))\mathbf{a}_L(t)$
      if $\mathbf{S}_L(t)\,\hat{\boldsymbol{\lambda}} = \mathbf{a}_L(t)$ **then**
        $(\hat{\lambda}_t, \hat{\lambda}_{t-1}, \ldots, \hat{\lambda}_1) = \hat{\boldsymbol{\lambda}}^{\mathsf{T}}$
        $\hat{\Lambda}(z) = 1 + \hat{\lambda}_1 z + \cdots + \hat{\lambda}_t z^t$
        if $\hat{\Lambda}(z)$ is t-valid **then**
          Compute error locations $\hat{j}_i, \hat{j}_2, \ldots, \hat{j}_t$ using a Chien search [41]
          for each l from 1 to L **do**
            From $\hat{j}_1, \ldots, \hat{j}_t$, and $S^{(l)}$, compute $\hat{\mathbf{E}}(l,:)$ using Forney's algorithm [41]
            Compute $\hat{\mathbf{D}}(l,:) = \mathbf{R}(l,:) - \hat{\mathbf{E}}(l,:)$
          end for
        end if
      end if
    end if
  end for
end if

3.4.2 Multiple Sequence Shift Register algorithm

A more computationally efficient decoding algorithm to achieve error correction up to $t \leq t_{\max} = \frac{L}{L+1}(N-K)$ is the Multiple Sequence Shift Register (MSSR) algorithm proposed by Schmidt *et al.* in [51]. This algorithm has a complexity of $O(Lt^2 + N)$. The MSSR algorithm, reviewed here for completeness, is described in Algorithm 2.

Algorithm 2 Collaborative IRS Decoder (Schmidt *et. al* [52])

Input: $S^{(l)} = \{S_i^{(l)}\}_{i=0}^{N-K-1} \; \forall l \in [L]$
Output: $\hat{\mathbf{D}} \in \{\mathbb{F}^{L\times N}, F \text{ (decoding failure)}\}$

```
# Synthesize t and Λ̂(z) using the shift register synthesis algorithm in [51]
[t, Λ̂(z)] = Shift Register Synthesis Algorithm(S^(1), ..., S^(L))
D̂ = F
if t ≤ t_max and Λ̂(z) is t-valid then
    for each l from 1 to L do
        From Λ̂(z) compute Ê(l, :)
        Compute D̂(l, :) = R̂(l, :) − Ê(l, :)
    end for
end if
```

It can be seen that in the absence of numerical round-off errors, the outputs of the collaborative Peterson's algorithm and the MSSR algorithm are identical for every $\mathbf{R}$ since both of them compute the solution to (3.3).

3.5 Analysis of probability of failure and error for finite fields ($\mathbb{F} = \mathbb{F}_q$)

In Section 3.2, we showed that Polynomial codes are IRS codes. Hence the fault tolerance of the Polynomial codes can be analyzed using similar techniques for IRS codes. In this section, we consider the uniformly random error model for finite fields (UREF), defined in Section 3.2.1, which was originally considered in [52]. In particular, we define the error events

$$\begin{aligned}
&\mathcal{E}_1(t) = \{\mathbf{E} : W_H(\mathbf{E}) = t \text{ and the MSSR/collaborative algorithm fails}\},\\
&\mathcal{E}_2(t) = \{\mathbf{E} : W_H(\mathbf{E}) = t \text{ and the MSSR/collaborative algorithm makes an undetected error}\},\\
&\mathcal{E}(t) = \{\mathbf{E} : W_H(\mathbf{E}) = t\}.
\end{aligned} \tag{3.4}$$

Since the outputs of the collaborative Peterson's algorithm and the MSSR algorithm are identical for every $\mathbf{R}$, both algorithms have the same probability of failure and the same probability of undetected error. We denote by $P_F(t)$ and $P_{\mathrm{ML}}(t)$ the probability of failure and the probability of undetected error, respectively, given that $W_H(\mathbf{E}) = t$. Under

the UREF model, $P_F(t)$ and $P_{\mathrm{ML}}(t)$ are given by [52]

$$P_F(t) = \frac{|\mathcal{E}_1(t)|}{|\mathcal{E}(t)|},\ P_{\mathrm{ML}}(t) = \frac{|\mathcal{E}_2(t)|}{|\mathcal{E}(t)|}.$$

3.5.1 Probability of Failure

A necessary condition for the failure of both the collaborative Peterson's algorithm and the MSSR algorithm is that the matrix $\mathbf{S}_L(t)$ is not full rank, as shown in [52]. To calculate an upper bound on $P_F(t)$, we refer to the analysis by schmidt *et al.* in [52], and recall the following result from [52].

Theorem 6. *[52, Theorem 7] Under the UREF model, for all* $t \leq t_{\max} = \frac{L}{L+1}(N-K)$,

$$P_F(t) \leq \left(\frac{q^L - \frac{1}{q}}{q^L - 1}\right) \frac{q^{-(L+1)(t_{\max}-t)}}{q-1}. \tag{3.5}$$

By the result of Theorem 6, it can be readily seen that for all $t < t_{\max}$, $P_F(t)$ diminishes as $q^{-\Omega\left(L\right)}$ and for $t = t_{\max}$, $P_F(t)$ decays as q^{-1} .

3.5.2 Probability of Undetected Error

As shown in [52, Theorem 5], the MSSR algorithm has the Maximum Likelihood (ML) certificate property, i.e., whenever the decoder of [51] does not fail, it yields the ML solution, namely the codeword at minimum Hamming distance from the received word. The collaborative Peterson's algorithm has the same ML certificate property as well. An error matrix $\mathbf{E}$ with $W_H(\mathbf{E}) = t$ is said to be a *bad error matrix of Hamming weight* t if there exists a non-zero codeword $\mathbf{D} \in \mathcal{C}_{\mathrm{IRS}}$ such that $W_H(\mathbf{D} \ominus \mathbf{E}) \leq t$.

We now use a result from [20, Page 141] without proof.

Lemma 7. *[20, Page 141] Let* $\mathcal{C} \subseteq \{0, 1, \cdots q-1\}^N$ *be a code with relative distance* $\delta = d_{\min}/N$, *and let* $S \subseteq [N]$ *be such that* $|S| = (1-\gamma)N$, *where* $0 < \gamma \leq \delta - \varepsilon$ *for some* $\varepsilon > 0$. *Let* $\mathcal{E}_S$ *be the set of all error vectors with support* S^c, *and let* $\mathcal{B}_S$ *be the set of all bad error vectors with support*

S^c. Then,

$$|\mathcal{B}_S| \leq q^{\frac{N}{\log_2 q} - \frac{\varepsilon N}{2} + \frac{1}{2}} |\mathcal{E}_S|.$$

Theorem 8. *Under the UREF model, for all $t \leq N - K - 1$ (and in particular, for all $t \leq t_{\max} = \frac{L}{L+1}(N-K)$), $P_{\mathrm{ML}}(t) \to 0$ as $q^L \to \infty$.*

Proof: It is easy to see that an IRS code can be viewed as a single code over $\mathbb{F}_{q^L}$, i.e. $\mathcal{C}_{\mathrm{IRS}}$ is a $\left(\mathbb{F}_{q^L}, N, K, N-K+1\right)$ code. Lemma 7 holds for a single code and, hence, can be applied to $\mathcal{C}_{\mathrm{IRS}}$ with q being replaced by q^L. Since the upper bound in Lemma 7 depends only on the cardinality of $\mathcal{E}_S$, it follows that the probability of having a bad error matrix with $W_H(\mathbf{E}) = t$ for the $\left(\mathbb{F}_{q^L}, N, K, N-K+1\right)$ code (replacing q by q^L since $\mathcal{C}_{\mathrm{IRS}}$ is over q^L) which we denote by $P_e(t)$ is upper bounded by

$$P_e(t) = \frac{|\mathcal{B}_S|}{|\mathcal{E}_S|} \leq q^{L(\frac{N}{\log_2 q^L} - \frac{\varepsilon N}{2} + \frac{1}{2})}. \tag{3.6}$$

By setting $\delta = \frac{N-K+1}{N}$ and $\varepsilon = \frac{2}{N}$, it is easy to see that $P_e(t) \to 0$ as $q^L \to \infty$. For this choice of δ and ε, it follows that $\gamma \leq \delta - \varepsilon = \frac{N-K-1}{N}$, which implies that (3.6) holds for all $t \leq N - K - 1$.

Note that the algorithms in Section 3.4 have the ML certificate property. Note, also, that the fraction of error matrices that give rise to an undetected error is upper bounded by the fraction of bad error matrices. This is simply because without a bad error matrix of Hamming weight up to $(\delta - \varepsilon)N$, an undetected error cannot occur. Thus, $P_{\mathrm{ML}}(t) \leq P_e(t)$. Since $P_e(t)$ vanishes as $q^L \to \infty$, then $P_{\mathrm{ML}}(t)$ vanishes as $q^L \to \infty$. Moreover, N and K are fixed and finite, and hence, $\sum_{t=1}^{N-K-1} P_{\mathrm{ML}}(t) \to 0$ as $q^L \to \infty$. □

3.6 Analysis of probablity of failure and probability of error for the real field

In this section, we analyze the probability of failure and probability of error under the GRE model when the computations are performed over the real field. In particular, we consider the case that the error values are independently and identically distributed

standard Gaussian random variables (with zero mean and unit variance). Note, however, that this assumption does not limit the generality of the results, and is made for the ease of exposition only. For this model, conditioned on t errors occurring, the probability of failure ($P_F(t)$) and the probability of undetected error ($P_{\mathrm{ML}}(t)$) are given by

$$P_F(t) = \frac{\int_{\mathcal{E}_1(t)} \phi(\mathbf{x})\, d\mathbf{x}}{\int_{\mathcal{E}(t)} \phi(\mathbf{x})\, d\mathbf{x}}, \quad P_{\mathrm{ML}}(t) = \frac{\int_{\mathcal{E}_2(t)} \phi(\mathbf{x})\, d\mathbf{x}}{\int_{\mathcal{E}(t)} \phi(\mathbf{x})\, d\mathbf{x}},$$

where $\mathcal{E}_1(t), \mathcal{E}_2(t), \mathcal{E}(t)$ are defined as in (3.4), and $\phi(\mathbf{x})$ is the probability density function of an Lt-dimensional standard Gaussian random vector (with zero-mean vector and identity covariance matrix).

3.6.1 Probability of Failure

It should be noted that the results of [52] for finite fields cannot be directly extended to the real field, simply because the counting arguments used in [52] for finite fields do not carry over to the real field. In this section, we propose a new approach to derive the probability of failure for the real field case.

For simplifying the notation, hereafter, we use $\rho \triangleq N - K - t$. Suppose that $t \leq t_{\max} = \frac{L}{L+1}(N-K)$ errors occur at positions $j_1, j_2, \cdots, j_t$ with values $e_{j_1}^{(l)}, e_{j_2}^{(l)}, \cdots, e_{j_t}^{(l)}$ for the lth RS code. Recall the syndrome matrix $\mathbf{S}^{(l)}(t)$ for the lth RS code (see (3.2)). As shown in [52], $\mathbf{S}^{(l)}(t)$ can be decomposed as

$$\mathbf{S}^{(l)}(t) = \mathbf{H}^{(l)}(t) \cdot \mathbf{F}^{(l)}(t) \cdot \mathbf{D}(t) \cdot \mathbf{Y}(t),$$

where $\mathbf{H}^{(l)}(t) \triangleq (\alpha_{j_k}^{(i-1)})_{i\in[\rho],k\in[t]}$ is an $\rho \times t$ matrix, $\mathbf{F}^{(l)}(t) \triangleq \mathrm{diag}((e_{j_i}^{(l)})_{i\in[t]})$ is a $t \times t$ diagonal matrix, $\mathbf{D}(t) \triangleq \mathrm{diag}((\alpha_{j_i})_{i\in[t]})$ is a $t \times t$ diagonal matrix, and $\mathbf{Y}(t) \triangleq (\alpha_{j_i}^{(k-1)})_{i\in[t],k\in[t]}$ is a $t \times t$ matrix.

Theorem 9. *Under the GRE model, for all $t \leq t_{\max} = \frac{L}{L+1}(N-K)$, $P_F(t) = 0$. In particular, for $L \geq N - K - 1$, for all $t \leq N - K - 1$, $P_F(t) = 0$.*

Proof: The decoding algorithms described in Section 3.4 fail when the stacked matrix $\mathbf{S}_L(t)$ defined in (3.3) is rank deficient, i.e., there exists a non-zero row-vector $\mathbf{v}$ such that $\mathbf{S}_L(t) \cdot \mathbf{v}^{\mathsf{T}} = 0$. Alternatively, $\mathbf{S}_L(t)$ is rank deficient iff there exists a non-zero row-vector $\mathbf{v}$ such that

$$\mathbf{S}^{(l)}(t) \cdot \mathbf{v}^{\mathsf{T}} = (\mathbf{H}^{(l)}(t) \cdot \mathbf{F}^{(l)}(t) \cdot \mathbf{D}(t) \cdot \mathbf{Y}(t)) \cdot \mathbf{v}^{\mathsf{T}} = 0 \quad \forall l \in [L]. \tag{3.7}$$

Since $\mathbf{D}(t)$ and $\mathbf{Y}(t)$ are invertible, the condition (3.7) holds iff there is a non-zero row-vector $\mathbf{v}$ such that

$$(\mathbf{H}^{(l)}(t) \cdot \mathbf{F}^{(l)}(t)) \cdot \mathbf{v}^{\mathsf{T}} = 0 \quad \forall l \in [L]. \tag{3.8}$$

Let $\mathbf{v} = (v_1, v_2, \ldots, v_t)$, and let $f_{i,l} \triangleq e_{j_i}^{(l)}$ for all $i \in [t]$. Expanding (3.8), it is easy to see that

$$\underbrace{\begin{pmatrix} v_1 & v_2 & \cdots & v_t \\ v_1 \cdot \alpha_{j_1} & v_2 \cdot \alpha_{j_2} & \cdots & v_t \cdot \alpha_{j_t} \\ v_1 \cdot \alpha_{j_1}^2 & v_2 \cdot \alpha_{j_2}^2 & \cdots & v_t \cdot \alpha_{j_t}^2 \\ \vdots & \vdots & & \vdots \\ v_1 \cdot \alpha_{j_1}^{(\rho-1)} & v_2 \cdot \alpha_{j_2}^{(\rho-1)} & \cdots & v_t \cdot \alpha_{j_t}^{(\rho-1)} \end{pmatrix}}_{\mathbf{H}} \underbrace{\begin{pmatrix} f_{1,l} \\ f_{2,l} \\ \vdots \\ f_{t,l} \end{pmatrix}}_{\mathbf{f}^{(l)}} = 0. \tag{3.9}$$

Combining the condition (3.9) for all the RS codes in the IRS code (for all $l \in [L]$), it holds that

$$\mathbf{H} \cdot \mathbf{F} = 0, \tag{3.10}$$

where $\mathbf{H}$ is defined in (3.9), and $\mathbf{F} \triangleq (\mathbf{f}^{(1)}, \mathbf{f}^{(2)}, \ldots, \mathbf{f}^{(L)})$ is a $t \times L$ matrix where $\mathbf{f}^{(l)}$ for $l \in [L]$ is defined in (3.9). Alternatively, (3.10) can be written as

$$\mathbf{v} \cdot \mathbf{\Phi} = 0, \tag{3.11}$$

where $\mathbf{\Phi}$ is a $t \times \rho L$ matrix given by

$$\mathbf{\Phi} \triangleq \begin{pmatrix} f_{1,1} & \cdots & f_{1,L} & (\alpha_{j_1} f_{1,1}) & \cdots & (\alpha_{j_1} f_{1,L}) & \cdots & (\alpha_{j_1}^{(\rho-1)} f_{1,1}) & \cdots & (\alpha_{j_1}^{(\rho-1)} f_{1,L}) \\ f_{2,1} & \cdots & f_{2,L} & (\alpha_{j_2} f_{2,1}) & \cdots & (\alpha_{j_2} f_{2,L}) & \cdots & (\alpha_{j_2}^{(\rho-1)} f_{2,1}) & \cdots & (\alpha_{j_2}^{(\rho-1)} f_{2,L}) \\ \vdots & & \vdots & \vdots & & \vdots & & \vdots & & \vdots \\ f_{t,1} & \cdots & f_{t,L} & (\alpha_{j_t} f_{t,1}) & \cdots & (\alpha_{j_t} f_{t,L}) & \cdots & (\alpha_{j_t}^{(\rho-1)} f_{t,1}) & \cdots & (\alpha_{j_t}^{(\rho-1)} f_{t,L}) \end{pmatrix}. \tag{3.12}$$

Let $\mathcal{F}$ be the set of all $t \times L$ matrices $\mathbf{F} = (f_{i,l})_{i\in[t],l\in[L]}$ for each of which the condition (3.10) holds for some non-zero vector $\mathbf{v}$. We need to show that $\mathcal{F}$ is a set of measure zero.

