Matrix Multiplication with Straggler Tolerance in Coded Elastic Computing via Lagrange Code

Xi Zhong[1], Jörg Kliewer[2] and Mingyue Ji[1]
[1]*Department of Electrical and Computer Engineering*, *University of Utah*, Salt Lake City, UT, USA
Email: {xi.zhong, mingyue.ji}@utah.edu
[2]*Department of Electrical and Computer Engineering*, *New Jersey Institute of Technology*, Newark, NJ, USA
Email: jkliewer@njit.edu

***Abstract*—In cloud computing systems, elastic events and stragglers increase the uncertainty of the system, leading to computation delays. Coded elastic computing (CEC) introduced by Yang *et al.* in 2018 is a framework which mitigates the impact of elastic events using Maximum Distance Separable (MDS) coded storage. It proposed a CEC scheme for both matrix-vector multiplication and general matrix-matrix multiplication applications. However, in these applications, the proposed CEC scheme cannot tolerate stragglers due to the limitations imposed by MDS codes. In this paper we propose a new elastic computing scheme using uncoded storage and Lagrange coded computing approaches. The proposed scheme can effectively mitigate the effects of both elasticity and stragglers. Moreover, it produces a lower complexity and smaller recovery threshold compared to existing coded storage based schemes.**

I. Introduction

Coded distributed computing is an recently emerging paradigm to tackle the straggler effect by computation redundancy. Examples include [1]–[8] that use codes to deal with stragglers in applications such as matrix multiplications, where any subset of machines with cardinality larger than the recovery threshold can recover the matrix multiplication. Another important phenomenon that can degrade the performance of the cloud systems is the elastic events, i.e., some machines can be preempted and additional machines may be available, which presents new challenges in allocating computation tasks to available machines. Recently, coded elastic computing (CEC) was proposed by Yang *et al*. to mitigate the impact of elastic events [9]. In elastic computing, computations are performed on the same set of data over many time steps. Between each time step, once an elastic event occurs, the system knows immediately about which machines leave or join, so that current available machines can adjust their allocated computation tasks immediately. In [9] the idea is to assign computation tasks to available machines in a cyclic way using a Maximum Distance Separable (MDS) coded storage scheme. This CEC scheme can effectively mitigate the impact of elasticity and minimize the computing load when all machines have the same storage capacity and computation speed.

A CEC scheme can be divided into two parts: a coded computing scheme and a computation assignment. On the one hand, for designing a computation assignment the authors in [10] aim to maximize the overlap of the task assignments between computation time steps for homogeneous computing systems where the machines have the same computation speed. On the other hand, most coded computing schemes used in existing CEC schemes, including [9] and [11], focus on matrix-vector multiplication and use MDS codes. In [9] a CEC scheme for matrix-matrix multiplication was also proposed. This scheme cannot mitigate the effect of stragglers due to the limitations of MDS codes.

In this paper, we focus on straggler tolerance for matrix-matrix multiplication in CEC. We first present an application of Lagrange coded computing proposed in [6] which can mitigate the impact of stragglers. Compared to existing coding techniques for matrix partitioning that can be applied to CEC, this scheme has a better trade-off between computational complexity and recovery threshold. Then, we apply this scheme to CEC by combining it with an uncoded storage assignment. In the case of the cyclic assignment, we obtain a homogeneous CEC scheme with straggler tolerance. Compared to the CEC scheme for matrix-matrix multiplication in [9], our CEC scheme enables new features as straggler mitigation, an arbitrary matrix partition, a larger maximum number of machines, and a flexible recovery threshold.

Notation Convention: $[n]$ denotes the set $\{1, 2, \cdots, n\}$.

II. Problem Formulation

We consider a distributed network that consists of a master and N machines $[N] = \{1, 2, \cdots, N\}$. Given data matrix $A \in \mathbb{F}^{q\times v}$, in time step t the input matrix is $B_t \in \mathbb{F}^{v\times r}$ and a set of N_t available machines $\mathcal{N}_t \subseteq [N]$ aims to compute $C_t = AB_t$. A CEC scheme can be divided into four phases as follows:

1) In the *storage phase* prior to any computation, each machine stores a function of A. The size of storage normalized by the size of A is the storage ratio R.
2) In time step t, the input is B_t and there are N_t available machines $\mathcal{N}_t$. In the *communication phase*, the master selects a computation assignment that decides the computation load and specifies the allocated computation task of available machines in the later computing phase. According to this assignment, the master sends an a function of B_t to the available machines.
3) In the *computing phase*, each machine computes the allocated computation task and then sends the computation result to the master.

ICC 2023 - IEEE International Conference on Communications | 978-1-5386-7462-8/23/$31.00 ©2023 IEEE | DOI: 10.1109/ICC45041.2023.10279134

4) In the *decoding phase*, the master gathers the computation results from sufficient number of machines and decodes C_t. Define L as the recovery threshold, which is the minimum number of machines required for successful decoding.

In this paper we focus on one time step t, for simplicity of notation we denote B_t as B, C_t as C, and let $\mathcal{N}_t = [N_t]$.

Yang *et al*. proposed the first CEC scheme for matrix multiplication in [9], without straggler tolerance due to the limitations of MDS codes. In order to illustrate these limitations and to explain our motivation, we use the following example in [9].

Example 1: ([9]) When $[N] = \mathcal{N}_t = [4]$ and $L = 3$, the master computes $C = AB$ as the following steps.

