Flexible Distributed Matrix Multiplication

Weiqi Li, Zhen Chen, Zhiying Wang, Syed A. Jafar, and Hamid Jafarkhani

Abstract

The distributed matrix multiplication problem with an unknown number of stragglers is considered, where the goal is to efficiently and flexibly obtain the product of two massive matrices by distributing the computation across N servers. There are up to $N-R$ stragglers but the exact number is not known a priori. Motivated by reducing the computation load of each server, a flexible solution is proposed to fully utilize the computation capability of available servers. The computing task for each server is separated into several subtasks, constructed based on Entangled Polynomial codes by Yu et al. The final results can be obtained from either a larger number of servers with a smaller amount of computation completed per server or a smaller number of servers with a larger amount of computation completed per server. The required finite field size of the proposed solution is less than $2N$. Moreover, the optimal design parameters such as the partitioning of the input matrices are discussed. Our constructions can also be generalized to other settings such as batch distributed matrix multiplication and secure distributed matrix multiplication.

I. Introduction

Distributed matrix multiplication has received wide interest because of the huge amount of data computation required by many popular applications like federated learning, cloud computing, and the Internet of things. In particular, the multiplication of two massive input matrices $A \in \mathbb{F}^{\lambda\times\kappa}$ and $B \in \mathbb{F}^{\kappa\times\mu}$, where $\mathbb{F}$ is some finite field is considered. Each matrix is encoded into N *shares* and distributed to N servers. Each server performs computation on its own shares and sends the *results* to the central computational node, e.g., the cloud. After collecting enough results, the desired product AB can be calculated. However, stragglers (servers that fail to respond or respond after the the reconstruction is executed) are inevitable in distributed systems, due to various reasons [2], [3] including network latency, resource contention, workload imbalance, failures of hardware or software, etc. To reduce the overall system latency caused by stragglers, distributed matrix computing schemes

Part of this paper was presented in ISIT 2021 [1].

The authors are with Center for Pervasive Communications and Computing (CPCC), University of California, Irvine, USA. emails: {weiqil4, zhenc4, zhiying, syed, hamidj}@uci.edu.

with straggler tolerance are provided in [4]–[35] with a predetermined *recovery threshold* R such that the final product can be obtained using computation results from any R out of N servers. Among the state-of-the-art schemes, some are based on matrix partitioning such as Polynomial codes [5], MatDot codes and PolyDot codes [6], Generalized PolyDot codes [7] and Entangled Polynomial (EP) codes [8], and others are based on batch processing such as Lagrange Coded Computing [9], Cross Subspace Alignment (CSA) codes and Generalized Cross Subspace Alignment (GCSA) codes [33].

The above literature assumes there are a fixed number $N-R$ of stragglers. However, the number of stragglers is unpredictable in practical systems. When the number of stragglers is smaller than $N-R$, each non-straggler server still needs to do the same amount of computation as if there are $N-R$ stragglers and the central node still only uses the results from R servers. A significant amount of computation power is wasted. To handle this situation, a setting in which the number of stragglers is not known a priori has been considered in [36]–[48] and schemes that can cope with such a setting have been designed. The underlying idea is to assign a sequence of small tasks to each server instead of assigning a single large task. Therefore, besides the scenario that the fastest R servers finish all their tasks, there are other scenarios that make the computation complete. References [36], [37] focus on the task scheduling for general distributed computing. The matrix-vector multiplication setting is considered in [38]–[41]. In these works, only the input matrix is partitioned. References [42]–[47] consider matrix-matrix multiplication, but they can only handle a special partitioning, i.e., A and B are row-wisely and column-wisely split, respectively, or only A is row-wisely split. In [48], the authors propose 3 hierarchical schemes for matrix multiplication to leverage partial stragglers. The main idea is that the task is first divided into several small subtasks, i.e., the multiplication of several pairs of small matrices, and each subtask is coded separately with existing schemes.

Arbitrary partitioning of input matrices is important in massive matrix multiplication since it enables different utilization of system resources, e.g., the required amount of storage and computation at each server. When the number of stragglers is fixed, many codes such as PolyDot codes [6], EP codes [8] and GCSA codes [33] provide elegant solutions for arbitrary partitioning by encoding the input matrix blocks into a carefully designed polynomial. In particular, EP codes effectively align the servers' computation with the terms that the central node needs and achieve the optimal recovery threshold among all linear coding strategies in some cases.

A naive solution to achieve flexibility for distributed matrix multiplication with arbitrary partitioning is simply applying a fixed EP code with a recovery threshold of PR, where each server gets P pairs

of shares instead of one pair of shares. The central node can calculate the final results with any PR out of the PN computing results. Thus, each server only needs to compute RP/N results when there is no straggler, and in general, the number of results computed in each server can be adjusted based on the number of stragglers. However, by doing so, the code requires both the code length and the field size to be PN[1].