We consider two cases as follows: (i) $t \leq L$, and (ii) $t > L$.

Case (i): For the condition (3.10) to hold, there must exist a non-zero vector $\mathbf{v}$ in the left null space of $\mathbf{F}$. It is easy to see that, under the GRE model, the set of all matrices $\mathbf{F}$ that have a row-rank of t is a set of measure 1. This implies that the set of all matrices $\mathbf{F}$ for each of which there exists some non-zero vector $\mathbf{v}$ in the left null space of $\mathbf{F}$ is a set of measure zero. Thus, for $t \leq L$, $\mathcal{F}$ is a set of measure zero.

Case (ii): For a vector $\mathbf{v}$, let the weight of $\mathbf{v}$, denoted by $\mathrm{wt}(\mathbf{v})$, be the number of non-zero elements in $\mathbf{v}$. For any integer $1 \leq w \leq t$, let $\mathcal{F}_w$ be the set of all matrices $\mathbf{F}$ for each of which there exists a non-zero vector $\mathbf{v}$ such that $\mathrm{wt}(\mathbf{v}) = w$ and the condition (3.10) holds.

We consider two cases as follows: (1) $w \leq \rho$, and (2) $w > \rho$. (Recall that $\rho = N - K - t$.)

(1) $w \leq \rho$: Assume, without loss of generality, that $v_1, v_2, \cdots, v_w$ are the non-zero elements of $\mathbf{v}$. Let $\mathbf{H}_w \triangleq ((v_k \cdot \alpha_{j_k}^{(i-1)})_{i\in[w],k\in[w]})$ be the $w \times w$ sub-matrix of $\mathbf{H}$ (defined in (3.10)) corresponding to the first w rows and the first w columns, and let $\mathbf{F}_w \triangleq ((f_{i,l})_{i\in[w],l\in[L]})$ be the $w \times L$ sub-matrix of $\mathbf{F}$ corresponding to the first w rows. Then, the condition (3.10) reduces to

$$\mathbf{H}_w \cdot \mathbf{F}_w = 0.$$

It is easy to see that the matrix $\mathbf{H}_w$ generates a Generalized Reed-Solomon code with distinct parameters $\{\alpha_{j_i}\}_{i\in[w]}$ and non-zero multipliers $\{v_i\}_{i\in[w]}$. Thus, $\mathbf{H}_w$ is full rank (and hence, invertible). This implies that for each $l \in [L]$ the column-vector $\mathbf{f}^{(l)}$ (defined in (3.9)) is an all-zero vector. Thus, every matrix in $\mathcal{F}_w$ for $w \leq \rho$ contains a $w \times L$ all-zero sub-matrix. In particular, every matrix in $\mathcal{F}_w$ for $w \leq \rho$ has at least one fixed (zero, in this case) entry. Under the GRE model, it is then easy to see that $\mathcal{F}_w$ for $w \leq \rho$ is a set of measure zero.

(2) $w > \rho$: Assume, without loss of generality, that $v_1, \ldots, v_w$ are the non-zero elements of $\mathbf{v}$, and let $\tilde{\mathbf{v}} \triangleq (v_1, v_2, \cdots, v_w)$. Let $\mathbf{\Phi}_w$ be the $w \times \rho L$ sub-matrix of $\mathbf{\Phi}$ (defined in (3.12)) corresponding to the first w rows,

$$\mathbf{\Phi}_w \triangleq \begin{pmatrix} f_{1,1} & \cdots & f_{1,L} & (\alpha_{j_1} f_{1,1}) & \cdots & (\alpha_{j_1} f_{1,L}) & \cdots & (\alpha_{j_1}^{(\rho-1)} f_{1,1}) & \cdots & (\alpha_{j_1}^{(\rho-1)} f_{1,L}) \\ f_{2,1} & \cdots & f_{2,L} & (\alpha_{j_2} f_{2,1}) & \cdots & (\alpha_{j_2} f_{2,L}) & \cdots & (\alpha_{j_2}^{(\rho-1)} f_{2,1}) & \cdots & (\alpha_{j_2}^{(\rho-1)} f_{2,L}) \\ \vdots & & \vdots & \vdots & & \vdots & & \vdots & & \vdots \\ f_{w,1} & \cdots & f_{w,L} & (\alpha_{j_w} f_{w,1}) & \cdots & (\alpha_{j_w} f_{w,L}) & \cdots & (\alpha_{j_w}^{(\rho-1)} f_{w,1}) & \cdots & (\alpha_{j_w}^{(\rho-1)} f_{w,L}) \end{pmatrix}.$$

Then, the condition (3.11) reduces to

$$\tilde{\mathbf{v}} \cdot \mathbf{\Phi}_w = 0. \tag{3.13}$$

Since in (3.3) the number of variables must be less than the number of equations, then $w \leq t \leq \rho L$. Note that $\mathbf{\Phi}_w$ is a $w \times \rho L$ matrix. Thus, $\mathrm{rank}(\mathbf{\Phi}_w) \leq w$. Moreover, there exists a non-zero vector $\tilde{\mathbf{v}}$ in the left null space of $\mathbf{\Phi}_w$. This implies that $\mathrm{rank}(\mathbf{\Phi}_w) \leq w - 1$. Since the row-rank and the column-rank are equal, there exists a non-zero column-vector $\mathbf{u}$ such that

$$\mathbf{\Phi}_w \cdot \mathbf{u} = 0.$$

Let $\alpha_i \triangleq \alpha_{j_i}$ for $i \in [w]$, and let $\boldsymbol{\alpha}^{(k)} = (\alpha_1^{k-1}, \alpha_2^{k-1}, \cdots, \alpha_w^{k-1})^{\mathsf{T}}$ for $k \in [\rho]$. We define the product operator $\odot$ between the two vectors $\boldsymbol{\alpha}^{(k)}$ and $\mathbf{f}^{(l)}$ as

$$\boldsymbol{\alpha}^{(k)} \odot \mathbf{f}^{(l)} \triangleq (\alpha_1^{(k-1)} f_{1,l}, \alpha_2^{(k-1)} f_{2,l}, \ldots, \alpha_w^{(k-1)} f_{w,l})^{\mathsf{T}}.$$

Then, we can rewrite $\boldsymbol{\Phi}_w$ as

$$\left(\boldsymbol{\alpha}^{(1)} \odot \mathbf{f}^{(1)}, \ldots, \boldsymbol{\alpha}^{(1)} \odot \mathbf{f}^{(L)}, \boldsymbol{\alpha}^{(2)} \odot \mathbf{f}^{(1)}, \ldots, \boldsymbol{\alpha}^{(2)} \odot \mathbf{f}^{(L)}, \ldots, \boldsymbol{\alpha}^{(\rho)} \odot \mathbf{f}^{(1)} \ldots, \boldsymbol{\alpha}^{(\rho)} \odot \mathbf{f}^{(L)}\right).$$

Since $\mathbf{u} = (u_1, \ldots, u_L, u_{L+1}, \ldots, u_{L+L}, \ldots, u_{(\rho-1)L+1}, \ldots, u_{(\rho-1)L+L}) \neq 0$, there exist $l \in [L]$ and $k \in [\rho]$ such that $u_{(k-1)L+l}$ is non-zero. Assume, without loss of generality, that $u_1 \neq 0$. Consider the columns $\boldsymbol{\alpha}^{(1)} \odot \mathbf{f}^{(1)}, \boldsymbol{\alpha}^{(2)} \odot \mathbf{f}^{(1)}, \ldots, \boldsymbol{\alpha}^{(\rho)} \odot \mathbf{f}^{(1)}$ in the matrix $\boldsymbol{\Phi}_w$, and their corresponding elements $u_1, u_{L+1}, \ldots, u_{(\rho-1)L+1}$ in the vector $\mathbf{u}$. Let $\tilde{u}_k \triangleq u_{(k-1)L+1}$ for $k \in [\rho]$, and let $\tilde{\mathbf{u}} \triangleq (\tilde{u}_1, \ldots, \tilde{u}_\rho)$. Note that $\tilde{\mathbf{u}} \neq 0$ (by construction). Consider the vector

$$\mathbf{g} \triangleq \tilde{u}_1(\boldsymbol{\alpha}^{(1)} \odot \mathbf{f}^{(1)}) + \tilde{u}_2(\boldsymbol{\alpha}^{(2)} \odot \mathbf{f}^{(1)}) + \cdots + \tilde{u}_\rho(\boldsymbol{\alpha}^{(\rho)} \odot \mathbf{f}^{(1)}).$$

Expanding $\mathbf{g} = (g_1, \ldots, g_w)^{\mathsf{T}}$, we get $g_i = (\tilde{u}_1\alpha_i^0 + \tilde{u}_2\alpha_i^1 + \cdots + \tilde{u}_\rho\alpha_i^{\rho-1}) f_{i,1}$ for all $i \in [w]$. Note that there exists $i \in [w]$ such that the coefficient of $f_{i,1}$ in g_i, i.e., $\tilde{u}_1\alpha_i^0 + \tilde{u}_2\alpha_i^1 + \cdots + \tilde{u}_\rho\alpha_i^{\rho-1}$, is non-zero. The proof is by the way of contradiction. Suppose that for all $i \in [w]$ the coefficient of $f_{i,1}$ in g_i is zero. Let $\mathbf{M} \triangleq ((\alpha_i^{k-1})_{i\in[w],k\in[\rho]})$. Then it is easy to see that $\mathbf{M} \cdot \tilde{\mathbf{u}} = 0$. Since $\mathbf{M}$ is a $w \times \rho$ Vandermonde matrix with $\rho < w$, then $\mathrm{rank}(\mathbf{M}) = \rho$. This implies that $\tilde{\mathbf{u}} = 0$. This is however a contradiction because $\tilde{\mathbf{u}} \neq 0$ (by assumption). Thus, for some $i \in [w]$ the coefficient of $f_{i,1}$ in g_i must be non-zero. Thus, every matrix in $\mathcal{F}_w$ for $w > \rho$ contains at least one entry which can be written as a linear combination of the rest of the entries. Under the GRE model, this readily implies that $\mathcal{F}_w$ is a set of measure zero.

Noting that $\mathcal{F} = \cup_{w=1}^{t} \mathcal{F}_w$ and taking a union bound over all w ($1 \leq w \leq t$), it follows that for $t > L$, $\mathcal{F}$ is a set of measure zero. This completes the proof. □

3.6.2 Probability of Undetected Error

Similarly as in the case of the finite fields, both the MSSR decoding algorithm and the collaborative Peterson's decoding algorithm give an error locator polynomial $\Lambda(z)$ over the real field ($\mathbb{R}$) of the least possible degree which satisfies all the syndrome equations in (3.3). This implies that these decoding algorithms have the ML certificate property (for details, see Section 3.5.2).

As was shown by Dutta *et al.* in [12, Theorem 3], under the GRE model, when the number of errors (i.e., the Hamming weight of the error matrix) is less than $N - K$, with probability 1 the closest codeword to the received vector is the transmitted codeword. This implies that for any decoding algorithm satisfying the ML certificate property, the set of all bad error matrices (defined in Section 3.5.2) is of measure zero, and thereby, the probability of undetected error is zero.

Theorem 10. *Under the GRE model, for all $t \leq N - K - 1$ (and in particular, for all $t \leq t_{\max} = \frac{L}{L+1}(N-K)$), $P_{\mathrm{ML}}(t) = 0$.*

3.7 Numerical Results

We present simulation results for $N = 8$, $K = 2$, and $\alpha_i = 0.9^i$ for different L. Fig. 3.1 shows the probability of error ($P_e(t) = P_F(t) + P_{\mathrm{ML}}(t)$) for decoding GRS codes individually using Peterson's algorithm ($L = 1$), decoding GRS codes individually using the ℓ_1 minimization decoder, and collaborative decoding using the CPDA algorithm with $L = 6$. For each data point, 12500 IGRS codewords were simulated. It can be seen that the CPDA with $L = 6$ corrects all t errors for $t \leq N - K - 1$, which is a significant improvement over decoding GRS codes individually. This is consistent with the theoretical results. The probability of error for the ℓ_1 minimization decoder remains fairly high for several values of $t \leq N - K - 1$. These results are consistent with the results of Candes and

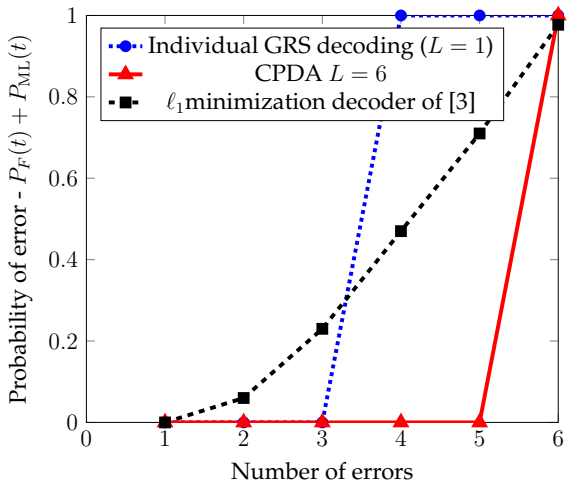

Figure 3.1: The probability of error versus the number of errors for different decoding algorithms for $N = 8$ and $K = 2$. Reprinted from [55]

Tao (Figures 2 and 3 in [3]). This shows that individually decoding GRS decoder using the ℓ_1-minimization decoder does not suffice to achieve small probability of error as suggested in [12]; whereas, collaborative decoding can achieve the decoding radius bound of $N - K - 1$ with polynomial complexity.

For larger values of N and K, we noticed that computing the rank of $\mathbf{S}_L(t)$ had numerical inaccuracies. This is a well-known issue with decoding GRS codes over the real field. Interestingly, from simulations, we observe that collaborative decoding seems to alleviate this issue. Table 3.1 shows the probability of error ($P_e(t) = P_F(t) + P_{\mathrm{ML}}(t)$) for $N = 20$, $K = 12$ and $\alpha_i = i$. For a fixed number of errors, increasing L improved the condition number of $\mathbf{S}_L(t)^{\mathsf{T}}\mathbf{S}_L(t)$. With $L = 20$, we were able to decode up to $N - K - 1$ errors with $P_e(t) = 0$ in 12500 trials.

Our results have shown that collaborative decoding of Polynomial codes can correct up to $t_{\max} = \frac{L}{L+1}(N - K)$ errors. It can be seen that $t_{\max} = N - K - 1$ for all $L \geq N - K - 1$ and hence, it is natural to wonder if there is any advantage in increasing L beyond $N - K - 1$. Here we empirically show that increasing L improves the numerical stability of the collaborative Peterson's algorithm for determining the error locator polynomial. Fig. 4.4 ($N = 8$, $K = 2$, $\alpha_i = 0.9^i$) shows a plot of the average condition number of the stacked syndome matrix $\mathbf{S}_L(t)$ (defined in (3.3)) as a function of t for different L. It can be

Table 3.1: Probability of error for the CPDA, $N = 20$ $K = 12$, 12500 trials. Reprinted from [55]

$L\backslash t$	1	2	3	4	5	6	7
1	0	0	0	0.0008	-	-	-
2	0	0	0	0	0	-	-
3	0	0	0	0	0	0	-
4	0	0	0	0	0	0	-
5	0	0	0	0	0	0	-
6	0	0	0	0	0	0	-
7	0	0	0	0	0	0	0.0026
8	0	0	0	0	0	0	0.0008
20	0	0	0	0	0	0	0

seen from simulations that for all t, increasing L decreases the average condition number. Since the collaborative Peterson's algorithm requires inversion of the matrix $\mathbf{S}_L^{\mathsf{T}}(t)\mathbf{S}_L(t)$, the numerical stability of the algorithm will improve with increasing L.