1) In the storage phase, the master partitions A into 3 column blocks A_1, A_2, $A_3 \in \mathbb{F}^{q\times \frac{v}{3}}$ and selects a MDS generator matrix $X = [\boldsymbol{x}_1^T, \boldsymbol{x}_2^T, \boldsymbol{x}_3^T, \boldsymbol{x}_4^T]^T \in \mathbb{F}^{4\times 3}$, where $\boldsymbol{x}_i = (x_{i,1}, x_{i,2}, x_{i,3})$. Each machine $n \in [4]$ stores $\tilde{A}_n = \sum_{j\in[3]} x_{n,j}A_j$ and it is further partitioned into 4 groups of columns with equal size, i.e., $\tilde{A}_n = \{\tilde{A}_{n,1}, \tilde{A}_{n,2}, \tilde{A}_{n,3}, \tilde{A}_{n,4}\}$.
2) In the communication phase, the master partitions B into 3 row blocks B_1, B_2, $B_3 \in \mathbb{F}^{\frac{v}{3}\times r}$ and further partitions each B_j into 4 groups of rows with equal size, i.e., $B_j = \{B_{j,1}, B_{j,2}, B_{j,3}, B_{j,4}\}$. Next, define 4 groups of machines $\mathcal{P}_1 = \{1,3,4\}$, $\mathcal{P}_2 = \{1,2,4\}$, $\mathcal{P}_3 = \{1,2,3\}$, $\mathcal{P}_4 = \{2,3,4\}$, and define 4 matrices $H_k = (X_k{}^T)^{-1}$ for $k \in [4]$, where $X_k \in \mathbb{F}^{3\times 3}$ is a invertible matrix composed of vectors $\boldsymbol{x}_n$ for $n \in \mathcal{P}_k$. Then, for group $k \in [4]$ and the i-th assigned machine in $\mathcal{P}_k$, denoted by $\mathcal{P}_{k,i}$, the master encodes $\tilde{B}_{\mathcal{P}_{k,i},k} = \sum_{j\in[3]} H_{k,i,j}B_{j,k}$ and sends $\tilde{B}_{\mathcal{P}_{k,i},k}$ to machine $\mathcal{P}_{k,i}$, where $H_{k,i,j}$ is the value of entry (i,j) in matrix H_k.
3) In the computing phase, for group $k \in [4]$ the machine $\mathcal{P}_{k,i}$ computes $\tilde{A}_{\mathcal{P}_{k,i},k}\tilde{B}_{\mathcal{P}_{k,i},k}$ and sends it to the master.
4) In the decoding phase, $\sum_{j\in[3]} A_{j,k}B_{j,k}$, $k \in [4]$ is obtained by summing 3 computation results $\tilde{A}_{\mathcal{P}_{k,i},k}\tilde{B}_{\mathcal{P}_{k,i},k}$ from machines in $\mathcal{P}_k$. For example, when $k = 1$, the master decodes $\sum_{j\in[3]} A_{j,1}B_{j,1}$ by computing

$$\begin{aligned}
&\sum_{i\in[3]} \tilde{A}_{\mathcal{P}_{1,i},1}\tilde{B}_{\mathcal{P}_{1,i},1}\\
&= \tilde{A}_{1,1}\tilde{B}_{1,1} + \tilde{A}_{3,1}\tilde{B}_{3,1} + \tilde{A}_{4,1}\tilde{B}_{4,1}\\
&= [A_{1,1}, A_{2,1}, A_{3,1}]X_1^T H_1 [B_{1,1}^T, B_{2,1}^T, B_{3,1}^T]^T\\
&= [A_{1,1}, A_{2,1}, A_{3,1}]I[B_{1,1}^T, B_{2,1}^T, B_{3,1}^T]^T\\
&= A_{1,1}B_{1,1} + A_{2,1}B_{2,1} + A_{3,1}B_{3,1},
\end{aligned}$$

where I is an identity matrix. Therefore, for each group $k \in [4]$ the master repeats the decoding and recovers $AB = \sum_{k\in[4]}\sum_{j\in[3]} A_{j,k}B_{j,k}$.

From this example, we can see the limitation of straggler tolerance is derived from that 1) the encoding of B requires the master to assign exactly L machines for each group k, since H_k is determined by X_k using matrix invertibility; 2) the recovery threshold is L due to $X_k^T H_k = I$ in the decoding.

This observation motivates us to present a new CEC scheme such that for each group the master assigns $D \geq L$ machines and there are at least L machines completing their computing successfully, which results in straggler tolerance of $D - L$. We decompose the problem into two sub-problems: 1) giving a coded computing scheme with straggler tolerance, meaning that any set of L machines can recover the computation output C. 2) applying this scheme to CEC by combining it with a computation assignment.

III. Coded Computing Scheme via Lagrange Code

In this section, we present an uncoded storage coded computing scheme using the Lagrange code [6], such that any set of L machines can recover the computation output C.

A. An illustrating Example

Given two matrices $A \in \mathbb{F}^{q\times v}$ and $B \in \mathbb{F}^{v\times r}$, the master computes $C = AB$ with $N = 5$ machines. First, we divide each matrix into 3×3 sub-matrices as follows.

$$A = \begin{bmatrix} A_{1,1} & A_{1,2} & A_{1,3}\\ A_{2,1} & A_{2,2} & A_{2,3}\\ A_{3,1} & A_{3,2} & A_{3,3}\end{bmatrix}, B = \begin{bmatrix} B_{1,1} & B_{1,2} & B_{1,3}\\ B_{2,1} & B_{2,2} & B_{2,3}\\ B_{3,1} & B_{3,2} & B_{3,3}\end{bmatrix}. \quad (1)$$

In the storage phase, each machine stores the entire A. In the communication phase, the master selects 3 distinct numbers $\boldsymbol{\beta} = \{\beta_1, \beta_2, \beta_3\}$ in $\mathbb{F}$ and generates 3 polynomials

$$\begin{bmatrix} V_1(z)\\ V_2(z)\\ V_3(z)\end{bmatrix} = \begin{bmatrix} B_{1,1} & B_{1,2} & B_{1,3}\\ B_{2,1} & B_{2,2} & B_{2,3}\\ B_{3,1} & B_{3,2} & B_{3,3}\end{bmatrix} \cdot \begin{bmatrix} \prod_{k\in[3]\setminus\{1\}} \frac{z-\beta_k}{\beta_1-\beta_k}\\ \prod_{k\in[3]\setminus\{2\}} \frac{z-\beta_k}{\beta_2-\beta_k}\\ \prod_{k\in[3]\setminus\{3\}} \frac{z-\beta_k}{\beta_3-\beta_k}\end{bmatrix}.$$

Let $\mathcal{V}(z) = \{V_1(z), V_2(z), V_3(z)\}$. We notice that for $i \in [3]$ and $j \in [3]$, $V_i(\beta_j) = B_{i,j}$ holds. Then we have

$$\mathcal{V}(\beta_j) = \{B_{1,j}, B_{2,j}, B_{3,j}\}, \quad (2)$$

which is exactly the j-th column block in matrix B. The master then selects 5 distinct numbers $\boldsymbol{\alpha} = \{\alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_5\}$ in $\mathbb{F}$, such that $\boldsymbol{\alpha} \cap \boldsymbol{\beta} = \emptyset$. For each machine $j \in [5]$, the master obtains $\tilde{\mathcal{B}}_j = \mathcal{V}(\alpha_j) = \{V_1(\alpha_j), V_2(\alpha_j), V_3(\alpha_j)\} = \{\tilde{B}_{1,j}, \tilde{B}_{2,j}, \tilde{B}_{3,j}\}$ and sends $\tilde{\mathcal{B}}_j$ to machine j.