As analyzed in [50], multiplications in Galois fields are usually done by pre-computed look-up tables, and thus the memory cost and the time complexity increase significantly with the field size. See [51]–[54] for efforts to alleviate the memory and the time complexity problems of large fields.

In this paper, we present a flexible coding scheme for distributed matrix multiplication that allows a flexible number of stragglers and arbitrary matrix partitioning while only requiring a much smaller field size. The main idea is that non-straggler servers can finish more tasks to compensate for the effect of the stragglers without knowing the stragglers a priori. Specifically, the computation is encoded into several tasks for each server, and each server keeps calculating and sending results to the central node until enough results are obtained. Enough results can be either a larger number of servers with a smaller amount of completed computation by each server or a smaller number of servers with a larger amount of completed computation by each server. Therefore, the number of available servers is flexible and the number of required tasks is adjusted to the number of available servers. In our model, we treat stragglers as unavailable servers, which may occur in the cases of bad disks, low priority of the tasks, or broken processes [55]. Our scheme is different from those that leverage partial stragglers [40], [41], [43], [46]–[48]. In our construction, the computation load (the number of multiplication operations) of each non-straggler server is the same and the computation by stragglers is neglected, while in schemes with partial stragglers, the computation load varies in different servers including stragglers.

The main contributions of the paper are as follows. We present a coding framework of flexible distributed matrix multiplication schemes, and one-round and multi-round communication models. A construction with multiple layers of computation tasks is proposed, which only requires a field size of less than $2N$ and the computation load of each server is reduced significantly when there are fewer

[1]Our method can also be applied to matrices over real numbers. As discussed in [49], real number field suffers from stability issues owing to the condition number of the corresponding real Vandermonde-structured recovery matrices. For a Reed-Solomon-type code with length n, the condition number grows exponentially or polynomially in n, depending on the implementation. For a code over a finite field, n corresponds to the finite field size. Due to the small field size of our proposed approach, it provides a solution for the real matrices with a small condition number.

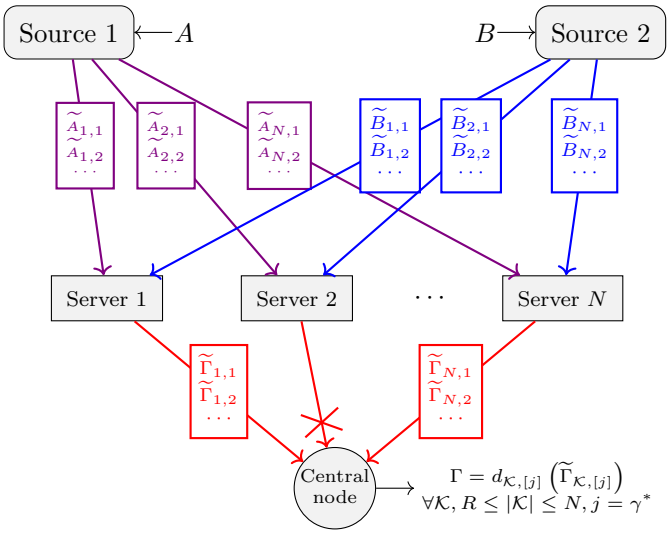

Fig. 1. The flexible distributed matrix multiplication problem.

stragglers than $N - R$. We also demonstrate the optimization of the parameters to obtain the lowest computation load. We show that the two-layer construction outperforms the fixed scheme under the one-round model as long as the server storage is above a threshold, and the maximum number of layers is preferred under the multi-round model.

The rest of the paper is organized as follows. Section II presents the problem statement. In Section III, we present our construction and its performance. The choice of parameters to optimize the computation load given the storage capacity is discussed in Section IV. Section V concludes the paper.

Notation: We use calligraphic characters to denote sets. For positive integer N, $[N]$ stands for the set $\{1, 2, \ldots, N\}$. For a matrix M, $|M|$ denotes its number of entries. For a set of matrices $\mathcal{M}$, $|\mathcal{M}|$ represents the sum of the number of entries in all its matrices. When M is partitioned into sub-block matrices, $M_{(u,v)}$ denotes the block in the u-th row and the v-th column.