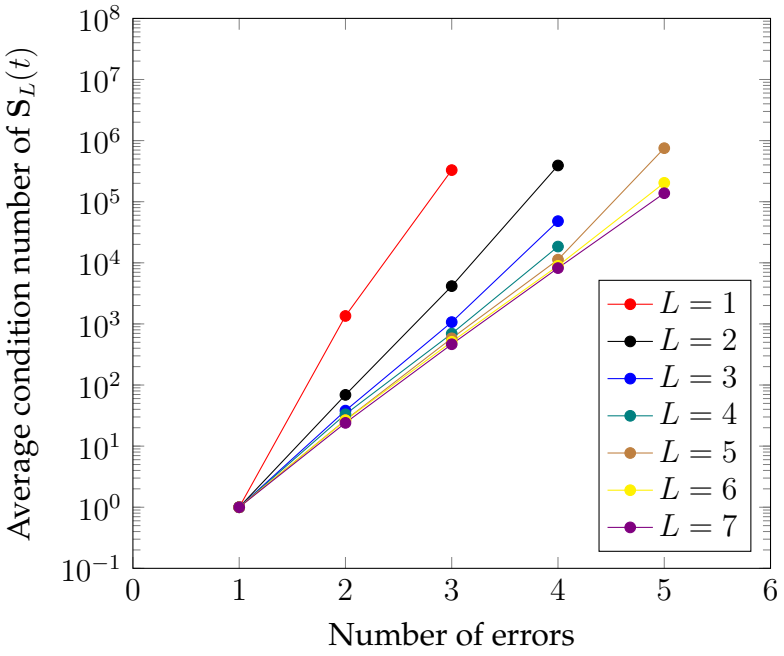

Figure 3.2: Average condition number of $\mathbf{S}_L^T(t)\mathbf{S}_L(t)$, $N = 8, K = 2$. Reprinted from [55]

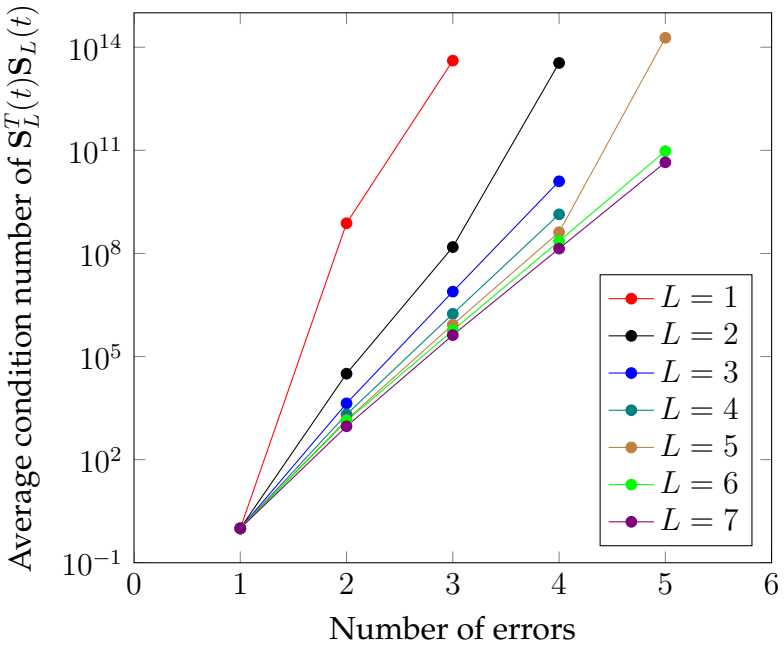

Figure 3.3: Average condition number of $\mathbf{S}_L^T(t)\mathbf{S}_L(t)$, $N = 8$ $K = 2$. Reprinted from [55]

4. RANDOM KHATRI-RAO-PRODUCT CODES FOR NUMERICALLY-STABLE DISTRIBUTED MATRIX MULTIPLICATION[1]

4.1 Introduction and Main Results

In this chapter, we consider the same problem in chapter 3 of computing $\mathbf{A}^\mathsf{T}\mathbf{B}$ for two matrices $\mathbf{A} \in \mathbb{R}^{N_2 \times N_1}$ and $\mathbf{B} \in \mathbb{R}^{N_2 \times N_3}$ in a distributed fashion using a coded matrix multiplication scheme with N worker nodes but in the presence of straggling/erasure nodes [12, 13, 29, 30, 32, 33, 49, 63, 67, 68, 71]. In [67], Yu, Maddah-Ali and Avestimehr proposed an elegant encoding scheme called Polynomial codes in which the matrices $\mathbf{A}^\mathsf{T}$ and $\mathbf{B}$ are each split into m and n sub-matrices, respectively, the sub-matrices are encoded using a polynomial code and the computations are distributed to N worker nodes. This scheme is shown to have optimal recovery threshold, i.e., the matrix product $\mathbf{A}^\mathsf{T}\mathbf{B}$ can be computed (recovered) using the results of computation from any subset of worker nodes of cardinality $K = mn$. In the language of coding theory, Polynomial codes are generalized Reed-Solomon codes, their generator matrices have Vandermonde structures, and they are maximum distance separable (MDS) codes.

In this chapter, we propose a coding scheme for the distributed matrix multiplication problem which we call Random Khatri-Rao-Product (RKRP) codes which exhibits substantially better numerical stability than Polynomial codes [67] and OrthoPoly codes [18]. The proposed coding scheme is not based on polynomial interpolation; rather, it is designed in the spirit of random codes in information theory.

RKRP codes split both $\mathbf{A}^\mathsf{T}$ and $\mathbf{B}$ into sub-matrices and encode them by forming random linear combinations of the sub-matrices. The proposed RKRP codes have several desirable features: (i) RKRP codes have the same thresholds, encoding complexity and communication cost as that of Polynomial codes and OrthoPoly codes; (ii) Decoding pro-

[1]The contents of this section have been presented in verbatim as part of the paper "Random Khatri-Rao-product codes for numerically-stable distributed matrix multiplication" at Allerton 2019 [57] for which I was the first author

cess of RKRP codes is substantially more numerically stable than that of Polynomial and OrthoPoly codes, and decoding can be implemented even for fairly large values of K, S, N (e.g., $K = 1000$ and any S, N); and (iii) decoding complexity of RKRP codes is lower than that of OrthoPoly codes. To the best of our knowledge, the RKRP code construction and the analysis of their MDS property are new.

We present two ensembles of generator matrices for RKRP codes called the non-systematic RKRP ensemble and the systematic RKRP ensemble. Codes from these ensembles will be referred to as *non-systematic RKRP codes* and *systematic RKRP codes*[2], respectively. Systematic RKRP codes have better average decoding complexity and better numerical stability. Hence, systematic RKRP codes would be preferred over non-systematic RKRP codes for most applications. However, we present both non-systematic and systematic ensembles in this chapter for the following reasons. Since Polynomial and OrthoPoly codes are presented with non-systematic encoding, non-systematic RKRP codes allow for a fair comparison with Polynomial and OrthoPoly codes. The proofs are also easier to follow when presented for the non-systematic ensemble first and then extended to the systematic ensemble. Finally, non-systematically RKRP codes provide privacy which systematic RKRP codes do not, although this issue is not studied further in this chapter.

4.2 System Model and Preliminaries

We consider a system with one master node which has access to matrices $\mathbf{A}^\mathsf{T}$ and $\mathbf{B}$ and N worker nodes which can perform multiplication of sub-matrices of $\mathbf{A}^\mathsf{T}$ and $\mathbf{B}$. At the master node, the matrix $\mathbf{A}^\mathsf{T}$ is split into m sub-matrices row-wise and $\mathbf{B}$ is split into n

[2]The terminology of associating the words systematic and non-systematic with the code, rather than with the encoder is not standard in coding theory. While it is possible to find a systematic encoder for a non-systematic RKRP code, the resulting code would not belong to the systematic RKRP ensemble and hence, should be treated as a non-systematic RKRP code.

sub-matrices column-wise as shown below.

$$\mathbf{A}^\mathsf{T} = \begin{bmatrix} \mathbf{A}_1^\mathsf{T} \\ \mathbf{A}_2^\mathsf{T} \\ \vdots \\ \mathbf{A}_m^\mathsf{T} \end{bmatrix}, \quad \mathbf{B} = \begin{bmatrix} \mathbf{B}_1 & \mathbf{B}_2 & \cdots & \mathbf{B}_n \end{bmatrix} \tag{4.1}$$

In order to compute the matrix product $\mathbf{A}^\mathsf{T}\mathbf{B}$, we need to compute the matrix products $\mathbf{A}_j^\mathsf{T}\mathbf{B}_l$ for $j = 1, \ldots, m$ and $l = 1, \ldots, n$. The main idea in distributed coded computation is to first encode $\mathbf{A}_1^\mathsf{T}, \ldots, \mathbf{A}_m^\mathsf{T}$ and $\mathbf{B}_1, \ldots, \mathbf{B}_n$ into N pairs of matrices $\left(\tilde{\mathbf{A}}_i^\mathsf{T}, \tilde{\mathbf{B}}_i\right), i = 1, \ldots, N$.[3] The ith worker node is then tasked with computing the matrix product $\tilde{\mathbf{C}}_i = \tilde{\mathbf{A}}_i^\mathsf{T}\tilde{\mathbf{B}}_i$. It is assumed that K out of the N workers return the result of their computation; these worker nodes are called non-stragglers. Without loss of generality we assume that the non-stragglers are worker nodes $1, \ldots, K$.

Definition 11. *An* encoding scheme *is a mapping from* $(\mathbf{A}_1^\mathsf{T}, \ldots, \mathbf{A}_m^\mathsf{T}, \mathbf{B}_1, \ldots, \mathbf{B}_n)$ *to* $\{(\tilde{\mathbf{C}}_i = \tilde{\mathbf{A}}_i^\mathsf{T}\tilde{\mathbf{B}}_i)\}$ *for* $i = 1, \ldots, N$. *A* codeword *is a vector of matrices* $\underline{\tilde{\mathbf{C}}_i} = [\tilde{\mathbf{C}}_1, \tilde{\mathbf{C}}_2, \ldots, \tilde{\mathbf{C}}_N]$. *A* code *is the set of possible codewords* $\{\underline{\tilde{\mathbf{C}}}\}$.

Definition 12. *An encoding scheme is said to result in a maximum distance separable (MDS) code, or the corresponding code is said to be MDS, if the set of matrix products* $\{\mathbf{A}_i^\mathsf{T}\mathbf{B}_j\}$ *for* $i = 1, \ldots, m$ *and* $j = 1, \ldots, n$ *can be computed (recovered) from any subset of* $\{\tilde{\mathbf{C}}_1, \tilde{\mathbf{C}}_2, \ldots, \tilde{\mathbf{C}}_N\}$ *of size* mn, *where* $\tilde{\mathbf{C}}_i = \tilde{\mathbf{A}}_i^\mathsf{T}\tilde{\mathbf{B}}_i$.

Definition 13. *The row-wise Khatri-Rao product of two matrices* $\mathbf{P} \in \mathbb{R}^{K \times m}$ *and* $\mathbf{Q} \in \mathbb{R}^{K \times n}$ *denoted by* $\mathbf{P} \odot \mathbf{Q}$ *is given by the matrix* $\mathbf{M}$ *whose* i*th row is the Kronecker product of the* i*th row of* $\mathbf{P}$ *and the* i*th row of* $\mathbf{Q}$, *i.e.,*

$$\mathbf{M}(i,:) = \mathbf{P}(i,:) \otimes \mathbf{Q}(i,:) \tag{4.2}$$

where $\otimes$ *refers to the Kronecker product.*

[3] This is not the most general form of encoding but many of the existing encoding schemes in the literature as well as the proposed scheme can be represented in this way.

4.3 Non-Systematically encoded Random Khatri-Rao-Product Codes

4.3.1 Encoding:

Our proposed non-systematic RKRP codes are encoded as follows. For $i = 1, \ldots, N$, the master node computes

$$\tilde{\mathbf{A}}_i^\mathsf{T} = \sum_{j=1}^{m} p_{i,j} \mathbf{A}_j^\mathsf{T}, \tag{4.3}$$

$$\tilde{\mathbf{B}}_i = \sum_{l=1}^{n} q_{i,l} \mathbf{B}_l \tag{4.4}$$

where $p_{i,j}, q_{i,l}$ are realizations of independent identically distributed random variables $P_{i,j}$ and $Q_{i,l}$, respectively. Both $P_{i,j}$ and $Q_{i,l}$ are assumed to be continuous random variables with a probability density function f $\forall i, j, l$, i.e., their distribution is absolutely continuous with respect to the Lebesgue measure. $\tilde{\mathbf{A}}_i$ and $\tilde{\mathbf{B}}_i$ are then transmitted to the ith worker node which is tasked with computing $\tilde{\mathbf{C}}_i = \tilde{\mathbf{A}}_i^\mathsf{T} \tilde{\mathbf{B}}_i$. We first note that $\tilde{\mathbf{C}}_i$ can be written as

$$\begin{aligned} \tilde{\mathbf{C}}_i &= \left(\sum_{j=1}^{m} p_{i,j} \mathbf{A}_j^\mathsf{T} \right) \left(\sum_{l=1}^{n} q_{i,l} \mathbf{B}_l \right) \\ &= \sum_{j=1}^{m} \sum_{l=1}^{n} p_{i,j} q_{i,l} \mathbf{A}_j^\mathsf{T} \mathbf{B}_l. \end{aligned} \tag{4.5}$$

Since the matrix $\tilde{\mathbf{C}}_i$ is a linear combination of the desired matrix products $\mathbf{A}_j^\mathsf{T} \mathbf{B}_l$, the (s,t)th entry of $\tilde{\mathbf{C}}_i$, namely $[\tilde{\mathbf{C}}_i](s,t)$, is a linear combination of the (s,t)th entries of the matrix products $\mathbf{A}_j^\mathsf{T} \mathbf{B}_l$, namely $[\mathbf{A}_j^\mathsf{T} \mathbf{B}_l](s,t)$. During the decoding process, we attempt to recover $[\mathbf{A}_j^\mathsf{T} \mathbf{B}_l](s,t)$ from $\tilde{\mathbf{C}}_i(s,t), \ldots, [\tilde{\mathbf{C}}_i](s,t)$ for each pair of s, t separately.

To keep the discussions clear, we focus on the recovery of the $(1,1)$th entry of $\mathbf{A}_j^\mathsf{T} \mathbf{B}_l$, namely $[\mathbf{A}_j^\mathsf{T} \mathbf{B}_l](1,1)$. The same idea extends to the recovery of other indices as well. Let $y_i = [\tilde{\mathbf{C}}_i](1,1)$ denote the $(1,1)$th entry in the matrix product computed by the ith non-straggler worker node, and let $z_{j,l} = [\mathbf{A}_j^\mathsf{T} \mathbf{B}_l](1,1)$.