In the computing phase, machine $j \in [5]$ computes 3 functions $f_i(A, \tilde{\mathcal{B}}_j)$, $i \in [3]$ and sends the computation results to the master, where

$$\begin{aligned}
f_1(A, \tilde{\mathcal{B}}_j) &= A_{1,1}\tilde{B}_{1,j} + A_{1,2}\tilde{B}_{2,j} + A_{1,3}\tilde{B}_{3,j},\\
f_2(A, \tilde{\mathcal{B}}_j) &= A_{2,1}\tilde{B}_{1,j} + A_{2,2}\tilde{B}_{2,j} + A_{2,3}\tilde{B}_{3,j},\\
f_3(A, \tilde{\mathcal{B}}_j) &= A_{3,1}\tilde{B}_{1,j} + A_{3,2}\tilde{B}_{2,j} + A_{3,3}\tilde{B}_{3,j}.
\end{aligned}$$

In the decoding phase, we define 3 polynomials as follows.

$$\begin{aligned}
h_1(z) &= f_1(A, \mathcal{V}(z)) = A_{1,1}V_1(z) + A_{1,2}V_2(z) + A_{1,3}V_3(z),\\
h_2(z) &= f_2(A, \mathcal{V}(z)) = A_{2,1}V_1(z) + A_{2,2}V_2(z) + A_{2,3}V_3(z),\\
h_3(z) &= f_3(A, \mathcal{V}(z)) = A_{3,1}V_1(z) + A_{3,2}V_2(z) + A_{3,3}V_3(z).
\end{aligned}$$

For each polynomial $h_i(z)$, $i \in [3]$, we have two observations. On the one hand, from (2) we have $h_i(\beta_j) = f_i(A, \mathcal{V}(\beta_j)) = A_{i,1}B_{1,j} + A_{i,2}B_{2,j} + A_{i,3}B_{3,j}$ for all $j \in [3]$, which is the

block $C_{i,j}$, i.e., the product between the i-th row block of A and the j-th column block of B. On the other hand, each computation result $f_i(A, \tilde{\mathcal{B}}_j)$ from machine $j \in [5]$ is also an evaluation of the polynomial $h_i(z)$ on argument α_j since $h_i(\alpha_j) = f_i(A, \mathcal{V}(\alpha_j)) = f_i(A, \tilde{\mathcal{B}}_j)$. Based on above observations, decoding a block $C_{i,j}$, $i \in [3]$, $j \in [3]$ means to interpolate the polynomial $h_i(z)$ with degree $d = 2$ and to evaluate $h_i(\beta_j)$. That is, upon receiving computation results from any $d + 1 = 3$ machines denoted by $\mathcal{L}$, the master obtains $C_{i,j}$ by computing $h_i(\beta_j) = \sum_{k\in\mathcal{L}} f_i(A, \tilde{\mathcal{B}}_k) \cdot \prod_{k'\in\mathcal{L}\setminus\{k\}} \frac{\beta_j - \alpha_{k'}}{\alpha_k - \alpha_{k'}}$. Finally, the master recovers $C = AB$ in the following way.

$$C = \begin{bmatrix} h_1(\beta_1) & h_1(\beta_2) & h_1(\beta_3) \\ h_2(\beta_1) & h_2(\beta_2) & h_2(\beta_3) \\ h_3(\beta_1) & h_3(\beta_2) & h_3(\beta_3) \end{bmatrix}.$$

B. General Coded Computing Scheme

We divide matrix A into $m \times p$ sub-matrices and B into $p \times n$ sub-matrices as follows.

$$A = \begin{bmatrix} A_{1,1} & A_{1,2} & \cdots & A_{1,p} \\ A_{2,1} & A_{2,2} & \cdots & A_{2,p} \\ \vdots & \vdots & \ddots & \vdots \\ A_{m,1} & A_{m,2} & \cdots & A_{m,p} \end{bmatrix}, B = \begin{bmatrix} B_{1,1} & B_{1,2} & \cdots & B_{1,n} \\ B_{2,1} & B_{2,2} & \cdots & B_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ B_{p,1} & B_{p,2} & \cdots & B_{p,n} \end{bmatrix}. \tag{3}$$

Then the computation output C contains mn blocks $C_{i,j}$, $i \in [m]$, $j \in [n]$, which is the product between the i-th row block of A and the j-th column block of B.

In the storage phase, each machine stores the entire A. In the communication phase, the master selects n distinct numbers $\{\beta_i : i \in [n]\}$ in $\mathbb{F}$ and generates a set of p polynomials $\mathcal{V}(z) = \{V_i(z) : i \in [p]\}$ each of degree $n - 1$, where

$$V_i(z) = \sum_{j\in[n]} B_{i,j} \cdot \prod_{k\in[n]\setminus\{j\}} \frac{z - \beta_k}{\beta_j - \beta_k}. \tag{4}$$

For $i \in [p]$ and $j \in [n]$, $V_i(\beta_j) = B_{i,j}$ holds. Then we have

$$\mathcal{V}(\beta_j) = \{B_{1,j}, B_{2,j}, \cdots, B_{p,j}\}, \tag{5}$$

which is exactly the j-th column block of B. The master then selects N distinct numbers $\{\alpha_j : j \in [N]\}$ in $\mathbb{F}$ such that $\{\alpha_j : j \in [N]\} \cap \{\beta_i : i \in [n]\} = \emptyset$. For each machine $j \in [N]$, the master obtains $\tilde{\mathcal{B}}_j = \mathcal{V}(\alpha_j) = \{V_i(\alpha_j) : i \in [p]\} = \{\tilde{B}_{i,j} : i \in [p]\}$ and sends $\tilde{\mathcal{B}}_j$ to machine j.

In the computing phase, each machine $j \in [N]$ computes m functions $f_i(A, \tilde{\mathcal{B}}_j), i \in [m]$ and sends the computation results to the master, where

$$f_i(A, \tilde{\mathcal{B}}_j) = \sum_{k\in[p]} A_{i,k} \cdot \tilde{B}_{k,j}. \tag{6}$$

In the decoding phase, we define m polynomials $h_i(z), i \in [m]$ each of degree $n - 1$, where

$$h_i(z) = f_i(A, \mathcal{V}(z)) = \sum_{k\in[p]} A_{i,k} \cdot V_k(z), \tag{7}$$

such that for $j \in [n]$ we have $h_i(\beta_j) = f_i(A, \mathcal{V}(\beta_j)) = \sum_{k\in[p]} A_{i,k} \cdot V_k(\beta_j) = \sum_{k\in[p]} A_{i,k} \cdot B_{k,j} = C_{i,j}$ due to (5). Meanwhile, we have $h_i(\alpha_j) = f_i(A, \mathcal{V}(\alpha_j)) = f_i(A, \tilde{\mathcal{B}}_j)$ from machine $j \in [N]$. Hence, decoding $C_{i,j}$, $i \in [m]$, $j \in [n]$ means to interpolate the polynomial $h_i(z)$ in (7) by using computation results in (6) from any n machines denoted by $\mathcal{L} \subseteq [N]$, and to evaluate $h_i(\beta_j)$. That is, the master computes

$$C_{i,j} = h_i(\beta_j) = \sum_{k\in\mathcal{L}} f_i(A, \tilde{\mathcal{B}}_k) \cdot \prod_{k'\in\mathcal{L}\setminus\{k\}} \frac{\beta_j - \alpha_{k'}}{\alpha_k - \alpha_{k'}}. \tag{8}$$

Therefore, the master evaluates m polynomials $h_i(z)$, $i \in [m]$ on n arguments β_j, $j \in [n]$ to obtain mn blocks $C_{i,j}$. The computation output $C = AB$ is recovered.