II. Problem Statement

We consider a problem of matrix multiplication (see Fig. 1) with two input matrices $A \in \mathbb{F}^{\lambda \times \kappa}$ and $B \in \mathbb{F}^{\kappa \times \mu}$, for some integers λ, κ, μ and a field $\mathbb{F}$. We are interested in computing the product $\Gamma = AB$ in a distributed computing environment with 2 sources, a central node, and N servers. Sources 1 and 2 hold matrices A and B, respectively. It is assumed that there are up to $N - R$ stragglers among the servers. In non-flexible distributed matrix multiplication, R is called the *recovery threshold*. The shares (coded matrix sets) $\widetilde{\mathcal{A}}_i$ and $\widetilde{\mathcal{B}}_i$ are generated by sources for Server $i, i \in [N]$. Each share consists of

some coded matrices, denoted by $\left\{\widetilde{A}_{i,1},\cdots,\widetilde{A}_{i,\widetilde{\gamma}}\right\}$, or $\left\{\widetilde{B}_{i,1},\cdots,\widetilde{B}_{i,\widetilde{\gamma}}\right\}$, where $\widetilde{\gamma}$ is a function of N and R. For $i\in[N]$, the shares and the encoding functions are

$$\widetilde{\mathcal{A}}_i=\left\{\widetilde{A}_{i,j}\mid j\in[\widetilde{\gamma}]\right\}=u_i(A), \tag{1}$$

$$\widetilde{\mathcal{B}}_i=\left\{\widetilde{B}_{i,j}\mid j\in[\widetilde{\gamma}]\right\}=v_i(B). \tag{2}$$

Then, $\widetilde{\mathcal{A}}_i$ and $\widetilde{\mathcal{B}}_i$ are sent to Server i from the sources. Server i sequentially computes $\widetilde{\gamma}$ *tasks* in order:

$$\widetilde{\Gamma}_{i,j}=\widetilde{A}_{i,j}\cdot\widetilde{B}_{i,j},\ j\in[\widetilde{\gamma}], \tag{3}$$

and sends $\widetilde{\Gamma}_{i,j}$ to the central node once its computation is finished. Due to the sequential processing nature of the servers, the central node receives $\widetilde{\Gamma}_{i,j_1}$ before $\widetilde{\Gamma}_{i,j_2}$ for $\forall i\in[N],j_1<j_2$. Denote $\widetilde{\Gamma}_{i,[j]}=\left\{\widetilde{\Gamma}_{i,t}\mid t\in[j]\right\}$ and $\widetilde{\Gamma}_{\mathcal{K},[j]}=\left\{\widetilde{\Gamma}_{i,[j]}\mid i\in\mathcal{K}\right\},\forall\mathcal{K}\subseteq[N]$.

We require that the central node be able to decode the desired product Γ from arbitrary $\hat{R}\geq R$ servers, where each server calculates γ^* (a function of $\hat{R}$) tasks. Equivalently, the decoding function $d_{\mathcal{K},[j]}$ of the central node for recovering Γ satisfies

$$\Gamma=d_{\mathcal{K},[j]}\left(\widetilde{\Gamma}_{\mathcal{K},[j]}\right),\forall\mathcal{K},R\leq|\mathcal{K}|=\hat{R}\leq N,j=\gamma^*. \tag{4}$$

The function set $\{u_i,v_i,d_{\mathcal{K},[j]}\mid 1\leq i\leq N,R\leq|\mathcal{K}|=\hat{R}\leq N,j=\gamma^*\}$ is called the *flexible constructions for distributed matrix multiplication*.

In other words, the sources send coded matrices to each server. Each server keeps calculating and sending results to the central node until it obtains enough results – when the quickest $\hat{R}$ servers complete the first γ^* tasks. The remaining servers are viewed as stragglers and the computation results from stragglers are ignored.

In this work, we consider two communication models: the one-round communication model and the multi-round communication model. For the *one-round communication model*, the sources send *all* $\widetilde{\gamma}$ pairs of coded matrices to the server at one time. After that there are no communications between sources and servers. For the *multi-round communication model*, first, the sources send one pair of coded matrices to the servers. Once a server finishes its tasks, it will ask the sources to send another pair of coded matrices. It is not necessary for the sources to know which servers are the stragglers. This procedure lasts until the central node obtains enough results. Note that there are no communications among servers in either model.

TABLE I

CALCULATION TASKS IN EACH SERVER FOR EXAMPLE 1.