The vector of computed values can be written as a linear combination of matrix prod-

ucts given by

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_i \\ \vdots \\ y_N \end{bmatrix} = \begin{bmatrix} p_{1,1}q_{1,1} & p_{1,1}q_{1,2} & \cdots & p_{1,1}q_{1,n} & \cdots & p_{1,m}q_{1,n} \\ p_{2,1}q_{2,1} & p_{2,1}q_{2,2} & \cdots & p_{2,1}q_{2,n} & \cdots & p_{2,m}q_{2,n} \\ & & \vdots & & & \\ p_{i,1}q_{i,1} & p_{i,1}q_{i,2} & \cdots & p_{i,1}q_{i,n} & \cdots & p_{i,m}q_{i,n} \\ & & \vdots & & & \\ p_{N,1}q_{N,1} & p_{N,1}q_{N,2} & \cdots & p_{N,1}q_{N,n} & \cdots & p_{N,m}q_{N,n} \end{bmatrix} \begin{bmatrix} z_{1,1} \\ z_{1,2} \\ \vdots \\ z_{1,n} \\ \vdots \\ z_{m,n} \end{bmatrix} \tag{4.6}$$

It will be more convenient to express (4.6) in a slightly different form. For $j \in \{1, \ldots, mn\}$, let $j' = \lceil j/n \rceil$ and $j'' = (j-1) \mod n + 1$, and let $w_j = z_{j',j''}$. Without loss of generality, let us assume that the worker nodes which return their computation are worker nodes $1, 2, \ldots, K$. The computed values y_i's are related to the unknown values w_j's according to

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_i \\ \vdots \\ y_K \end{bmatrix} = \begin{bmatrix} p_{1,1}q_{1,1} & p_{1,1}q_{1,2} & \cdots & p_{1,1}q_{1,n} & \cdots & p_{1,m}q_{1,n} \\ p_{2,1}q_{2,1} & p_{2,1}q_{2,2} & \cdots & p_{2,1}q_{2,n} & \cdots & p_{2,m}q_{2,n} \\ & & \vdots & & & \\ p_{i,1}q_{i,1} & p_{i,1}q_{i,2} & \cdots & p_{i,1}q_{i,n} & \cdots & p_{i,m}q_{i,n} \\ & & \vdots & & & \\ p_{K,1}q_{K,1} & p_{K,1}q_{K,2} & \cdots & p_{K,1}q_{K,n} & \cdots & p_{K,m}q_{K,n} \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_j \\ \vdots \\ w_K \end{bmatrix} \tag{4.7}$$

or, more succinctly as

$$\underline{y} = \mathbf{G}\, \underline{w} \tag{4.8}$$

where $\underline{y} = [y_1, y_2, \ldots, y_K]^\mathsf{T}$, $\underline{w} = [w_1, w_2, \ldots, w_{mn}]^\mathsf{T}$, and $\mathbf{G}$ is an $N \times mn$ generator matrix for a code with $[\mathbf{G}]_{i,j} = p_{i,j'}q_{i,j''}$.

Let $\mathbf{P}$ and $\mathbf{Q}$ be two matrices whose entries are given by $[\mathbf{P}]_{i,j'} = p_{i,j'}$ and $[\mathbf{Q}]_{i,j''} = q_{i,j''}$. It can be seen that

$$\mathbf{G} = \mathbf{P} \odot \mathbf{Q}, \tag{4.9}$$

i.e., $\mathbf{G}$ is the row-wise Khatri-Rao product of two matrices $\mathbf{P}$ and $\mathbf{Q}$. Hence, we call these codes as Random Khatri-Rao-Product codes.

Example 14. *In order to clarify the main idea, consider an example with $m = 2$ and $n = 3$ and $N > 6$. Without loss of generality, assume that the worker nodes $1, 2, \ldots, 6$ return the results of their computations, namely, $\tilde{\mathbf{C}}_1, \ldots, \tilde{\mathbf{C}}_6$. In this case, the set of computations returned by the worker nodes is related to the matrix products that we need to compute according to*

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \end{bmatrix} = \begin{bmatrix} p_{1,1}q_{1,1} & p_{1,1}q_{1,2} & p_{1,1}q_{1,3} & p_{1,2}q_{1,1} & p_{1,2}q_{1,2} & p_{1,2}q_{1,3} \\ p_{2,1}q_{2,1} & p_{2,1}q_{2,2} & p_{2,1}q_{2,3} & p_{2,2}q_{2,1} & p_{2,2}q_{2,2} & p_{2,2}q_{2,3} \\ p_{3,1}q_{3,1} & p_{3,1}q_{3,2} & p_{3,1}q_{3,3} & p_{3,2}q_{3,1} & p_{3,2}q_{3,2} & p_{3,2}q_{3,3} \\ p_{4,1}q_{4,1} & p_{4,1}q_{4,2} & p_{4,1}q_{4,3} & p_{4,2}q_{4,1} & p_{4,2}q_{4,2} & p_{4,2}q_{4,3} \\ p_{5,1}q_{5,1} & p_{5,1}q_{5,2} & p_{5,1}q_{5,3} & p_{5,2}q_{5,1} & p_{5,2}q_{5,2} & p_{5,2}q_{5,3} \\ p_{6,1}q_{6,1} & p_{6,1}q_{6,2} & p_{6,1}q_{6,3} & p_{6,2}q_{6,1} & p_{6,2}q_{6,2} & p_{6,2}q_{6,3} \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \\ w_5 \\ w_6 \end{bmatrix} \tag{4.10}$$

Definition 15. *The ensemble of $N \times K$ generator matrices obtained by choosing the generator matrix $\mathbf{G}$ as in* (4.9) *where $p_{i,j}, q_{i,j}$ are realizations of random variables $P_{i,j}, Q_{i,j}$ such that $\{P_{1,1}, \ldots, P_{N,m}, Q_{1,1}, \ldots, Q_{N,n}\}$ is a set of independent random variables with probability density function f will be referred to as the* non-systematic random Khatri-Rao-product generator matrix ensemble $\mathcal{G}_{non-sys}(N, K, f)$.

4.3.2 Decoding:

During decoding, an estimate of $\underline{w}$, namely $\underline{\hat{w}}$, is obtained as follows

$$\underline{\hat{w}} = \mathbf{G}^{-1}\underline{y}. \tag{4.11}$$

In the absence of numerical round-off errors, if $\mathbf{G}$ is invertible, then $\underline{\hat{w}} = \underline{w}$. However, when performing computation with finite bits of precision, there will be numerical errors

in the computation. Let $\underline{e} = \underline{w} - \underline{\hat{w}}$ be the error, and define the relative error as

$$\eta := \frac{||\underline{e}||_2}{||\underline{w}||_2}. \tag{4.12}$$

4.4 Non-Systematic RKRP codes are MDS codes with probability 1

Our first main result in this chapter is that if a generator matrix is randomly chosen from the non-systematic RKRP ensemble $\mathcal{G}_{non-sys}(N, K, f)$, the encoding scheme defined in (4.4) results in an MDS code with probability 1.

Lemma 16. *Consider an analytic function $h(\underline{x})$ of several real variables $\underline{x} = [x_1, x_2, \cdots, x_n] \in \mathbb{R}^n$. If $h(\underline{x})$ is nontrivial in the sense that there exists $\underline{x}_0 \in \mathbb{R}^n$ such that $h(\underline{x}_0) \neq 0$ then the zero set of $h(\underline{x})$,*

$$\mathcal{Z} = \{\underline{x} \in \mathbb{R}^n \mid h(\underline{x}) = 0\}$$

is of measure (Lebesgue measure in $\mathbb{R}^n$) zero.

Proof: This lemma is proved in [22, Lemma 1] for the complex field $\mathbb{C}$. The proof for the real field can be obtained by following the same steps and replacing $\mathbb{C}$ with $\mathbb{R}$. □

Theorem 17. *Non-systematic RKRP codes are MDS codes with probability 1.*

Proof: To prove the theorem, we need to prove that the matrix $\mathbf{G}$ obtained when $K = mn$ in (4.7) is a full rank matrix with probability 1. Let $p_{i,j}$ be a realization of the random variable $P_{i,j}$ and let $q_{i,j}$ be a realization of the random variable $Q_{i,j}$. The generator matrix

in (4.7) is a realization of the matrix of random variables $\{P_{i,j}\}_{i\in[K],j\in[m]}$ and $\{Q_{i,j}\}_{i\in[K],j\in[n]}$:

$$\mathbf{\Gamma} = \begin{bmatrix} P_{1,1}Q_{1,1} & P_{1,1}Q_{1,2} & \dots & P_{1,1}Q_{1,n} & \dots & P_{1m}Q_{1,n} \\ P_{2,1}Q_{2,1} & P_{2,1}Q_{2,2} & \dots & P_{2,1}Q_{2,n} & \dots & P_{2m}Q_{2,n} \\ & & \vdots & & & \\ P_{i,1}Q_{i,1} & P_{i,1}Q_{i2} & \dots & P_{i,1}Q_{in} & \dots & P_{i,m}Q_{in} \\ & & \vdots & & & \\ P_{K,1}Q_{K,1} & P_{K,1}Q_{K,2} & \dots & P_{K,1}Q_{K,n} & \dots & P_{K,m}Q_{K,n} \end{bmatrix} \tag{4.13}$$

We will show that $\Pr(\text{rank}(\mathbf{\Gamma}) \neq mn) = 0$. The determinant of $\mathbf{\Gamma}$ is a polynomial in the variables $\{P_{i,j}\}_{i\in[K],j\in[m]}$ and $\{Q_{i,j}\}_{i\in[K],j\in[n]}$ with degree $2mn$. Let

$$\det(\mathbf{\Gamma}) = h(P_{1,1}, \dots, P_{K,m}, Q_{1,1}, \dots, Q_{K,n}) \tag{4.14}$$

We first show that there exists at least one $P_{1,1}, \dots, P_{K,m}, Q_{1,1}, \dots, Q_{K,n}$ for which

$$h(P_{1,1}, \dots, P_{K,m}, Q_{1,1}, \dots, Q_{K,n}) \neq 0.$$

For $j \in [mn]$, let $j' = \lceil j/n \rceil$ and $j'' = ((j-1) \mod n) + 1$. Let $P_{j,j'} = 1, Q_{j,j''} = 1, \forall j \in [mn]$, $P_{j,l} = 0, \forall j \in [mn], l \neq j'$, and $Q_{j,l} = 0, \forall j \in [mn], l \neq j''$. For this choice of $P_{1,1}, \dots, P_{K,m}, Q_{1,1}, \dots, Q_{K,n}$, it can be seen that the matrix $\mathbf{\Gamma}$ reduces to an identity matrix, and hence $h(P_{1,1}, \dots, P_{K,m}, Q_{1,1}, \dots, Q_{K,n}) = 1$ ($\neq 0$). From Lemma 16, we then see that the zero set of h has measure zero, and hence, $\Pr(\text{rank}(\mathbf{\Gamma}) \neq mn) = 0$. □

4.5 Systematic Khatri-Rao-Product Codes

In this section, we introduce a systematic construction of random Khatri-Rao-Product codes which reduces the average encoding and decoding complexities compared to its non-systematic counterpart.

4.5.1 Encoding:

In systematic encoding, the first K worker nodes are simply given the submatrices $\mathbf{A}_j^\mathsf{T}$ and $\mathbf{B}_l$ without encoding and the other $N-K$ worker nodes are given encoded versions as in the non-systematic version. For $i \in \{1, \ldots, K = mn\}$, let $i' = \lceil i/n \rceil$ and $i'' = ((i-1) \mod n) + 1$. The encoding process can be described as below

$$\tilde{\mathbf{A}}_i^\mathsf{T} = \begin{cases} \mathbf{A}_{i'}^\mathsf{T}, & i \in [K], \\ \sum_{j=1}^{m} p_{i-K,j} \mathbf{A}_j^\mathsf{T}, & K+1 \leq i \leq N, \end{cases} \tag{4.15}$$

$$\tilde{\mathbf{B}}_i = \begin{cases} \mathbf{B}_{i''}, & i \in [K], \\ \sum_{l=1}^{n} q_{i-K,l} \mathbf{B}_l, & K+1 \leq i \leq N, \end{cases} \tag{4.16}$$

where $p_{i,j}, q_{i,j}$ are realizations of $P_{i,j}, Q_{i,j}$ which are absolutely continuous random variables with respect to the Lebesgue measure. $\tilde{\mathbf{A}}_i$ and $\tilde{\mathbf{B}}_i$ are then transmitted to the ith worker node which is tasked with computing $\tilde{\mathbf{C}}_i = \tilde{\mathbf{A}}_i^\mathsf{T} \tilde{\mathbf{B}}_i$. We will refer to worker nodes $1, 2, \ldots, K$ as *systematic worker nodes* and we will refer to worker nodes $K+1, \ldots, N$ as *parity worker nodes*.

As in the case of non-systematic encoding, we focus on the recovery of the $(1,1)$th entry of $\mathbf{A}_j^\mathsf{T} \mathbf{B}_l$, namely $[\mathbf{A}_j^\mathsf{T} \mathbf{B}_l](1,1)$. The same idea extends to the recovery of other indices as well. Let $y_i = [\tilde{\mathbf{C}}_i](1,1)$ denote the $(1,1)$th entry in the matrix product computed by the ith non-straggler worker node and let $z_{j,l} = [\mathbf{A}_j^\mathsf{T} \mathbf{B}_l]_{1,1}$. For $j \in [mn]$, let $j' = \lceil j/n \rceil$ and $j'' = ((j-1) \mod n) + 1$ and let $w_j = z_{j',j''}$. The computed values y_i's are related to the unknown values w_j's according to

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_K \\ y_{K+1} \\ \vdots \\ y_N \end{bmatrix} = \begin{bmatrix} 1 & 0 & \dots & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 & \dots & 0 \\ & & \vdots & & & \\ 0 & 0 & \dots & 0 & \dots & 1 \\ p_{1,1}q_{1,1} & p_{1,1}q_{1,2} & \dots & p_{1,1}q_{1,n} & \dots & p_{1m}q_{1,n} \\ p_{2,1}q_{2,1} & p_{2,1}q_{2,2} & \dots & p_{2,1}q_{2,n} & \dots & p_{2m}q_{2,n} \\ & & \vdots & & & \\ p_{S,1}q_{S,1} & p_{S,1}q_{S,2} & \dots & p_{S,1}q_{S,n} & \dots & p_{S,m}q_{S,n} \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_j \\ \vdots \\ w_K \end{bmatrix}, \tag{4.17}$$

where $S = N - K$.

The generator matrix in (4.17) can be written as $[\mathbf{I}_{K\times K}\ \mathbf{F}^{\mathsf{T}}]^{\mathsf{T}}$ where $\mathbf{F}$ is an $N-K\times K$ matrix whose ith row is given by $\underline{p}_i \odot \underline{q}_i$, i.e.,

$$\mathbf{F} = \begin{bmatrix} p_{1,1}q_{1,1} & p_{1,1}q_{1,2} & \dots & p_{1,1}q_{1,n} & \dots & p_{1m}q_{1,n} \\ p_{2,1}q_{2,1} & p_{2,1}q_{2,2} & \dots & p_{2,1}q_{2,n} & \dots & p_{2m}q_{2,n} \\ & & \vdots & & & \\ p_{S,1}q_{S,1} & p_{S,1}q_{S,2} & \dots & p_{S,1}q_{S,n} & \dots & p_{S,m}q_{S,n} \end{bmatrix}. \tag{4.18}$$

Definition 18. *The ensemble of $N \times K$ generator matrices obtained by choosing the generator matrix* $\mathbf{G}$ *as in* (4.17) *where $P_{i,j}, Q_{i,j} \sim f$ will be referred to as the* systematic random Khatri-Rao-product generator matrix ensemble $\mathcal{G}_{sys}(N, K, f)$.

4.5.2 Decoding

We consider the case when there are S_1 stragglers among the systematic worker nodes and $S_2 = S - S_1$ stragglers among the parity worker nodes. Without loss of generality we assume that the stragglers are the worker nodes $1, 2, \dots, S_1$ and $K + S_1 + 1, \dots, N$. This implies that the master nodes obtains $y_{S_1+1}, \dots, y_K$ and since the encoding is systematic, the master node can trivially recover $w_{S_1+1}, \dots, w_K$ by setting $w_i = y_i$ for $i = S_1+1, \dots, S_K$.