IV. CEC Scheme via Lagrange code

We first provide the definition of computation assignment in CEC and then apply the coded computing scheme presented in Section III to CEC.

Definition 1: In time step t, there are N_t available machines $\mathcal{N}_t$. Given a coded computing scheme with recovery threshold L and straggler tolerance $D - L$, where $L \le D \le N_t$, $(D, G, \boldsymbol{\mu}, \boldsymbol{\gamma}, \boldsymbol{\mathcal{P}})$ is a computation assignment applying the coded computing scheme to CEC by the following steps:

1) Define a partition vector $\boldsymbol{\gamma} = (\gamma_1, \cdots, \gamma_G)$, where $\sum_{i\in[G]} \gamma_i = 1$. Partition the computation task of each machine into G disjoint groups. The proportion of the g-th group is γ_g.
2) Define G groups of machines $\boldsymbol{\mathcal{P}} = \{\mathcal{P}_1, \cdots, \mathcal{P}_G\}$ such that for the g-th group we have $\mathcal{P}_g \subseteq \mathcal{N}_t$ and $|\mathcal{P}_g| = D$.
3) In the computing phase, machines in $\mathcal{P}_g$, $g \in [G]$ compute the g-th group of computation task.
4) In the decoding phase, the master decodes a partial C for each group $g \in [G]$ so that it can recover C.

In computation load vector $\boldsymbol{\mu} = (\mu_1, \cdots, \mu_{N_t})$, where $\sum_{i\in[N_t]} \mu_i = D$, μ_i is the fraction of allocated computation task of the i-th machine in $\mathcal{N}_t$.

Remark 1: For homogeneous computing system, the cyclic assignment in [9] can be noted as $(D = L, G = N_t, \boldsymbol{\mu} = (\frac{L}{N_t}, \cdots, \frac{L}{N_t}), \boldsymbol{\gamma} = (\frac{1}{N_t}, \cdots, \frac{1}{N_t}), \boldsymbol{\mathcal{P}} = \{\mathcal{P}_g : g \in [N_t]\})$, where $\mathcal{P}_g = \{(g - L + 1)\%N_t, (g - L + 2)\%N_t, \cdots, g\%N_t\}$ and we define $a\%N_t \triangleq a + \lfloor \frac{N_t - a + 1}{N_t} \rfloor N_t$.

The key of applying the proposed coded computing scheme to CEC is to identify the computation task stated in step 1 in Definition 1 and to specify the partition of this computation task. In the coded computing scheme in Section III-B, the computation task of machine j is a set of m polynomials in (6), where each polynomial $f_i(A, \tilde{\mathcal{B}}_j)$ is the summation of p matrix multiplications of $A_{i,k} \in \mathbb{F}^{\frac{u}{m} \times \frac{v}{p}}$ and $\tilde{B}_{k,j} \in \mathbb{F}^{\frac{v}{p} \times \frac{r}{n}}$. Partitioning this computation task into G groups is equivalent to partitioning each polynomial $f_i(A, \tilde{\mathcal{B}}_j)$ into G groups symmetrically. To do this, according to the partition vector $\boldsymbol{\gamma}$ we divide each $\tilde{B}_{k,j}$ in (6) into G disjoint groups of columns, i.e., $\tilde{B}_{k,j} = \{(\tilde{B}_{k,j})_g \in \mathbb{F}^{\frac{v}{p} \times \frac{\gamma_g r}{n}} : g \in [G]\}$. Hence, polynomial $f_i(A, \tilde{\mathcal{B}}_j)$ is divided into G disjoint groups $\{f_{i,g}(A, \tilde{\mathcal{B}}_j) : g \in [G]\}$, where $f_{i,g}(A, \tilde{\mathcal{B}}_j) = \sum_{k\in[p]} A_{i,k} \cdot (\tilde{B}_{k,j})_g$. Then, the g-th group of computation task in machine j is $\{f_{i,g}(A, \tilde{\mathcal{B}}_j) : i \in [m]\}$, which contains the g-th groups

of blocks $\tilde{B}_{k,j}$, $k \in [p]$. Then for each machine j, selecting its allocated computation task is equivalent to selecting allocated groups in $\tilde{B}_{k,j}$, $k \in [p]$. Moreover, to reduce the complexity of encoding B and communication cost, in the communication phase the master can exactly encode and send the allocated groups in $\tilde{B}_{k,j}$, $k \in [p]$ to machine j, instead of sending $\tilde{B}_{k,j}$ and then making the machine to use the allocated groups.

To apply our coded computing scheme to CEC by using Definition 1, we first use the example in Section III-A by adding the cyclic assignment in Remark 1 for homogeneous systems. Then we present the general homogeneous CEC scheme[1] with straggler tolerance.

A. Applying the Example in Section III-A to CEC

Let $\mathcal{N}_t = [4]$ and the machine 5 is preempted. Prior to any computation, the master has partitioned A and B as shown in (1) and each machine has already stored the entire A.