	Server 1	Server 2	Server 3	Server 4	Server 5
Layer 1	$A_{\alpha_1} \cdot B_{\alpha_1}$	$A_{\alpha_2} \cdot B_{\alpha_2}$	$A_{\alpha_3} \cdot B_{\alpha_3}$	$A_{\alpha_4} \cdot B_{\alpha_4}$	$A_{\alpha_5} \cdot B_{\alpha_5}$
Layer 2	$(A_{\alpha_6,1} + \alpha_1 A_{\alpha_6,2})$ $\cdot(\alpha_1 B_{\alpha_6,1} + B_{\alpha_6,2})$, $(A_{\alpha_7,1} + \alpha_1 A_{\alpha_7,2})$ $\cdot(\alpha_1 B_{\alpha_7,1} + B_{\alpha_7,2})$	$(A_{\alpha_6,1} + \alpha_2 A_{\alpha_6,2})$ $\cdot(\alpha_2 B_{\alpha_6,1} + B_{\alpha_6,2})$, $(A_{\alpha_7,1} + \alpha_2 A_{\alpha_7,2})$ $\cdot(\alpha_2 B_{\alpha_7,1} + B_{\alpha_7,2})$	$(A_{\alpha_6,1} + \alpha_3 A_{\alpha_6,2})$ $\cdot(\alpha_3 B_{\alpha_6,1} + B_{\alpha_6,2})$, $(A_{\alpha_7,1} + \alpha_3 A_{\alpha_7,2})$ $\cdot(\alpha_3 B_{\alpha_7,1} + B_{\alpha_7,2})$	$(A_{\alpha_6,1} + \alpha_4 A_{\alpha_6,2})$ $\cdot(\alpha_4 B_{\alpha_6,1} + B_{\alpha_6,2})$, $(A_{\alpha_7,1} + \alpha_4 A_{\alpha_7,2})$ $\cdot(\alpha_4 B_{\alpha_7,1} + B_{\alpha_7,2})$	$(A_{\alpha_6,1} + \alpha_5 A_{\alpha_6,2})$ $\cdot(\alpha_5 B_{\alpha_6,1} + B_{\alpha_6,2})$, $(A_{\alpha_7,1} + \alpha_5 A_{\alpha_7,2})$ $\cdot(\alpha_5 B_{\alpha_7,1} + B_{\alpha_7,2})$

The *computation load* L is defined as the number of multiplication operations per server. Moreover, each server has a *storage capacity* C [2]. At any time, any server cannot store more than C. Specifically, for the one-round communication model, $\max_{i\in[N]} \left(\left|\widetilde{\mathcal{A}}_i\right| + \left|\widetilde{\mathcal{B}}_i\right|\right) \leq C$. For the multi-round communication model, $\max_{i\in[N],j\in[\widetilde{\gamma}]} \left(\left|\widetilde{A}_{i,j}\right| + \left|\widetilde{B}_{i,j}\right|\right) \leq C$. This is because once a server finishes a task and sends the result to the central node, it can refresh the storage and delete the coded matrices related to this task. In general, the storage constraint is stricter in the one-round communication model. We want to find flexible constructions with the *storage capacity* C and the *computation load* L at each server as small as possible.

III. CONSTRUCTION

In this section, we present our flexible constructions. The scheme is based on EP code [8] and the computation tasks are divided into several layers to provide flexibility. The main idea of EP code is polynomial interpolation where the desired matrix product can be retrieved as polynomial coefficients and the servers compute evaluations of the polynomial. In our construction, we further view the evaluations of the original polynomial as the coefficients of another polynomial, thus enabling flexibility. We start with a motivating example. The general construction and its storage and computation load are presented afterwards.

Example 1. The construction is shown in Table I. Consider the matrix multiplication of A and B, for $A \in \mathbb{F}^{\lambda\times\kappa}, B \in \mathbb{F}^{\kappa\times\mu}$, with $N = 5$ servers and at most $N - R = 2$ stragglers. Suppose A is column-wisely partitioned as $A = [A_1, A_2]$, $A_1, A_2 \in \mathbb{F}^{\lambda\times\frac{\kappa}{2}}$, and B is row-wisely partitioned as $B = \begin{bmatrix} B_1 \\ B_2 \end{bmatrix}$,

[2]The maximum storage size C is usually smaller than $|A| + |B|$, otherwise the sources can send A and B to the servers.

$B_1, B_2 \in \mathbb{F}^{\frac{\kappa}{2} \times \mu}$. The central node requires $AB = A_1B_1 + A_2B_2$. Applying the EP code, Server $i, i \in [5]$ receives coded matrices $A_1 + \alpha_i A_2$ and $\alpha_i B_1 + B_2$, for $\alpha_i \in \mathbb{F}$, and calculates

$$\begin{aligned}&(A_1 + \alpha_i A_2) \cdot (\alpha_i B_1 + B_2) \\ =&A_1B_2 + \alpha_i(A_1B_1 + A_2B_2) + \alpha_i^2 A_2B_1,\end{aligned} \tag{5}$$

which is a degree 2 polynomial with respect to α_i. Thus, $A_1B_1 + A_2B_2$ can be calculated by 3 distinct evaluations from $\{\alpha_i \mid i \in [5]\}$ using polynomial interpolation. The total computation load of directly multiplying A and B is $L = \lambda\kappa\mu$, while using the EP code the computation load of each server is $L/2$. However, when there is no straggler, the computation of 2 servers are wasted.