We can recover $w_1, \ldots, w_{S_1}$ from $y_{K+1}, \ldots, y_{K+S_1}$ as follows. Notice that

$$\begin{bmatrix} y_{K+1} \\ y_{K+2} \\ \vdots \\ y_{K+S_1} \end{bmatrix} = [\mathbf{F}](K+1:K+S_1, 1:S_1) \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_{S_1} \end{bmatrix} \tag{4.19}$$

$$+ [\mathbf{F}](K+1:K+S_1, S_1+1:K) \begin{bmatrix} w_{S_1+1} \\ w_{S_1+2} \\ \vdots \\ w_K \end{bmatrix}, \tag{4.20}$$

which in turn implies that

$$\underbrace{\begin{bmatrix} y_{K+1} \\ y_{K+2} \\ \vdots \\ y_{K+S_1} \end{bmatrix} - [\mathbf{F}](K+1:K+S_1, S_1+1:K) \begin{bmatrix} w_{S_1+1} \\ w_{S_1+2} \\ \vdots \\ w_K \end{bmatrix}}_{\underline{y}} \tag{4.21}$$

$$= \underbrace{[\mathbf{F}](K+1:K+S_1, 1:S_1)}_{\mathbf{G}_{sys}} \underbrace{\begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_{S_1} \end{bmatrix}}_{\underline{w}},$$

or more succinctly,

$$\underline{y} = \mathbf{G}_{sys}\, \underline{w}. \tag{4.22}$$

We can obtain an estimate of $\underline{w}$, namely $\underline{\hat{w}}$, as

$$\underline{\hat{w}} = \mathbf{G}_{sys}^{-1}\, \underline{y}. \tag{4.23}$$

Note that in the above description it is assumed that the stragglers were worker nodes $1, 2, \ldots, S$ and $K + S_1 + 1, \ldots, N$. However, the same ideas can be used for arbitrary sets of stragglers. The following example will clarify this.

Example 19. *Consider an example with $m = 2, n = 3, K = 6$ with $N = 10$ worker nodes. Let the straggler nodes be the worker nodes 2,4,5, and 8. In this case, we first recover w_1, w_3, w_6 by setting $w_1 = y_1, w_3 = y_3$ and $w_6 = y_6$. Then, we recover w_2, w_4, and w_5 from $w_1, w_3, w_6, y_7, y_9, y_{10}$ using*

$$\begin{bmatrix} y_7 \\ y_9 \\ y_{10} \end{bmatrix} - [\mathbf{F}](\{7,8,9\},\{1,3,6\}) \begin{bmatrix} w_1 \\ w_3 \\ w_6 \end{bmatrix} = [\mathbf{F}](\{7,8,9\},\{2,4,5\}) \begin{bmatrix} w_2 \\ w_4 \\ w_5 \end{bmatrix}. \tag{4.24}$$

We show that if a generator matrix is chosen at random from the systematic RKRP ensemble $\mathcal{G}_{sys}(N, K, f)$, the encoding scheme in (4.17) result in an MDS code with probability 1.

Theorem 20. *Systematic RKRP codes are MDS codes with probability 1.*

Proof: To prove the theorem, we need to prove that $\mathbf{G}_{sys}$ is full rank with probability 1. The proof follows along the same lines as the proof of Theorem 17. □

4.6 Decoding Complexities

In this section, we briefly discuss the decoding complexity of systematic RKRP codes and OrthoPoly codes. Decoding of systematic RKRP codes involves two steps. It involves inversion of the $S_1 \times S_1$ matrix $\mathbf{G}_{sys}$ in (4.22) whose complexity is $O(S_1^3)$. To retrieve every entry of $[\mathbf{A}_j^\intercal \mathbf{B}_l]$, we need to multiply $\mathbf{G}_{sys}^{-1}$ and $\underline{y}$ which requires $O(S_1^2)$ operations. This

step needs to be repeated for each of the $\frac{N_1 N_3}{mn}$ entries of $[\mathbf{A}_j^\mathsf{T}\mathbf{B}_l]$ and hence, the overall decoding complexity is $O(S_1^3 + S_1^2\frac{N_1 N_3}{mn})$.

OrthoPoly codes cannot be easily implemented in systematic form because of the multiplication by $\mathbf{H}$. Hence, the decoding complexity of OrthoPoly codes involves inverting a $K \times K$ matrix followed by $\frac{N_1 N_3}{mn}$ multiplication of a $K \times K$ matrix and a $K \times 1$ vector. The overall complexity is hence $O(K^3 + K^2\frac{N_1 N_3}{mn})$.

Since $S_1 \leq K$, the average decoding complexity for systematic RKRP codes is lower than that of OrthoPoly codes and the worst-case complexities (when $S_1 = K$) are identical.

4.7 Simulation results

We now present simulation results to demonstrate the superior numerical stability of RKRP codes. We performed Monte Carlo simulations of the encoding and decoding process by choosing the entries of $\mathbf{A}$ and $\mathbf{B}$ to be realizations of i.i.d Gaussian random variables with zero mean and unit variance. For the presented results, we have considered the recovery of the $(1,1)$th entry of $\mathbf{A}_j^T\mathbf{B}_l$. The corresponding vector $\underline{w}$ is then a realization of the vector-valued random variable $\underline{W}$. Then, we computed $\underline{y}$ using (4.8), (4.17), and (2.3) for non-systematic RKRP codes, systematic RKRP codes, and OrthoPoly codes, respectively. We randomly chose a subset of $N-K$ worker nodes to be stragglers. Then, we computed $\underline{\hat{w}}$ using (4.11), (4.23), (2.4) for non-systematic RKRP codes, systematic RKRP codes, and OrthoPoly codes, respectively. For each of these codes, we define the average relative error to be

$$\eta_{\text{ave}} := \mathbb{E}\left[\frac{||\underline{W} - \underline{\hat{W}}||_2}{||\underline{W}||_2}\right]$$

and we estimate η_{ave} from Monte Carlo simulations.

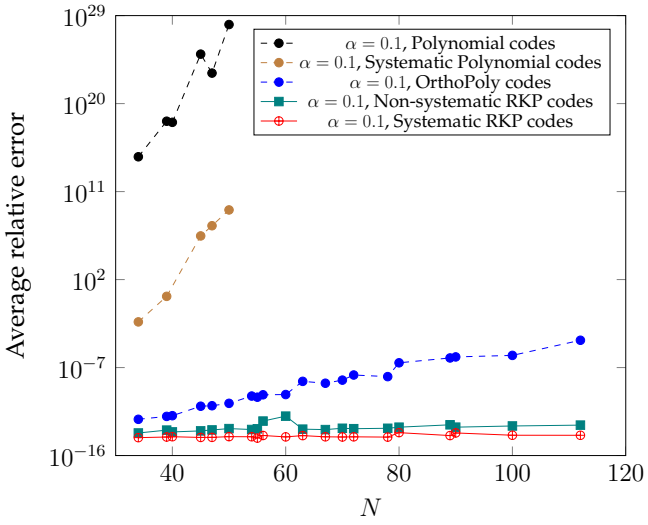

Figure 4.1: Plot of average relative error as a function of N for a fixed α; $N = \lceil K/(1-\alpha) \rceil$. Reprinted from [57]

4.7.1 MDS property

Firstly, in several million simulations, we never observed any instance where the generator matrix $\mathbf{G}$ in (4.8) or $\mathbf{G}_{sys}$ in (4.21) was singular, which provides empirical evidence to our claim that non-systematic and systematic RKRP codes are MDS codes with probability 1.

4.7.2 Average relative error

In Fig. 4.1, we plot the average relative error as a function of N when the total number of worker nodes is set to be $N = \lceil K/(1-\alpha) \rceil$ or $K = \lfloor N(1-\alpha) \rfloor$. This model is meaningful when we consider practical scenarios where each worker node fails with a fixed probability. In the plots in Fig. 4.1, α is fixed and K and N are varied. The results are shown for $\alpha = 0.1$ and for OrthoPoly codes, non-systematic RKRP codes, and systematic RKRP codes. It can be seen that the average relative error is several orders of magnitude lower for RKRP codes when N is about 100.

In Figure 4.2, we plot the average relative error versus $\alpha = \frac{N-K}{N}$ for a fixed K. Again, it can be seen that the proposed RKRP codes are very robust to numerical precision errors and substantially outperform OrthoPoly codes. It should also be noted that the aver-

age relative error remains largely independent of α for RKRP codes whereas they grow rapidly with α for OrthoPoly codes.

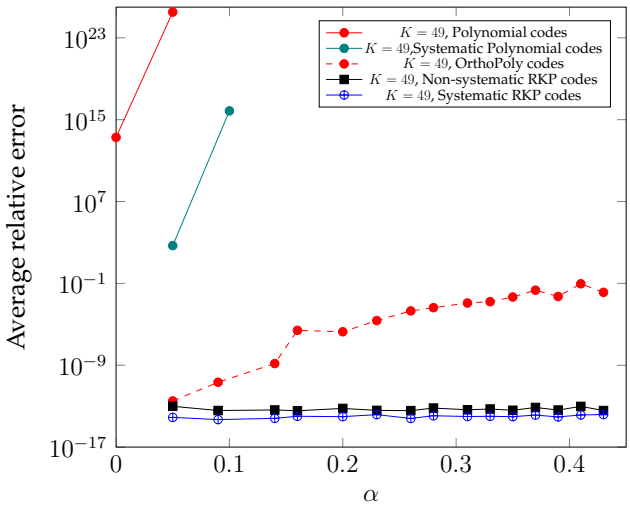

Figure 4.2: Plot of average relative error versus fraction of straggler nodes (α) for $K = 49$; $\alpha = \frac{N-K}{N}$. Reprinted from [57]

In Figure 4.3, we plot the average relative error versus the number of straggler nodes S for a fixed K when $N = K + S$. It can be seen that the proposed RKRP codes provide excellent robustness even as the number of stragglers increases.

4.7.3 Average log condition number

The expected value of the logarithm of the condition number of a random matrix is a measure of loss in precision in computing the inverse of the determinant of the matrix, when the matrix is chosen from an underlying ensemble [14]. We computed the expected value of the logarithm of the condition number of matrices from three ensembles. For non-systematic RKRP codes, we chose $\mathbf{G}$ from the $\mathcal{G}_{non-sys}(N, K, f)$ ensemble where f is a Gaussian density with zero mean and unit variance. For systematic RKRP codes, we chose $\mathbf{G}_{sys}$ from the $\mathcal{G}_{sys}(N, K, f)$ ensemble, and for OrthoPoly codes, we randomly chose $K \times K$ submatrices of $\mathbf{G}_O$ and multiplied the matrix by $\mathbf{H}$. In Figure 4.4, we plot the

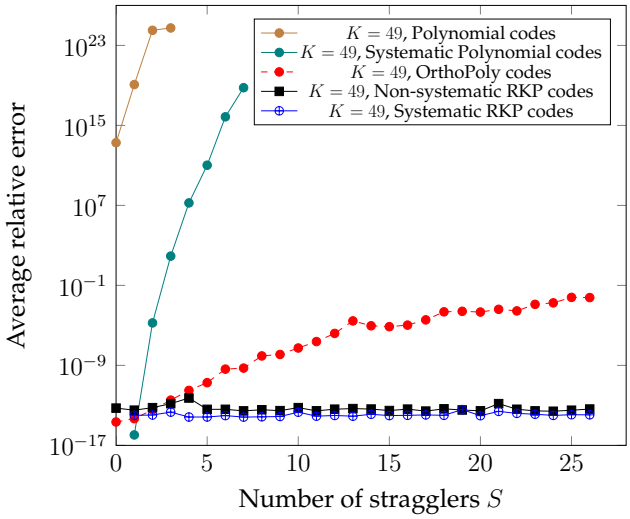

Figure 4.3: Plot of average relative error versus number of stragglers (S) for $K = 49$; $N = K + S$. Reprinted from [57]

average of the log of the condition number as a function of α for the three ensembles. We fix K and let $N = K(1 + \alpha)$.

It can be seen that the average of the log of the condition number is substantially lower for RKRP codes than for Orthopoly codes showing that the number of bits of precision lost is substantially lower for RKRP codes.

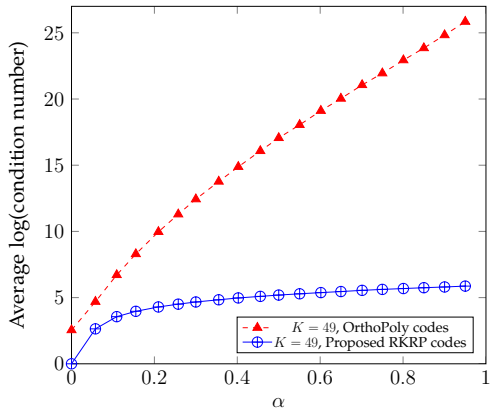

Figure 4.4: Plot of E[log(condition number)] of inverted matrix versus fraction of stragglers (α) for $K = 49$. Reprinted from [57]

4.8 Summary

We proposed a new class of codes called random Khatri-Rao-product (RKRP) codes for which the generator matrix is the row-wise Khatri-Rao product of two random matri-

ces. We proposed two random ensembles of generator matrices and corresponding codes called non-systematic RKRP codes and systematic RKRP codes. We showed that RKRP codes are maximum distance separable with probability 1 and that their decoding is substantially more numerically stable than Polynomial codes and OrthoPoly codes. The average decoding complexity of RKRP codes is lower than that of OrthoPoly codes.

5. PRODUCT LAGRANGE CODED COMPUTING[1]

5.1 Introduction

In this chapter, we consider the third problem introduced in section [1.3] of the Introduction chapter; the Distributed Multivariate Polynomial Evaluation (DMPE) problem which appears in several machine learning and deep learning algorithms—for instance, in computing the gradient of a loss-function on a big dataset in gradient descent/ascent optimization algorithms [53], or in decomposing low-rank tensors in high-dimensional optimization problems [46].

For any large-scale master-worker distributed computing scheme, robust algorithms need to address several important factors including (i) resilience to *straggling* workers, where each worker takes a random amount of time to conclude their task, and (ii) *scalability*—the numerical accuracy and implementation complexity of distributed algorithms need to scale efficiently with the number of workers.

Starting from the work of Lee *et al.* [27], there have been breakthrough developments in the design of distributed algorithms (mostly for distributed matrix-matrix multiplication) addressing the issue of resilience to stragglers, by leveraging ideas from coding theory. This has led to the paradigm of *coded distributed computing* [1,4,6–8,10,11,15–17,19,23–25, 28,31,34–37,39,40,42,44,45,47,48,50,54,56,59–62,64–66,69,70]. The use of codes based on polynomial-evaluation (particularly, Reed-Solomon codes over the real field) has played a central role in the design of coded distributed computing schemes. They have been shown to be optimal for distributed matrix-matrix multiplication in terms of the number of stragglers that can be tolerated [17,69,70]. The issue of scalability has been addressed in recent works [6,7,16,44,47,48,56].

Recently, Yu *et al.* in [66] introduced a coding-based scheme relying on polynomial-based

[1]The contents of this section will appear in verbatim as part of the paper "Product Lagrange Coded Computing" [58] at the International Symposium on Information Theory 2020 for which I was the first author

codes, called *Lagrange Coded Computing (LCC)*, for the DMPE problem. They showed that the LCC scheme provides optimal resilience to stragglers amongst all linear coding schemes for DMPE [66]. The theory of LCC is mathematically elegant and powerful and it provides an excellent solution to addresses the resilience issue for the DMPE problem. However, the LCC scheme of [66] is not highly scalable. Specifically, the decoding algorithm for LCC is *numerically unstable* as it involves explicitly or implicitly inverting a Vandermonde matrix and Vandermonde matrices over the real field are very poorly conditioned. In fact, the condition number of Vandermonde matrices grows exponentially in the size of the matrix [5, 43]. One possible approach to implement the LCC scheme of [66] is to use quantized inputs and perform computation over finite fields to prevent numerical overflow such as in [54]. However, a comprehensive study of how the loss in precision from quantization scales as a function of the degree of f, K and N is still not available in the literature.