In the communication phase, consider the cyclic assignment in Remark 1, i.e., $(D = L = 3,\ G = 4,\ \boldsymbol{\mu} = (\frac{3}{4}, \frac{3}{4}, \frac{3}{4}, \frac{3}{4}),\ \boldsymbol{\gamma} = (\frac{1}{4}, \frac{1}{4}, \frac{1}{4}, \frac{1}{4}),\ \boldsymbol{\mathcal{P}} = \{\mathcal{P}_g : g \in [4]\})$, where $\mathcal{P}_1 = \{1, 3, 4\}$, $\mathcal{P}_2 = \{1, 2, 4\}$, $\mathcal{P}_3 = \{1, 2, 3\}$, $\mathcal{P}_4 = \{2, 3, 4\}$. Each block $B_{i,j} \in \mathbb{F}^{\frac{v}{3} \times \frac{r}{3}}$ for $i, j \in [3]$ is further partitioned into 4 groups of columns with equal size, i.e., $(B_{i,j})_1$, $(B_{i,j})_2$, $(B_{i,j})_3$, $(B_{i,j})_4 \in \mathbb{F}^{\frac{v}{3} \times \frac{r}{12}}$. The master uses the set $\boldsymbol{\beta}$ in Section III-A to generate $Gp = 12$ polynomials as follows

$$\begin{bmatrix} V_{1,1}(z) \\ V_{2,1}(z) \\ V_{3,1}(z) \\ \hline V_{1,2}(z) \\ V_{2,2}(z) \\ V_{3,2}(z) \\ \hline V_{1,3}(z) \\ V_{2,3}(z) \\ V_{3,3}(z) \\ \hline V_{1,4}(z) \\ V_{2,4}(z) \\ V_{3,4}(z) \end{bmatrix} = \begin{bmatrix} (B_{1,1})_1 & (B_{1,2})_1 & (B_{1,3})_1 \\ (B_{2,1})_1 & (B_{2,2})_1 & (B_{2,3})_1 \\ (B_{3,1})_1 & (B_{3,2})_1 & (B_{3,3})_1 \\ \hline (B_{1,1})_2 & (B_{1,2})_2 & (B_{1,3})_2 \\ (B_{2,1})_2 & (B_{2,2})_2 & (B_{2,3})_2 \\ (B_{3,1})_2 & (B_{3,2})_2 & (B_{3,3})_2 \\ \hline (B_{1,1})_3 & (B_{1,2})_3 & (B_{1,3})_3 \\ (B_{2,1})_3 & (B_{2,2})_3 & (B_{2,3})_3 \\ (B_{3,1})_3 & (B_{3,2})_3 & (B_{3,3})_3 \\ \hline (B_{1,1})_4 & (B_{1,2})_4 & (B_{1,3})_4 \\ (B_{2,1})_4 & (B_{2,2})_4 & (B_{2,3})_4 \\ (B_{3,1})_4 & (B_{3,2})_4 & (B_{3,3})_4 \end{bmatrix} \cdot \begin{bmatrix} \prod_{k \in [3]\setminus\{1\}} \frac{z-\beta_k}{\beta_1-\beta_k} \\ \prod_{k \in [3]\setminus\{2\}} \frac{z-\beta_k}{\beta_2-\beta_k} \\ \prod_{k \in [3]\setminus\{3\}} \frac{z-\beta_k}{\beta_3-\beta_k} \end{bmatrix}. \tag{9}$$

Let $\mathcal{V}_g(z) = \{V_{1,g}(z), V_{2,g}(z), V_{3,g}(z)\}$ denote polynomials containing the g-th group of all blocks $B_{i,j}$. For $i \in [3]$, $j \in [3]$ and $g \in [4]$, we have $V_{i,g}(\beta_j) = (B_{i,j})_g$. Then we have

$$\mathcal{V}_g(\beta_j) = \{(B_{1,j})_g, (B_{2,j})_g, (B_{3,j})_g\}, \tag{10}$$

which is the g-th group of the j-th column block in B. Due to $\mathcal{N}_t = [4]$ the master selects numbers $\alpha_1, \alpha_2, \alpha_3, \alpha_4 \in \boldsymbol{\alpha}$ in Section III-A. According to the computation assignment, for group $g \in [4]$ and assigned machine $j \in \mathcal{P}_g$, the master computes and sends $\tilde{\mathcal{B}}_{g,j} = \mathcal{V}_g(\alpha_j) = \{V_{1,g}(\alpha_j), V_{2,g}(\alpha_j), V_{3,g}(\alpha_j)\} = \{\tilde{B}_{g,j,1}, \tilde{B}_{g,j,2}, \tilde{B}_{g,j,3}\}$ to machine j. For example, machine 1 is assigned to groups 1, 2 and 3 since $1 \in \mathcal{P}_1 \cap \mathcal{P}_2 \cap \mathcal{P}_3$. Then machine 1 receives $\tilde{\mathcal{B}}_{1,1}$ for group 1, receives $\tilde{\mathcal{B}}_{2,1}$ for group 2, and receives $\tilde{\mathcal{B}}_{3,1}$ for group 3.

[1]Our scheme also can be extended to CEC in heterogeneous computing systems, by adding the computation assignment obtained by the Algorithm 1 in [11]. While in this paper we focus on the homogeneous systems.

Compared to the previous example in Section III-A, where each machine $j \in [5]$ receives $\tilde{\mathcal{B}}_j = \{\tilde{B}_{k,j} : k \in [3]\}$, in this example machine $j \in \mathcal{P}_g$ receives $\tilde{\mathcal{B}}_{g,j} = \{\tilde{B}_{g,j,k} : k \in [3]\} = \{(\tilde{B}_{k,j})_g : k \in [3]\}$, i.e., the collection of the g-th groups of matrices in $\tilde{\mathcal{B}}_j$. Actually, $\tilde{\mathcal{B}}_j$ in the previous example can be partitioned into 4 groups $\tilde{\mathcal{B}}_j = \{\tilde{\mathcal{B}}_{1,j}, \tilde{\mathcal{B}}_{2,j}, \tilde{\mathcal{B}}_{3,j}, \tilde{\mathcal{B}}_{4,j}\}$, and in this example each machine only receives its assigned groups according to the computation assignment. the cyclic assignment is shown in Figure 1, where the j-th column corresponds to $\tilde{\mathcal{B}}_j$ and the g-th row corresponds to the g-th group of all $\tilde{\mathcal{B}}_j$ in the previous example. The yellow sub-blocks are the data received by the machines in this example. From

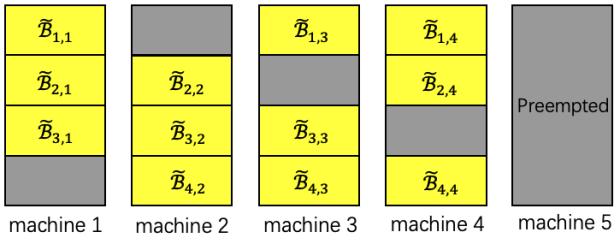

Fig. 1. Cyclic assignment added to the example in Section III-A.

Fig. 1, we can see that each available machine $j \in [4]$ receives a fraction $\frac{3}{4}$ of $\tilde{\mathcal{B}}_j$. This shows that both encoding complexity and communication cost for each machine are reduced by factor $\frac{3}{4}$.

In the computing phase, for group $g \in [4]$ the assigned machine $j \in \mathcal{P}_g$ computes the following 3 functions:

$$\begin{aligned} f_{1,g}(A, \tilde{\mathcal{B}}_{g,j}) &= A_{1,1}\tilde{B}_{g,j,1} + A_{1,2}\tilde{B}_{g,j,2} + A_{1,3}\tilde{B}_{g,j,3}, \\ f_{2,g}(A, \tilde{\mathcal{B}}_{g,j}) &= A_{2,1}\tilde{B}_{g,j,1} + A_{2,2}\tilde{B}_{g,j,2} + A_{2,3}\tilde{B}_{g,j,3}, \\ f_{3,g}(A, \tilde{\mathcal{B}}_{g,j}) &= A_{3,1}\tilde{B}_{g,j,1} + A_{3,2}\tilde{B}_{g,j,2} + A_{3,3}\tilde{B}_{g,j,3}, \end{aligned} \tag{11}$$

and sends the computation results to the master. Compared to the previous example, the computation complexity of each available machine j is also reduced by factor $\frac{3}{4}$ since it uses a fraction $\frac{3}{4}$ of $\tilde{\mathcal{B}}_j$ to multiply with its storage.

In the decoding phase, for group $g \in [4]$ we perform the following processes. Define 3 polynomials

$$\begin{aligned} h_{1,g}(z) &= f_{1,g}(A, \mathcal{V}_g(z)) = A_{1,1}V_{1,g}(z) + A_{1,2}V_{2,g}(z) + A_{1,3}V_{3,g}(z), \\ h_{2,g}(z) &= f_{2,g}(A, \mathcal{V}_g(z)) = A_{2,1}V_{1,g}(z) + A_{2,2}V_{2,g}(z) + A_{2,3}V_{3,g}(z), \\ h_{3,g}(z) &= f_{3,g}(A, \mathcal{V}_g(z)) = A_{3,1}V_{1,g}(z) + A_{3,2}V_{2,g}(z) + A_{3,3}V_{3,g}(z), \end{aligned}$$

such that for $i \in [3]$ and $j \in [3]$, we have $h_{i,g}(\beta_j) = f_{i,g}(A, \mathcal{V}_g(\beta_j)) = A_{i,1}V_{1,g}(\beta_j) + A_{i,2}V_{2,g}(\beta_j) + A_{i,3}V_{3,g}(\beta_j) = A_{i,1}(B_{1,j})_g + A_{i,2}(B_{2,j})_g + A_{i,3}(B_{3,j})_g = (A_{i,1}B_{1,j})_g + (A_{i,2}B_{2,j})_g + (A_{i,3}B_{3,j})_g = \left(A_{i,1}B_{1,j} + A_{i,2}B_{2,j} + A_{i,3}B_{3,j}\right)_g = (C_{i,j})_g$ due to (10), where $(\cdot)_g$ is the g-th group of columns in matrix $(\cdot)$. Also, we have $h_{i,g}(\alpha_j) = f_{i,g}(A, \mathcal{V}_g(\alpha_j)) = f_{i,g}(A, \tilde{\mathcal{B}}_{j,g})$ for machine $j \in \mathcal{P}_g$. Hence, decoding $(C_{i,j})_g$, $i \in [3]$, $j \in [3]$ means to interpolate the polynomial $h_{i,g}(z)$ by using computation results $\{f_{i,g}(A, \tilde{\mathcal{B}}_{g,k}) : k \in \mathcal{P}_g\}$, and to evaluate $h_{i,g}(\beta_j)$. Therefore, the master decodes the g-th groups of all blocks $C_{i,j}$. After repeating the above decoding

process for all groups $g \in [4]$, the master is able to decode $C = AB$.

B. General Homogeneous Straggler Tolerant CEC Scheme

Let us consider the general homogeneous CEC scheme with arbitrary partitioning parameters. The master has divided A into $m \times p$ sub-matrices and divided B into $p \times n$ sub-matrices as shown in (3). Each machine has stored the entire A.

In the communication phase, consider the cyclic assignment with straggler tolerance S, where $0 \leq S \leq N_t - n$, i.e., $(D = n + S,\ G = N_t,\ \boldsymbol{\mu} = (\frac{D}{N_t}, \cdots, \frac{D}{N_t}),\ \boldsymbol{\gamma} = (\frac{1}{N_t}, \cdots, \frac{1}{N_t}),\ \boldsymbol{\mathcal{P}} = \{\mathcal{P}_g : g \in [N_t]\})$, where $\mathcal{P}_g = \{(g-D+1)\%N_t, (g-D+2)\%N_t, \cdots, g\%N_t\}$ and we define $a\%N_t \triangleq a + \lfloor \frac{N_t - a + 1}{N_t} \rfloor N_t$. In this assignment, each machine is assigned to D groups. According to $\boldsymbol{\gamma}$, the master partitions each block $B_{i,j} \in \mathbb{F}^{\frac{v}{p} \times \frac{r}{n}}$ into N_t groups, i.e., $B_{i,j} = \{(B_{i,j})_g \in \mathbb{F}^{\frac{v}{p} \times \frac{r}{nN_t}} : g \in [N_t]\}$. From this notation, the computation output C is composed of mnN_t blocks $(C_{i,j})_g \in \mathbb{F}^{\frac{q}{m} \times \frac{r}{nN_t}}$, $i \in [m]$, $j \in [n]$ and $g \in [N_t]$. The master selects n distinct numbers $\{\beta_i : i \in [n]\}$ in $\mathbb{F}$ and for group $g \in [N_t]$ it generates a set of p polynomials $\mathcal{V}_g(z) = \{V_{i,g}(z) : i \in [p]\}$ each of degree $n-1$, where

$$V_{i,g}(z) = \sum_{j \in [n]} (B_{i,j})_g \cdot \prod_{k \in [n] \setminus \{j\}} \frac{z - \beta_k}{\beta_j - \beta_k}, \tag{12}$$

such that for $i \in [p]$ and $j \in [n]$, $V_{i,g}(\beta_j) = (B_{i,j})_g$ holds. Then we have

$$\mathcal{V}_g(\beta_j) = \{(B_{1,j})_g, (B_{2,j})_g, \cdots, (B_{p,j})_g\}, \tag{13}$$

which is exactly the g-th group of the j-th column block in B. Next, the master selects N_t distinct numbers $\{\alpha_j : j \in [N_t]\}$ in $\mathbb{F}$ such that $\{\alpha_j : j \in [N_t]\} \cap \{\beta_i : i \in [n]\} = \emptyset$. According to the computation assignment, for group $g \in [N_t]$ and assigned machine $j \in \mathcal{P}_g$, the master computes and sends $\tilde{\mathcal{B}}_{g,j} = \mathcal{V}_g(\alpha_j) = \{V_{1,g}(\alpha_j), V_{2,g}(\alpha_j), \cdots, V_{p,g}(\alpha_j)\} = \{\tilde{B}_{g,j,1}, \tilde{B}_{g,j,2}, \cdots, \tilde{B}_{g,j,p}\}$ to machine j.