5.1.1 Main Contributions

In this work, we propose a new variant of LCC, referred to as the *Product Lagrange Coded Computing (PLCC)*, which is inspired by and builds upon the LCC scheme [66] and product codes [38]. The PLCC scheme is more numerically stable than the LCC scheme, in the presence of numerical errors due to computing with finite precision. This advantage of PLCC comes at the price of sacrificing the optimality of LCC in terms of resilience to stragglers. That said, in many real-world applications dealing with large datasets, due to physical limitations it may be required to partition the dataset into many chunks (i.e., large K), and accordingly distribute the task of computation among many workers (i.e., large N). In such scenarios, PLCC can be implemented for cases with much larger K and N than those that can be handled by LCC, while providing a satisfactory level of accuracy.

5.2 Problem Formulation

Let $\mathbf{X}_1, \ldots, \mathbf{X}_K \in \mathbb{R}^{r \times d}$ be K matrices, each of size $r \times d$, and let $f : \mathbb{R}^{r \times d} \to \mathbb{R}^{a \times b}$ be a matrix function. In particular, let the matrix function f have the form $f(\mathbf{X}_k) = [f_{i,j}(\mathbf{X}_k)]_{1 \leq i \leq a, 1 \leq j \leq b}$, where $f_{i,j} : \mathbb{R}^{r \times d} \to \mathbb{R}$, and $f_{i,j}(\mathbf{X}_k)$ is a multivariate polynomial whose variables are the rd entries of the matrix $\mathbf{X}_k$. Let $\deg(f)$ be the maximum total degree of the multivariate polynomials $f_{i,j}$'s, where the *total degree* of $f_{i,j}$ is the maximum of degrees of all monomials in $f_{i,j}$.

In this work, we consider the problem of computing $f(\mathbf{X}_1), \ldots, f(\mathbf{X}_K)$ distributedly in a master-worker framework with one master node and N worker nodes. This problem is referred to as the *Distributed Multivariate Polynomial Evaluation (DMPE)*. For example, consider solving a linear least-squares regression problem using the gradient descent (GD) algorithm [66]. In each GD iteration, one needs to compute the function $f(\mathbf{X}) = \mathbf{X}^\mathsf{T}\mathbf{X}\mathbf{w}$, where $\mathbf{X} \in \mathbb{R}^{Kr \times d}$ is a feature/design matrix and $\mathbf{w} \in \mathbb{R}^{d \times 1}$ is a weight vector. Splitting $\mathbf{X}$ into K row-disjoint sub-matrices $\mathbf{X}_1, \ldots, \mathbf{X}_K \in \mathbb{R}^{r \times d}$, the problem of evaluating $f(\mathbf{X})$ becomes equivalent to computing the sum of the K evaluations $f(\mathbf{X}_1), \ldots, f(\mathbf{X}_K)$. Note that, in this example, the matrix function $f(\mathbf{X}_k) = [f_{1,1}(\mathbf{X}_k), \ldots, f_{d,1}(\mathbf{X}_k)]^\mathsf{T}$ has degree $\deg(f) = 2$, because $f_{i,1}(\mathbf{X}_k)$ for each $1 \leq i \leq d$ is a multivariate polynomial in the entries of the matrix $\mathbf{X}_k$ and has total degree 2. This is a multivariate polynomial evaluation problem, and computing the K evaluations $f(\mathbf{X}_1), \ldots, f(\mathbf{X}_K)$ distributedly is an instance of the DMPE problem.

We assume that S (out of N) worker nodes are stragglers, and that the computation results of stragglers are not received by the master node, i.e., they are erased. The *worst-case recovery threshold* of a DMPE scheme is defined as the minimum number of worker nodes (regardless of the configuration of the S stragglers) that the master node needs to wait for in order to guarantee recoverability of all K evaluations. The *average-case recovery threshold* of a DMPE scheme is defined as the average of the minimum number of worker nodes for which the master node needs to wait, where the average is taken over all possible

configurations of S stragglers (assuming that all configurations are equally likely). In addition, we measure the *numerical stability* of a DMPE scheme by the *relative error* — due to numerical round-off errors from computing with finite precision, in the estimated evaluation matrix $\widehat{\mathbf{Y}} = \widehat{f}(\mathbf{X})$ in comparison to the actual evaluation matrix $\mathbf{Y} = f(\mathbf{X})$, where the *relative error* is defined as $\frac{\|\widehat{\mathbf{Y}}-\mathbf{Y}\|}{\|\mathbf{Y}\|}$ where $\|\cdot\|$ denotes the L_2-norm.

In this work, we are interested in designing a DMPE scheme that is more numerically stable than the state-of-the-art LCC scheme.

5.3 Product Lagrange Coded Computing

In this section, we propose a new solution for the DMPE problem. The proposed scheme, referred to as the *Product Lagrange-Coded Computing (PLCC)*, is inspired by the LCC scheme [66] and product codes [38]. As will be discussed later, PLCC resolves the numerical instability issue associated with LCC to a great extent; this improvement however comes at a cost in terms of the recovery threshold.

In the following, we will explain the construction of two-dimensional PLCC, based on two-dimensional product codes. The extension of the code construction to higher dimensional PLCC is straightforward, and hence omitted.

The main idea of the proposed construction is to design a product code whose component codes per row and column are Reed-Solomon codes with message polynomials of the form $f(u(z))$ or $f(v(z))$ where $u(z)$ and $v(z)$ are Lagrange interpolation polynomials. The key challenge in the design of such a product code is to guarantee the consistency between the components codes along the rows and the columns. To solve this problem, we propose to use two sets of Lagrange basis polynomials (defined shortly) and couple the Lagrange interpolation polynomials $u(z)$ and $v(z)$ using the two sets of Lagrange basis polynomials.

5.3.1 Encoding

Let $K = K_1K_2$ and $N = N_1N_2$ be such that $N_i > (K_i - 1)\deg(f) + 1$. For each $1 \leq k \leq K$, we rename $\mathbf{X}_k$ by $\mathbf{X}_{i,j}$ where $1 \leq i \leq K_1$ and $1 \leq j \leq K_2$ are the unique integers such that

$k = (i-1)K_2 + j$.

For example, for $K_1 = 3$ and $K_2 = 2$, we have

$$\begin{bmatrix} \mathbf{X}_1 & \mathbf{X}_2 \\ \mathbf{X}_3 & \mathbf{X}_4 \\ \mathbf{X}_5 & \mathbf{X}_6 \end{bmatrix} = \begin{bmatrix} \mathbf{X}_{1,1} & \mathbf{X}_{1,2} \\ \mathbf{X}_{2,1} & \mathbf{X}_{2,2} \\ \mathbf{X}_{3,1} & \mathbf{X}_{3,2} \end{bmatrix}.$$

We shall construct a two-dimensional PLCC scheme with parameters (N_1, N_2, K_1, K_2) based on a two-dimensional $(N_1, (K_1-1)\deg(f)+1) \times (N_2, (K_2-1)\deg(f)+1)$ product code, represented by an $N_1 \times N_2$ matrix $\mathbf{C} = [\mathbf{C}_{i,j}]_{1 \le i \le N_1, 1 \le j \le N_2}$ whose $N = N_1 N_2$ entries $\mathbf{C}_{i,j}$'s are the code symbols. The master node then requests each of the N worker nodes to compute one of the code symbols $\mathbf{C}_{i,j}$.

Let $\alpha_1, \ldots, \alpha_{N_2} \in \mathbb{R}$ and $\beta_1, \ldots, \beta_{N_1} \in \mathbb{R}$ be two (not necessarily disjoint) sets of distinct real numbers. For each $1 \le j \le K_2$ and each $1 \le i \le K_1$, we define Lagrange basis polynomials $l_j(z)$ and $m_i(z)$ given as follows:

$$l_j(z) = \prod_{\substack{1 \le k \le K_2 \\ k \ne j}} \frac{z - \alpha_k}{\alpha_j - \alpha_k}, \quad m_i(z) = \prod_{\substack{1 \le k \le K_1 \\ k \ne i}} \frac{z - \beta_k}{\beta_i - \beta_k}.$$

Note that the polynomials $l_j(z)$'s and $m_i(z)$'s have degree $K_2 - 1$ and $K_1 - 1$, respectively. For example, for $K_1 = 3$ and $K_2 = 2$, we have $l_1(z) = \frac{z-\alpha_2}{\alpha_1-\alpha_2}$ and $l_2(z) = \frac{z-\alpha_1}{\alpha_2-\alpha_1}$, each of degree $K_2 - 1 = 1$, and we have $m_1(z) = \frac{(z-\beta_2)(z-\beta_3)}{(\beta_1-\beta_2)(\beta_1-\beta_3)}$, $m_2(z) = \frac{(z-\beta_1)(z-\beta_3)}{(\beta_2-\beta_1)(\beta_2-\beta_3)}$, and $m_3(z) = \frac{(z-\beta_1)(z-\beta_2)}{(\beta_3-\beta_1)(\beta_3-\beta_2)}$, each of degree $K_1 - 1 = 2$.

Using the Lagrange basis polynomials $l_j(z)$'s and $m_i(z)$'s, we define the code symbols $\mathbf{C}_{i,j}$'s as follows:

(i) For $1 \le i \le K_1$ and $1 \le j \le K_2$, we set $\mathbf{C}_{i,j} = f(\mathbf{X}_{i,j})$.

(ii) For $1 \le i \le K_1$, we associate a Lagrange interpolation polynomial $u_i(z) = \sum_{j=1}^{K_2} l_j(z)\mathbf{X}_{i,j}$ of degree $K_2 - 1$ to the ith row of matrix $\mathbf{C}$. For $1 \le i \le K_1$ and

$$\begin{bmatrix} f(\mathbf{X}_{1,1}) & f(\mathbf{X}_{1,2}) & f(u_1(\alpha_3)) = f(v_3(\beta_1)) & f(u_1(\alpha_4)) = f(v_4(\beta_1)) \\ f(\mathbf{X}_{2,1}) & f(\mathbf{X}_{2,2}) & f(u_2(\alpha_3)) = f(v_3(\beta_2)) & f(u_2(\alpha_4)) = f(v_4(\beta_2)) \\ f(\mathbf{X}_{3,1}) & f(\mathbf{X}_{3,2}) & f(u_3(\alpha_3)) = f(v_3(\beta_3)) & f(u_3(\alpha_4)) = f(v_4(\beta_3)) \\ f(u_4(\alpha_1)) = f(v_1(\beta_4)) & f(u_4(\alpha_2)) = f(v_2(\beta_4)) & f(u_4(\alpha_3)) = f(v_3(\beta_4)) & f(u_4(\alpha_4)) = f(v_4(\beta_4)) \\ f(u_5(\alpha_1)) = f(v_1(\beta_5)) & f(u_5(\alpha_2)) = f(v_2(\beta_5)) & f(u_5(\alpha_3)) = f(v_3(\beta_5)) & f(u_5(\alpha_4)) = f(v_4(\beta_5)) \\ f(u_6(\alpha_1)) = f(v_1(\beta_6)) & f(u_6(\alpha_2)) = f(v_2(\beta_6)) & f(u_6(\alpha_3)) = f(v_3(\beta_6)) & f(u_6(\alpha_4)) = f(v_4(\beta_6)) \end{bmatrix} \tag{5.1}$$

$K_2 < j \leq N_2$, we set $\mathbf{C}_{i,j} = f(u_i(\alpha_j))$. (Note that for $1 \leq i \leq K_1$ and $1 \leq j \leq K_2$, we have $u_i(\alpha_j) = \mathbf{X}_{i,j}$.)

(iii) For $1 \leq j \leq K_2$, we associate a Lagrange interpolation polynomial $v_j(z) = \sum_{i=1}^{K_1} m_i(z)\mathbf{X}_{i,j}$ of degree $K_1 - 1$ to the jth column of matrix $\mathbf{C}$. For $K_1 < i \leq N_1$ and $1 \leq j \leq K_2$, we set $\mathbf{C}_{i,j} = f(v_j(\beta_i))$. (Note that for $1 \leq i \leq K_1$ and $1 \leq j \leq K_2$, we have $v_j(\beta_i) = \mathbf{X}_{i,j}$.)

(iv) For $K_1 < i \leq N_1$ and $K_2 < j \leq N_2$, we associate a Lagrange interpolation polynomial $u_i(z) = \sum_{j=1}^{K_2} l_j(z)v_j(\beta_i)$ of degree $K_2 - 1$ to the ith row of matrix $\mathbf{C}$ and a Lagrange interpolation polynomial $v_j(z) = \sum_{i=1}^{K_1} m_i(z)u_i(\alpha_j)$ of degree $K_1 - 1$ to the jth column of matrix $\mathbf{C}$. (The evaluations $u_1(\alpha_j), \ldots, u_{K_1}(\alpha_j)$ required for computing $v_j(z)$ and the evaluations $v_1(\beta_i), \ldots, v_{K_2}(\beta_i)$ required for computing $u_i(z)$ are defined in (ii) and (iii), respectively.) The key feature of this construction is that $u_i(\alpha_j) = v_j(\beta_i)$ for all $1 \leq i \leq N_1$ and all $1 \leq j \leq N_2$ (see Lemma 21). This allows us to set $\mathbf{C}_{i,j} = f(u_i(\alpha_j)) = f(v_j(\beta_i))$ for all $K_1 < i \leq N_1$ and all $K_2 < j \leq N_2$.

Lemma 21. *For $1 \leq i \leq N_1$ and $1 \leq j \leq N_2$, we have*

$$u_i(\alpha_j) = v_j(\beta_i).$$

Proof: Recall that by the definition, we have

$$u_i(z) = \begin{cases} \sum_{k=1}^{K_2} l_k(z)\mathbf{X}_{i,k}, & 1 \leq i \leq K_1, \\ \sum_{k=1}^{K_2} l_k(z) \sum_{h=1}^{K_1} m_h(\beta_i)\mathbf{X}_{h,k}, & K_1 < i \leq N_1, \end{cases}$$

and

$$v_j(z) = \begin{cases} \sum_{h=1}^{K_1} m_h(z)\mathbf{X}_{h,j}, & 1 \le j \le K_2, \\ \sum_{h=1}^{K_1} m_h(z) \sum_{k=1}^{K_2} l_k(\alpha_j)\mathbf{X}_{h,k}, & K_2 < j \le N_2. \end{cases}$$

We prove the result of the lemma for the following four different cases (depending on i, j) separately.

- For $1 \le i \le K_1$ and $1 \le j \le K_2$, we have

$$\begin{aligned} u_i(\alpha_j) &= \sum_{\substack{1 \le k \le K_2 \\ k \ne j}} l_k(\alpha_j)\mathbf{X}_{i,k} + l_j(\alpha_j)\mathbf{X}_{i,j} \\ &= \sum_{\substack{1 \le k \le K_2 \\ k \ne j}} 0 \cdot \mathbf{X}_{i,k} + 1 \cdot \mathbf{X}_{i,j} = \mathbf{X}_{i,j}, \end{aligned}$$

  and

$$\begin{aligned} v_j(\beta_i) &= \sum_{\substack{1 \le h \le K_1 \\ h \ne i}} m_h(\beta_i)\mathbf{X}_{h,j} + m_i(\beta_i)\mathbf{X}_{i,j} \\ &= \sum_{\substack{1 \le h \le K_1 \\ h \ne i}} 0 \cdot \mathbf{X}_{h,j} + 1 \cdot \mathbf{X}_{i,j} = \mathbf{X}_{i,j}, \end{aligned}$$

  noting that for $1 \le k \le K_2$ and $1 \le h \le K_1$, we have

$$l_k(\alpha_j) = \begin{cases} 0, & k \ne j, \\ 1, & k = j, \end{cases} \quad \text{and} \quad m_h(\beta_i) = \begin{cases} 0, & h \ne i, \\ 1, & h = i. \end{cases}$$

- For $1 \le i \le K_1$ and $K_2 < j \le N_2$, we have

$$u_i(\alpha_j) = \sum_{1 \le k \le K_2} l_k(\alpha_j)\mathbf{X}_{i,k},$$

and

$$
\begin{aligned}
v_j(\beta_i) &= \sum_{1\le h\le K_1} m_h(\beta_i) \sum_{1\le k\le K_2} l_k(\alpha_j)\mathbf{X}_{h,k} \\
&= \sum_{\substack{1\le h\le K_1 \\ h\neq i}} m_h(\beta_i) \sum_{1\le k\le K_2} l_k(\alpha_j)\mathbf{X}_{h,k} \\
&\quad + m_i(\beta_i) \sum_{1\le k\le K_2} l_k(\alpha_j)\mathbf{X}_{i,k} \\
&= \sum_{\substack{1\le h\le K_1 \\ h\neq i}} 0 \cdot \sum_{1\le k\le K_2} l_k(\alpha_j)\mathbf{X}_{h,k} \\
&\quad + 1 \cdot \sum_{1\le k\le K_2} l_k(\alpha_j)\mathbf{X}_{i,k} \\
&= \sum_{1\le k\le K_2} l_k(\alpha_j)\mathbf{X}_{i,k}.
\end{aligned}
$$

- For $K_1 < i \le N_1$ and $1 \le j \le K_2$, we have

$$
\begin{aligned}
u_i(\alpha_j) &= \sum_{1\le k\le K_2} l_k(\alpha_j) \sum_{1\le h\le K_1} m_h(\beta_i)\mathbf{X}_{h,k} \\
&= \sum_{\substack{1\le k\le K_2 \\ k\neq j}} l_k(\alpha_j) \sum_{1\le h\le K_1} m_h(\beta_i)\mathbf{X}_{h,k} \\
&\quad + l_j(\alpha_j) \sum_{1\le h\le K_1} m_h(\beta_i)\mathbf{X}_{h,j} \\
&= \sum_{\substack{1\le k\le K_2 \\ k\neq j}} 0 \cdot \sum_{1\le h\le K_1} m_h(\beta_i)\mathbf{X}_{h,k} \\
&\quad + 1 \cdot \sum_{1\le h\le K_1} m_h(\beta_i)\mathbf{X}_{h,j} \\
&= \sum_{1\le h\le K_1} m_h(\beta_i)\mathbf{X}_{h,j},
\end{aligned}
$$

and

$$v_j(\beta_i) = \sum_{1 \leq h \leq K_1} m_h(\beta_i) \mathbf{X}_{h,j}.$$

- For $K_1 < i \leq N_1$ and $K_2 < j \leq N_2$, we have

$$\begin{aligned} u_i(\alpha_j) &= \sum_{1 \leq k \leq K_2} l_k(\alpha_j) \sum_{1 \leq h \leq K_1} m_h(\beta_i) \mathbf{X}_{h,k} \\ &= \sum_{1 \leq k \leq K_2} \sum_{1 \leq h \leq K_1} l_k(\alpha_j) m_h(\beta_i) \mathbf{X}_{h,k}, \end{aligned}$$

and

$$\begin{aligned} v_j(\beta_i) &= \sum_{1 \leq h \leq K_1} m_h(\beta_i) \sum_{1 \leq k \leq K_2} l_k(\alpha_j) \mathbf{X}_{h,k} \\ &= \sum_{1 \leq h \leq K_1} \sum_{1 \leq k \leq K_2} m_h(\beta_i) l_k(\alpha_j) \mathbf{X}_{h,k}. \end{aligned}$$

□

For example, consider a PLCC with parameters $N_1 = 6$, $N_2 = 4$, $K_1 = 3$, and $K_2 = 2$. The code matrix $\mathbf{C} = [\mathbf{C}_{i,j}]_{1 \leq i \leq 6, 1 \leq j \leq 4}$ for this example is given in (5.1). For instance, in this example we have $v_3(\beta_1) = m_1(\beta_1)u_1(\alpha_3) + m_2(\beta_1)u_2(\alpha_3) + m_3(\beta_1)u_3(\alpha_3) = 1 \cdot u_1(\alpha_3) + 0 \cdot u_2(\alpha_3) + 0 \cdot u_3(\alpha_3) = u_1(\alpha_3)$; and $u_4(\alpha_1) = l_1(\alpha_1)v_1(\beta_4) + l_2(\alpha_1)v_2(\beta_4) = 1 \cdot v_1(\beta_4) + 0 \cdot v_2(\beta_4) = v_1(\beta_4)$.

Theorem 22. *The proposed code construction for a two-dimensional PLCC scheme with parameters* (N_1, N_2, K_1, K_2) *yields a two-dimensional* $(N_1, (K_1 - 1)\deg(f) + 1) \times (N_2, (K_2 - 1)\deg(f) + 1)$ *product code.*

Proof: The result follows from the following two facts: (i) for each $1 \leq j \leq N_2$, the code symbols $\mathbf{C}_{1,j} = f(v_j(\beta_1)), \ldots, \mathbf{C}_{N_1,j} = f(v_j(\beta_{N_1}))$ along the jth column of matrix $\mathbf{C}$ are N_1 evaluations of the univariate polynomial $f(v_j(z))$ of degree $(K_1 - 1)\deg(f)$ at points $z \in$

$\{\beta_1, \ldots, \beta_{N_1}\}$; and (ii) for each $1 \leq i \leq N_1$, the code symbols $\mathbf{C}_{i,1} = f(u_i(\alpha_1)), \ldots, \mathbf{C}_{i,N_2} = f(u_i(\alpha_{N_2}))$ along the ith row of matrix $\mathbf{C}$ are N_2 evaluations of the univariate polynomial $f(u_i(z))$ of degree $(K_2 - 1)\deg(f)$ at points $z \in \{\alpha_1, \ldots, \alpha_{N_2}\}$. That is, each column of matrix $\mathbf{C}$ is a codeword of an $(N_1, (K_1-1)\deg(f)+1)$ Reed-Solomon code with parameters $\beta_1, \ldots, \beta_{N_1}$; and each row of matrix $\mathbf{C}$ is a codeword of an $(N_2, (K_2 - 1)\deg(f) + 1)$ Reed-Solomon code with parameters $\alpha_1, \ldots, \alpha_{N_2}$. □

5.3.2 Decoding

The goal of the decoding is to recover $f(\mathbf{X}_{1,1}), \ldots, f(\mathbf{X}_{K_1,K_2})$ from the code symbols received by the master node. This can be done similar to the decoding of product codes, i.e., by decoding the Reed-Solomon codes along rows and columns combined with an iterative peeling decoding algorithm. More specifically, given the code symbols computed by any $(K_1 - 1)\deg(f) + 1$ worker nodes along a column or the code symbols computed by any $(K_2 - 1)\deg(f) + 1$ worker nodes along a row, the master node is able to recover all other code symbols in that row or column, respectively.

Theorem 23. *The (worst-case) recovery threshold of a two-dimensional PLCC scheme with parameters (N_1, N_2, K_1, K_2) is*

$$N_1N_2 - (N_1 - (K_1 - 1)\deg(f) + 1)(N_2 - (K_2 - 1)\deg(f) + 1).$$

In particular, for $K_1 = K_2 = \sqrt{K}$ and $N_1 = N_2 = \sqrt{N}$, the (worst-case) recovery threshold is

$$N - (\sqrt{N} - (\sqrt{K} - 1)\deg(f) + 1)^2.$$

Proof: The result follows directly from the (worst-case) erasure decoding guarantee of a two-dimensional $(N_1, (K_1 - 1)\deg(f) + 1) \times (N_2, (K_2 - 1)\deg(f) + 1)$ product code [38], and hence omitted. □

5.4 Numerical Stability vs. Recovery Threshold

The main advantage of PLCC over LCC is its numerical stability. When decoding a Reed-Solomon code along a column or a row in PLCC, the master node needs to interpolate a polynomial of degree $(K_1 - 1)\deg(f)$ or $(K_2 - 1)\deg(f)$, respectively. On the other hand, the decoding of LCC requires the master node to interpolate a polynomial of degree $(K - 1)\deg(f) = (K_1K_2 - 1)\deg(f)$, which can be significantly larger than $(K_1 - 1)\deg(f)$ and $(K_2 - 1)\deg(f)$. This implies that the decoding of LCC involves inverting a Vandemonde matrix of substantially larger size, namely $((K-1)\deg(f)+1) \times ((K-1)\deg(f)+1)$, and hence far less numerically stable. For example, when $\deg(f) = 2$ and $K = 100$, for the LCC scheme the master node needs to invert a 199×199 Vandermonde matrix; whereas for the PLCC scheme with $K_1 = K_2 = \sqrt{K} = 10$, Vandermonde matrices of much smaller size 19×19 need to be inverted.

On the other hand, a simple comparison shows the superiority of LCC over PLCC in terms of the worst-case recovery threshold. However, depending on which worker nodes are straggling the master node may still be able to successfully recover all $f(\mathbf{X}_{i,j})$'s from the results of less than $N_1N_2 - (N_1 - (K_1 - 1)\deg(f) + 1)(N_2 - (K_2 - 1)\deg(f) + 1)$ worker nodes. Thus, the average-case recovery threshold of PLCC can be lower than their worst-case recovery threshold.

5.5 Simulation Results

In this section, we present numerical results to show that PLCC has better numerical stability than that of LCC. To illustrate this, we consider the computation of degree two polynomial $\mathbf{Y} = \mathbf{X}^\mathsf{T}\mathbf{X}$ distributedly, where $\mathbf{X}$ is a matrix whose entries are the realizations of a Gaussian random variable with zero mean and unit variance. To measure the numerical stability of these two schemes, we consider the *relative error*, defined as $\eta \triangleq \frac{||\mathbf{Y}-\widehat{\mathbf{Y}}||}{||\mathbf{Y}||}$, where $\widehat{\mathbf{Y}}$ is the estimate of $\mathbf{Y}$. We performed Monte Carlo simulations of encoding and decoding process of LCC, described in Section 2.3, and PLCC, described in Section 5.3, to estimate

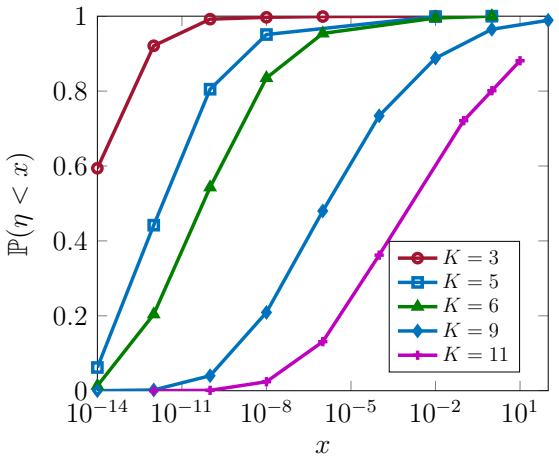

Figure 5.1: The empirical CDF of the relative error in LCC for different values of K and $N = 2K-1$, where there are no stragglers. Reprinted from [58]

η.

In Fig. 5.1, we plot the empirical cumulative distribution function (CDF) of the relative error, $\mathbb{P}(\eta < x)$, in LCC for different values of K. Notice that $\mathbb{P}(\eta < x)$ depends both on the probability of the iterative decoding process being successful and the relative error from numerical precision when the decoding is successful. It can be seen that the probability with which the relative error exceeds a fixed value increases as K increases. For example, the relative error is greater than 10^{-4} with probability 0.36 for $K = 9$; whereas it is greater than 10^{-4} with probability 0.62.

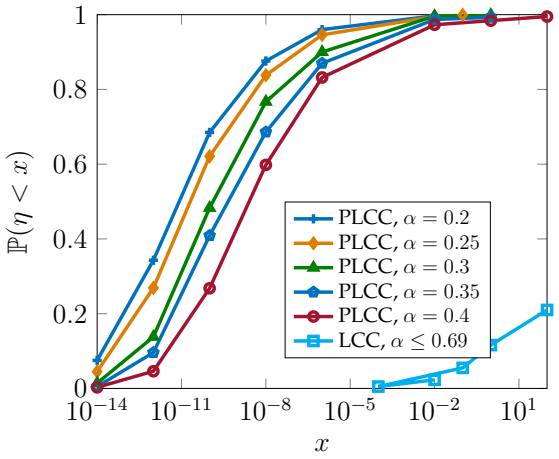

Figure 5.2: The empirical CDF of the relative error in PLCC and LCC for $K = 16$ and $N = 100$, where there are αN stragglers. Reprinted from [58]

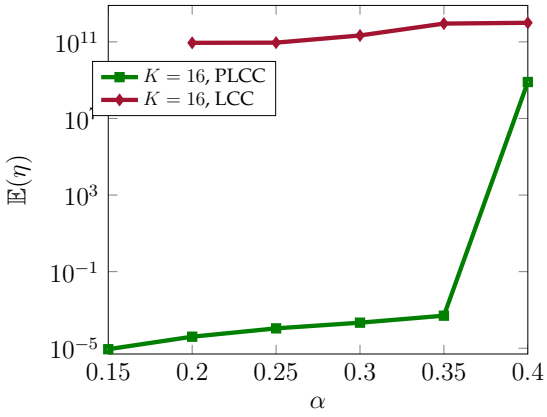

Figure 5.3: The average relative error in PLCC and LCC for $K = 16$ and $N = 100$, when there are αN stragglers. Reprinted from [58]

In Fig. 5.2, we plot the empirical CDF of the relative error, $\mathbb{P}(\eta < x)$, for a fixed K when α fraction of workers are stragglers. To simulate PLCC, we chose $K_1 = K_2 = 4$ and $N_1 = N_2 = 10$. For comparison, we also simulate LCC for $K = 16$ and $N = 100$. It can be seen that the relative error of PLCC is much smaller than that of LCC, even though LCC is optimal in terms of the recovery threshold.

In Fig. 5.3, we plot the average relative error, $\mathbb{E}(\eta)$, for both LCC and PLCC as a function of α when $K = 16$. It can be seen that the average relative error of PLCC is several orders less than that of LCC when α is sufficiently small, i.e., $N(1 - \alpha)$ is not much larger than the recovery threshold of PLCC. For larger values of α when $N(1 - \alpha)$ is far greater than the recovery threshold of PLCC, the average relative error of both schemes are very large, and almost the same.

One option to improve the numerical stability of LCC is to quantize the inputs and embed the quantities inside a finite field as in [54]. In this case, the relative error is determined by the quantization used. The quantization required may be coarser with the degree of f and hence, this needs to be studied in more detail in future work.

6. CONCLUSION

In this thesis, we have introduced the problem of numerical instability of polynomial codes in the real field and have proposed various coding schemes to improve the stability of codes for distributed machine learning.

In chapter 2, we have shown that polynomial codes (and some related codes) used for distributed matrix multiplication are *interleaved* Reed-Solomon codes and, hence, can be collaboratively decoded. We consider a fault tolerant setup where t worker nodes return erroneous values. For an additive random Gaussian error model, we show that for all $t < N - K$, errors can be corrected with probability 1. Further, numerical results show that in the presence of additive errors, when L Reed-Solomon codes are collaboratively decoded, the numerical stability in recovering the error locator polynomial improves with increasing L.

In chapter 3, we have proposed a class of codes called random Khatri-Rao-Product (RKRP) codes for distributed matrix multiplication in the presence of stragglers. The main advantage of the proposed code is that decoding of RKRP codes is highly numerically stable in comparison to decoding of Polynomial codes [67] and decoding of the recently proposed OrthoPoly codes [18]. We have shown that RKRP codes are maximum distance separable with probability 1. The communication cost and encoding complexity for RKRP codes are identical to that of OrthoPoly codes and Polynomial codes and the average decoding complexity of RKRP codes is lower than that of OrthoPoly codes. Numerical results presented in this chapter, show that the average relative L_2-norm of the reconstruction error for RKRP codes is substantially better than that of OrthoPoly codes.