In the computing phase, for $g \in [N_t]$ the assigned machine $j \in \mathcal{P}_g$ computes m functions $f_{i,g}(A, \tilde{\mathcal{B}}_{g,j}), i \in [m]$ and sends the computation results to the master, where

$$f_i(A, \tilde{\mathcal{B}}_{g,j}) = \sum_{k \in [p]} A_{i,k} \cdot \tilde{B}_{g,j,k}. \tag{14}$$

In the decoding phase, in order to decode mn blocks $\{(C_{i,j})_g : i \in [m], j \in [n]\}$ for group $g \in [N_t]$, we define m polynomials $h_{i,g}(z)$, $i \in [m]$, where

$$h_{i,g}(z) = f_{i,g}(A, \mathcal{V}_g(z)) = \sum_{k \in [p]} A_{i,k} \cdot V_{k,g}(z), \tag{15}$$

such that for $j \in [n]$ we have $h_{i,g}(\beta_j) = f_{i,g}(A, \mathcal{V}_g(\beta_j)) = \sum_{k \in [p]} A_{i,k} \cdot V_{k,g}(\beta_j) = \sum_{k \in [p]} A_{i,k} \cdot (B_{k,j})_g = \sum_{k \in [p]} (A_{i,k} \cdot B_{k,j})_g = \left(\sum_{k \in [p]} A_{i,k} \cdot B_{k,j}\right)_g = (C_{i,j})_g$ due to (13), where $(\cdot)_g$ is the g-th group of columns in matrix $(\cdot)$. Also, we have $h_{i,g}(\alpha_j) = f_{i,g}(A, \mathcal{V}_g(\alpha_j)) = f_i(A, \tilde{\mathcal{B}}_{g,j})$ for machine $j \in \mathcal{P}_g$. Hence, decoding $(C_{i,j})_g$, $i \in [m]$, $j \in [n]$ means to interpolate the polynomial $h_{i,g}(z)$ by using computation results $\{f_{i,g}(A, \tilde{\mathcal{B}}_{g,k}) : k \in \mathcal{L}_g\}$, and to evaluate $h_{i,g}(\beta_j)$, where $\mathcal{L}_g \subseteq \mathcal{P}_g$ is the set of n machines in $\mathcal{P}_g$. That is, the master computes

$$(C_{i,j})_g = h_{i,g}(\beta_j) = \sum_{k \in \mathcal{L}_g} f_i(A, \tilde{\mathcal{B}}_{g,k}) \cdot \prod_{k' \in \mathcal{L}_g \setminus \{k\}} \frac{\beta_j - \alpha_{k'}}{\alpha_k - \alpha_{k'}}. \tag{16}$$

For all groups $g \in [N_t]$ the master performs above decoding and obtains mnN_t blocks $(C_{i,j})_g$ for $i \in [m]$, $j \in [n]$, and $g \in [N_t]$. Finally, it recovers $C = AB$.

V. Discussion

A. Discussion of Homogeneous CEC Schemes

Compared to the existing homogeneous CEC scheme in [9] for matrix multiplication, our scheme presented in Section IV-B has the following advantages.

1) Straggler tolerance: As we already discussed in Section II, the CEC scheme in [9] is not robust for stragglers. In contrast, in our CEC scheme for each group we perform encoding using polynomials such that the master can assign $D \geq L$ machines by evaluating D arguments, where $L = n$. For decoding the master uses Lagrange polynomial interpolation that requires computation results from any L machines. Therefore, the scheme has straggler tolerance of $D - L$.

2) Maximum number of machines and recovery threshold: We prefer to larger maximum number of machines N_{max} and smaller recovery threshold L so that the distributed system can use more machines and require fewer successful machines. In the CEC scheme in [9], N_{max} and L are determined by the size of MDS generator matrix $X \in \mathbb{F}^{N_{max} \times L}$, which limits not only the size of N_{max} but also the flexibility of the parameters N_{max} and L. In contrast, in our CEC scheme N_{max} can be large and flexible, where $N_{max} = |\mathbb{F}| - L$, and $|\mathbb{F}|$ is the size of finite field $\mathbb{F}$. Note that N_{max} machines correspond to a set of N_{max} numbers $\{\alpha_j \in \mathbb{F} : j \in [N_{max}]\}$ that has no intersection with $\{\beta_i \in \mathbb{F} : i \in [L]\}$ in (12).

3) Partitioning and recovery threshold: Since the CEC scheme in [9] has the unique partitioning parameter L, it only can partition two matrices into column and row blocks respectively. In contrast, in our CEC scheme multiple partitioning parameters m, n, p divide matrices into smaller blocks flexibly, while keeping a lower recovery threshold n.

4) Applications to function computation: In some machine learning applications the objective is to compute a non-linear function $s(\cdot)$ based on a matrix multiplication, i.e., $s(AB)$. When $s(\cdot)$ is a polynomial function, our CEC scheme also can compute $s(AB)$. Specifically, in the computing phase the machines compute $s\left(f_i(A, \tilde{\mathcal{B}}_{g,j})\right)$ similar to (14). In the decoding phase, similar to (16) the master evaluates $s\left(h_{i,g}(z)\right)$ to obtain all blocks $s((C_{i,j})_g)$. Finally, the master is able to recover $s(C) = s(AB)$.

5) Computational complexity: Let our CEC scheme and the CEC scheme in [9] have the same matrix size q, v, r, number of available machines N_t and recovery threshold n. Besides, let the straggler tolerance $S = 0$ in our CEC scheme.

Denote the per-machine encoding complexity as the complexity of encoding the data computed by one machine, including the complexity of encoding both A and B. In our CEC scheme, for one machine in the g-th group the master obtains p matrices by (12) and each matrix requests $\mathcal{O}(\frac{vr}{pN_t})$ multiplications. Since each machine is assigned to n groups, after adding the complexity of all its allocated groups, the per-machine encoding complexity of B is $\mathcal{O}(\frac{vrn}{N_t})$, which is same as that of scheme in [9]. Note that [9] also has a per-machine encoding complexity of A being $\mathcal{O}(qv)$.

For per-machine computation complexity, from (14) one machine in the g-th group computes m functions requesting $\mathcal{O}(\frac{qvr}{nN_t})$ multiplications. By adding the complexity of all its allocated n groups, the per-machine computation complexity is $\mathcal{O}(\frac{qvr}{N_t})$, which is the same as that of scheme in [9].