In chapter 4 we have considered the problem of distributed multivariate polynomial evaluation (DPME) using a master-worker framework, which was originally considered by Yu *et al.*, where Lagrange Coded Computing (LCC) was proposed as a coded computation scheme to provide resilience against stragglers for the DPME problem. In this chapter, we

have proposed a variant of the LCC scheme; termed Product Lagrange Coded Computing (PLCC), by combining ideas from classical product codes and LCC. PLCC was demonstrated to be more numerically stable than LCC; however, their resilience to stragglers is sub-optimal.

6.1 Open problems

1. In chapter 2, it has been shown that collaborative decoding can correct upto $N - K - 1$ errors under the Gaussian noise model. It remains to be seen if it possible to construct an encoding scheme that can correct upto $N - K - 1$ errors when the noise injected by the workers is adversarial.

2. In chapter 3, we have proposed RKRP codes which have extremely high decoding complexity compared to reed-solomon codes decoded with the Berlekamp-Massey(BM) decoder. It remains an open problem to construct codes that have similar accuracy characteristics to that of RKRP codes, but have the structure to be decoded in $O(N^2)$(N is the size of the code), which is the complexity of the BM decoder.

3. The PLCC codes of chapter 4 have superior numerical stability compared to LCC codes. But this is at the cost of optimality in resilience to stragglers. It remains an open problem to construct numerically stable codes that have higher resilience to stragglers for the distributed polynomial evaluation problem.

REFERENCES

[1] Tavor Baharav, Kangwook Lee, Orhan Ocal, and Kannan Ramchandran. Straggler-proofing massive-scale distributed matrix multiplication with d-dimensional product codes. *2018 IEEE International Symposium on Information Theory (ISIT)*, pages 1993–1997, 2018.

[2] Rawad Bitar, Parimal Parag, and Salim El Rouayheb. Minimizing latency for secure coded computing using secret sharing via staircase codes. *arXiv:1802.02640*, 2018.

[3] Emmanuel J Candès and Terence Tao. Decoding by linear programming. *IEEE Trans. on Info. Theory*, 51(12):4203–4215, 2005.

[4] Lingjiao Chen, Hongyi Wang, Zachary B. Charles, and Dimitris S. Papailiopoulos. DRACO: Byzantine-resilient distributed training via redundant gradients. In *ICML*, 2018.

[5] Z. Chen. Optimal real number codes for fault tolerant matrix operations. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, pages 1–10, Nov 2009.

[6] Anindya Bijoy Das and Aditya Ramamoorthy. Distributed matrix-vector multiplication: A convolutional coding approach. *2019 IEEE International Symposium on Information Theory (ISIT)*, pages 3022–3026, 2019.

[7] Anindya Bijoy Das, Aditya Ramamoorthy, and Namrata Vaswani. Random convolutional coding for robust and straggler resilient distributed matrix computation. *arXiv:1907.08064*, 2019.

[8] Anindya Bijoy Das, Li Tang, and Aditya Ramamoorthy. C3LES: Codes for coded computation that leverage stragglers. *2018 IEEE Information Theory Workshop (ITW)*, pages 1–5, 2018.

[9] J Dean and S Ghemawat. Mapreduce: Simplified data processing on large clusters. *Proc. 6th Symposium on Operating System Design and Implementation (OSDI)*, 2004.

[10] S. Dutta, V. Cadambe, and P. Grover. Short-Dot: Computing large linear transforms distributedly using coded short dot products. *IEEE Transactions on Information Theory*, 65(10):6171–6193, Oct 2019.

[11] Sanghamitra Dutta, Ziqian Bai, Haewon Jeong, Tze Meng Low, and Pulkit Grover. A unified coded deep neural network training strategy based on generalized PolyDot codes. *2018 IEEE International Symposium on Information Theory (ISIT)*, pages 1585–1589, 2018.

[12] Sanghamitra Dutta, Ziqian Bai, Haewon Jeong, Tze Meng Low, and Pulkit Grover. A unified coded deep neural network training strategy based on generalized PolyDot codes for matrix multiplication. *arXiv:1811.10751*, 2018.

[13] Sanghamitra Dutta, Viveck R. Cadambe, and Pulkit Grover. "short-dot": Computing large linear transforms distributedly using coded short dot products. *arXiv:1704.05181*, 2017.

[14] Alan Edelman. Eigenvalues and condition numbers of random matrices. *SIAM Journal on Matrix Analysis and Applications*, 9(4):543–560, 1988.

[15] Yahya H. Ezzeldin, Mohammed Karmoose, and Christina Fragouli. Communication vs distributed computation: An alternative trade-off curve. *2017 IEEE Information Theory Workshop (ITW)*, pages 279–283, 2017.

[16] M. Fahim and V. R. Cadambe. Numerically stable polynomially coded computing. In *2019 IEEE International Symposium on Information Theory (ISIT)*, pages 3017–3021, July 2019.

[17] M. Fahim, H. Jeong, F. Haddadpour, S. Dutta, V. Cadambe, and P. Grover. On the optimal recovery threshold of coded matrix multiplication. In *2017 55th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pages 1264–1270, Oct 2017.

[18] Mohammed Fahim and Viveck Cadambe. Numerically stable polynomially coded computing. In *to appear in proceedings of the International Symposium on Information Theory*, 2019.

[19] Vipul Gupta, Swanand Kadhe, Thomas A. Courtade, Michael W. Mahoney, and Kannan Ramchandran. OverSketched Newton: Fast convex optimization for serverless systems. *arXiv:1903.08857*, 2019.

[20] Venkatesan Guruswami, Rudra Atri, and Madhu Sudan. Essential coding theory. *https://cse.buffalo.edu/faculty/atri/courses/coding-theory/book/web-coding-book.pdf*, 2018.

[21] Wael Halbawi, Navid Azizan, Fariborz Salehi, and Babak Hassibi. Improving distributed gradient descent using Reed-Solomon codes. *2018 IEEE International Symposium on Information Theory (ISIT)*, pages 2027–2031, 2018.

[22] Tao Jiang, Nicholas D Sidiropoulos, and Jos MF ten Berge. Almost-sure identifiability of multidimensional harmonic retrieval. *IEEE Transactions on Signal Processing*, 49(9):1849–1859, 2001.

[23] Can Karakus, Yifan Sun, Suhas Diggavi, and Wotao Yin. Straggler mitigation in distributed optimization through data encoding. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, pages 5440–5448, 2017.

[24] Shahrzad Kiani, Nuwan S. Ferdinand, and Stark C. Draper. Exploitation of stragglers in coded computation. *2018 IEEE International Symposium on Information Theory (ISIT)*, pages 1988–1992, 2018.

[25] Konstantinos Konstantinidis and Aditya Ramamoorthy. Leveraging coding techniques for speeding up distributed computing. *2018 IEEE Global Communications Conference (GLOBECOM)*, pages 1–6, 2018.

[26] V Yu Krachkovsky and Yuan Xing Lee. Decoding for iterative Reed-Solomon coding schemes. *IEEE Trans. on Magnetics*, 33(5):2740–2742, 1997.

[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] K. Lee, C. Suh, and K. Ramchandran. High-dimensional coded matrix multiplication. In *2017 IEEE International Symposium on Information Theory (ISIT)*, pages 2418–2422, June 2017.

[29] K. Lee, C. Suh, and K. Ramchandran. High-dimensional coded matrix multiplication. In *2017 IEEE International Symposium on Information Theory (ISIT)*, pages 2418–2422, June 2017.

[30] Kangwook Lee, Maximilian Lam, Ramtin Pedarsani, Dimitris S. Papailiopoulos, and Kannan Ramchandran. Speeding up distributed machine learning using codes. *arXiv:1512.02673*, 2015.

[31] S. Li, M. A. Maddah-Ali, and A. S. Avestimehr. Coded MapReduce. In *2015 53rd Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pages 964–971, Sep. 2015.

[32] S. Li, M. A. Maddah-Ali, and A. S. Avestimehr. Coded distributed computing: Straggling servers and multistage dataflows. In *2016 54th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pages 164–171, Sep. 2016.

[33] 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*, pages 1–6, Dec 2016.

[34] S. Li, M. A. Maddah-Ali, Q. Yu, and A. S. Avestimehr. A fundamental tradeoff between computation and communication in distributed computing. *IEEE Transactions on Information Theory*, 64(1):109–128, Jan 2018.

[35] S. Li, S. M. Mousavi Kalan, A. S. Avestimehr, and M. Soltanolkotabi. Near-optimal straggler mitigation for distributed gradient methods. In *2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 857–866, May 2018.

[36] S. Li, S. Supittayapornpong, M. A. Maddah-Ali, and S. Avestimehr. Coded terasort. In *2017 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 389–398, May 2017.

[37] S. Li, Q. Yu, M. A. Maddah-Ali, and A. S. Avestimehr. Coded distributed computing: Fundamental limits and practical challenges. In *2016 50th Asilomar Conference on Signals, Systems and Computers*, pages 509–513, Nov 2016.

[38] Shu Lin and Daniel J. Costello. *Error Control Coding, Second Edition*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2004.

[39] Raj Kumar Maity, Ankit Singh Rawat, and Arya Mazumdar. Robust gradient descent via moment encoding with LDPC codes. *arXiv:1805.08327*, 2018.

[40] A. Mallick, M. Chaudhari, and G. Joshi. Fast and efficient distributed matrix-vector multiplication using rateless fountain codes. In *2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 8192–8196, May 2019.

[41] Todd K Moon. Error correction coding. *Mathematical Methods and Algorithms. John Wiley and Son*, pages 2001–2006, 2005.

[42] H. A. Nodehi and M. A. Maddah-Ali. Limited-sharing multi-party computation for massive matrix operations. In *2018 IEEE International Symposium on Information Theory (ISIT)*, pages 1231–1235, June 2018.

[43] Victor Y. Pan. How bad are vandermonde matrices? *SIAM J. Matrix Analysis Applications*, 37:676–694, 2015.

[44] Asit Kumar Pradhan, Anoosheh Heidarzadeh, and Krishna R. Narayanan. Factored LT and factored Raptor codes for large-scale distributed matrix multiplication. *arXiv:1907.11018*, 2019.

[45] S. Prakash, A. Reisizadeh, R. Pedarsani, and S. Avestimehr. Coded computing for distributed graph analytics. In *2018 IEEE International Symposium on Information Theory (ISIT)*, pages 1221–1225, June 2018.

[46] Stephan Rabanser, Oleksandr Shchur, and Stephan Günnemann. Introduction to tensor decompositions and their applications in machine learning. *arXiv:1711.10781*, 2017.

[47] Aditya Ramamoorthy and Li Tang. Numerically stable coded matrix computations via circulant and rotation matrix embeddings. *arXiv:1910.06515*, 2019.

[48] Aditya Ramamoorthy, Li Tang, and Pascal O. Vontobel. Universally decodable matrices for distributed matrix-vector multiplication. *2019 IEEE International Symposium on Information Theory (ISIT)*, pages 1777–1781, 2019.

[49] Aditya Ramamoorthy, Li Tang, and Pascal O. Vontobel. Universally decodable matrices for distributed matrix-vector multiplication. *arXiv:1901.10674*, 2019.

[50] Netanel Raviv, Itzhak Tamo, Rashish Tandon, and Alexandros G. Dimakis. Gradient coding from cyclic MDS codes and expander graphs. In *ICML*, 2017.

[51] Georg Schmidt and Vladimir R Sidorenko. Linear shift-register synthesis for multiple sequences of varying length. *arXiv:cs/0605044*, 2006.

[52] Georg Schmidt, Vladimir R Sidorenko, and Martin Bossert. Collaborative decoding of interleaved Reed–Solomon codes and concatenated code designs. *IEEE Trans. on Info. Theory*, 55(7):2991–3012, 2009.

[53] Shai Shalev-Shwartz and Shai Ben-David. *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press, New York, NY, USA, 2014.

[54] Jinhyun So, Basak Guler, Amir Salman Avestimehr, and Payman Mohassel. CodedPrivateML: A fast and privacy-preserving framework for distributed machine learning. *IACR Cryptology ePrint Archive*, 2019:140, 2019.

[55] A Subramaniam, A Heiderzadeh, and K.R Narayanan. Collaborative decoding of polynomial codes. *arXiv:1905.13685*, 2019.

[56] Adarsh M. Subramaniam, Anoosheh Heidarzadeh, and Krishna Narayanan. Collaborative decoding of Polynomial codes for distributed computation. *arXiv:1905.13685*, 2019.

[57] Adarsh M. Subramaniam, Anoosheh Heidarzadeh, and Krishna R. Narayanan. Random Khatri-Rao-product codes for numerically-stable distributed matrix multiplication. *arxiv:1907.05965*, 2019.

[58] Adarsh M. Subramaniam, Anoosheh Heidarzadeh, Asit K. Pradhan, and Krishna R. Narayanan. Product lagrange coded computing. *International Symposium on Information Theory*, 2020.

[59] Rashish Tandon, Qi Lei, Alexandros G. Dimakis, and Nikos Karampatziakis. Gradient coding: Avoiding stragglers in distributed learning. In *Proceedings of the 34th International Conference on Machine Learning*, pages 3368–3376, Aug 2017.

[60] L. Tang, K. Konstantinidis, and A. Ramamoorthy. Erasure coding for distributed matrix multiplication for matrices with bounded entries. *IEEE Communications Letters*, 23(1):8–11, Jan 2019.

[61] Sinong Wang, Jiashang Liu, and Ness Shroff. Coded sparse matrix multiplication. In *Proceedings of the 35th International Conference on Machine Learning*, pages 5152–5160, 2018.

[62] Sinong Wang, Jiashang Liu, Ness B. Shroff, and Pengyu Yang. Fundamental limits of coded linear transform. *arXiv:1804.09791*, 2018.

[63] Yaoqing Yang, Pulkit Grover, and Soummya Kar. Coded distributed computing for inverse problems. In *Advances in Neural Information Processing Systems 30*, pages 709–719. 2017.

[64] Min Ye and Emmanuel Abbe. Communication-computation efficient gradient coding. *arXiv:1802.03475*, 2018.

[65] Q. Yu, S. Li, M. A. Maddah-Ali, and A. S. Avestimehr. How to optimally allocate resources for coded distributed computing? In *2017 IEEE International Conference on Communications (ICC)*, pages 1–7, May 2017.

[66] Qian Yu, Songze Li, Netanel Raviv, Seyed Mohammadreza Mousavi Kalan, Mahdi Soltanolkotabi, and Salman Avestimehr. Lagrange coded computing: Optimal design for resiliency, security and privacy. *arXiv:1806.00939*, 2018.

[67] Qian Yu, Mohammad Ali Maddah-Ali, and A Salman Avestimehr. Polynomial codes: An optimal design for high-dimensional coded matrix multiplication. *arXiv:1705.10464*, 2018.

[68] Qian Yu, Mohammad Ali Maddah-Ali, and A Salman Avestimehr. Straggler mitigation in distributed matrix multiplication: Fundamental limits and optimal coding. *arXiv:1801.07487*, 2018.

[69] Qian Yu, Mohammad Ali Maddah-Ali, and Amir Salman Avestimehr. Polynomial codes: An optimal design for high-dimensional coded matrix multiplication. In *NIPS*, 2017.

[70] Qian Yu, Mohammad Ali Maddah-Ali, and Amir Salman Avestimehr. Straggler mitigation in distributed matrix multiplication: Fundamental limits and optimal coding. *2018 IEEE International Symposium on Information Theory (ISIT)*, pages 2022–2026, 2018.

[71] Qian Yu, Netanel Raviv, Jinhyun So, and Amir Salman Avestimehr. Lagrange coded computing: Optimal design for resiliency, security and privacy. *arXiv:1806.00939*, 2018.

[72] M Zaharia, M Chowdhury, M.J Franklin, S Shenker, and I Stoica. Spark: Cluster computing with working sets. *Proc. 2nd USENIX Workshop on Hot Topics in Cloud Computing (HotCloud)*, 2010.