Consider the communication cost of sending encoded B and receiving computation results. In our CEC scheme, the former communication cost is $\mathcal{O}(vr)$ and the latter communication cost is $\mathcal{O}(qr)$. In the CEC scheme in [9], the former communication cost is also $\mathcal{O}(vr)$ while the latter communication cost is $\mathcal{O}(qrnN_t)$.

For the decoding complexity, in the g-th group the master decodes mn blocks in (16), and each one requests $\mathcal{O}(\frac{qr}{mN_t})$ multiplications. By adding the complexity of all N_t groups, the resulting decoding complexity is $\mathcal{O}(qrn)$. The scheme in [9] has the advantage of a decoding complexity of $\mathcal{O}(1)$ since it only has additive operations.

B. Discussion of Coded Computing Schemes

The straggler tolerance of CEC schemes is determined by the employed coded computing schemes. Apart from MDS and Lagrange codes, there are many other existing coded techniques that can be applied to CEC and mitigate the impact of stragglers, such as entangled polynomial codes (EP codes) [8] and PolyDot codes [5]. Given the same computation assignment, comparing the complexities of obtained CEC schemes equals to comparing the corresponding coded computing schemes. Moreover, given the same parameters, the PolyDot codes have the same complexity and larger recovery threshold then the EP codes. Hence, we only compare our coded computing scheme with the EP codes in [8].

Let these two schemes have the same number of machines N, partitioning parameters m, n, p, and matrix size q, v, r. Since the storage ratio of our coded computing scheme is $R = 1$ while that of EP codes is $R = \frac{1}{pm}$, we let each machine use an EP codes with pm coded matrices to achieve the same storage ratio for a fair comparison. Then, the recovery threshold for the EP codes is reduced from $pmn + p - 1$ to $\lceil n + \frac{1}{m} - \frac{1}{pm} \rceil$. Hence, for the balanced EP codes each machine receives pm coded matrices in the communication phase, computes pm computation tasks in the computing phase, and attempts to send pm computation results to the master in the decoding phase.

The comparison is summarized in Table I. Define $\mathcal{C}_{eA}$ and $\mathcal{C}_{eB}$ as the per-machine encoding complexity of A and B, respectively, $\mathcal{C}_c$ as the per-machine computation complexity and $\mathcal{C}_d$ as the decoding complexity. Further, define $\mathcal{U}$ and $\mathcal{D}$ as the communication cost of sending an encoded version of B to all machines and receiving computation results, respectively.

TABLE I
COMPLEXITY COMPARISON BETWEEN PROPOSED SCHEME AND EP CODES [8]

	Our Scheme	EP codes [8]
L	n	$\begin{cases} n & \text{if } p = 1 \\ n+1 & \text{if } p > 1 \end{cases}$
R	1	1
$(\mathcal{C}_{eA}, \mathcal{C}_{eB})$	$(\mathcal{O}(1), \mathcal{O}(vr))$	$(\mathcal{O}(qvmp), \mathcal{O}(vrmp))$
$\mathcal{C}_c$	$\mathcal{O}(\frac{qvr}{n})$	$\mathcal{O}(\frac{qvr}{n})$
$\mathcal{C}_d$	$\mathcal{O}(qrn)$	$\mathcal{O}(qrp\log^2(pmn)\log\log(pmn))$
$(\mathcal{U}, \mathcal{D})$	$\left(\mathcal{O}(\frac{vrN}{n}), \mathcal{O}(qr)\right)$	$\left(\mathcal{O}(\frac{vrmN}{n}), \mathcal{O}(qrp)\right)$

From this table, our coded computing scheme has the lower per-machine encoding complexity, the communication cost, the decoding complexity and a smaller recovery threshold when $p > 1$, under the same storage ratio and per-machine computation complexities.

ACKNOWLEDGEMENT

This research was sponsored by the National Science Foundation (NSF) CAREER Award 2145835.

REFERENCES

[1] K. Lee, M. Lam, R. Pedarsani, D. Papailiopoulos, and K. Ramchandran, "Speeding up distributed machine learning using codes," *IEEE Trans. Inf. Theory*, vol. 64, no. 3, pp. 1514–1529, 2018.

[2] Q. Yu, M. A. Maddah-Ali, and A. S. Avestimehr, "Straggler mitigation in distributed matrix multiplication: Fundamental limits and optimal coding," *IEEE Trans. Inf. Theory*, vol. 66, no. 3, pp. 1920–1933, 2020.

[3] J. So, B. Güler, and A. S. Avestimehr, "Codedprivateml: A fast and privacy-preserving framework for distributed machine learning," *IEEE J. Select. Areas Inf. Theory*, vol. 2, no. 1, pp. 441–451, 2021.

[4] S. Li, S. M. M. Kalan, Q. Yu, M. Soltanolkotabi, and A. S. Avestimehr, "Polynomially coded regression: Optimal straggler mitigation via data encoding," *arXiv:1805.09934*, 2020.

[5] S. Dutta, M. Fahim, F. Haddadpour, H. Jeong, V. Cadambe, and P. Grover, "On the optimal recovery threshold of coded matrix multiplication," *IEEE Trans. Inf. Theory*, vol. 66, no. 1, pp. 278–301, 2020.

[6] Q. Yu, S. Li, N. Raviv, S. M. M. Kalan, M. Soltanolkotabi, and S. A. Avestimehr, "Lagrange coded computing: Optimal design for resiliency, security, and privacy," in *Proc. IEEE Int. Conf. on Artificial Intelligence and Statistics (AISTATS)*, 2019, pp. 1215–1225.

[7] M. Aliasgari, O. Simeone, and J. Kliewer, "Private and secure distributed matrix multiplication with flexible communication load," *IEEE Trans. Inf. Forensic Secur.*, vol. 15, pp. 2722–2734, 2020.

[8] Q. Yu and A. S. Avestimehr, "Entangled polynomial codes for secure, private, and batch distributed matrix multiplication: Breaking the "cubic" barrier," in *Proc. IEEE Int. Symp. on Inform. Theory (ISIT)*, 2020, pp. 245–250.

[9] Y. Yang, M. Interlandi, P. Grover, S. Kar, S. Amizadeh, and M. Weimer, "Coded elastic computing," *arXiv:1812.06411v3*, 2018.

[10] H. Dau, R. Gabrys, Y.-C. Huang, C. Feng, Q.-H. Luu, E. Alzahrani, and Z. Tari, "Optimizing the transition waste in coded elastic computing," in *Proc. IEEE Int. Symp. on Inform. Theory (ISIT)*, 2020, pp. 174–178.

[11] N. Woolsey, R.-R. Chen, and M. Ji, "Heterogeneous computation assignments in coded elastic computing," in *Proc. IEEE Int. Symp. on Inform. Theory (ISIT)*, 2020, pp. 168–173.