Alternatively, we can use a flexible scheme to calculate AB, such that any $\hat{R}$ available servers can complete the computation, $3 = R \leq \hat{R} \leq N = 5$. First, we partition the matrices and get $A = [A_1, A_2, A_3]$, $A_1, A_2, A_3 \in \mathbb{F}^{\lambda \times \frac{\kappa}{3}}$, and $B = [B_1^T, B_2^T, B_3^T]^T$, $B_1, B_2, B_3 \in \mathbb{F}^{\frac{\kappa}{3} \times \mu}$. The central node requires $AB = A_1B_1 + A_2B_2 + A_3B_3$. Let $\{\alpha_i | i \in [7]\}$ be distinct elements in $\mathbb{F}$. The calculation will be divided into 2 layers.

Layer 1: Server $i, i \in [5]$, calculates $\gamma_1 = 1$ task

$$\begin{aligned}\Gamma_{i,1} =&(A_1 + \alpha_i A_2 + \alpha_i^2 A_3) \cdot (\alpha_i^2 B_1 + \alpha_i B_2 + B_3) \\ =&A_1B_3 + \alpha_i(A_2B_3 + A_1B_2) + \alpha_i^2(A_1B_1 + A_2B_2 + A_3B_3) \\ &+ \alpha_i^3(A_2B_1 + A_3B_2) + \alpha_i^4 A_3B_1.\end{aligned} \tag{6}$$

It is a degree 4 polynomial with respect to α_i and the final product can be obtained from all 5 servers. If there is no straggler, we stop here. In this layer, matrices A and B are divided into smaller submatrices compared to the fixed EP code and the computation load of each server is $L/3$. If there are stragglers, the servers continue the calculation in Layer 2.

Layer 2: We set $A_{\alpha_i} = A_1 + \alpha_i A_2 + \alpha_i^2 A_3, B_{\alpha_i} = \alpha_i^2 B_1 + \alpha_i B_2 + B_3, i \in \{6, 7\}$ and partition them into 2 parts,

$$A_{\alpha_i} = [A_{\alpha_i,1}, A_{\alpha_i,2}], B_{\alpha_i} = \begin{bmatrix} B_{\alpha_i,1} \\ B_{\alpha_i,2} \end{bmatrix}, \tag{7}$$

where $A_{\alpha_i,1}, A_{\alpha_i,2} \in \mathbb{F}^{\lambda \times \frac{\kappa}{6}}, B_{\alpha_i,1}, B_{\alpha_i,2} \in \mathbb{F}^{\frac{\kappa}{6} \times \mu}$. Server i has $\gamma_2 = 2$ computation tasks:

$$\Gamma_{i,2} = (A_{\alpha_6,1} + \alpha_i A_{\alpha_6,2}) \cdot (\alpha_i B_{\alpha_6,1} + B_{\alpha_6,2}), \tag{8}$$

$$\Gamma_{i,3} = (A_{\alpha_7,1} + \alpha_i A_{\alpha_7,2}) \cdot (\alpha_i B_{\alpha_7,1} + B_{\alpha_7,2}). \tag{9}$$

The detailed calculation of each server is shown in Table I.

Since Layer 2 has a similar structure as (5), from any 3 of the servers, we can get $A_{\alpha_6} \cdot B_{\alpha_6}$ and/or $A_{\alpha_7} \cdot B_{\alpha_7}$. If there is one straggler, the central node obtains $A_{\alpha_6} \cdot B_{\alpha_6}$ from Layer 2, which causes the additional computation load of $L/6$ in a server. If there are 2 stragglers, the central node obtains both $A_{\alpha_6} \cdot B_{\alpha_6}$ and $A_{\alpha_7} \cdot B_{\alpha_7}$, which causes the computation load of $L/3$ in Layer 2 for each server.

In this example, there are two recovery thresholds $R_1 = 5$ and $R_2 = 3$, corresponding to two layers, respectively. We term the choice of per-layer recovery thresholds as *recovery profile*. There are totally $\widetilde{\gamma} = \gamma_1 + \gamma_2 = 3$ coded matrices in a share where γ_1 coded matrices correspond to Layer 1 and γ_2 coded matrices correspond to Layer 2. Specifically, $\forall i \in [N]$, the shares $\widetilde{\mathcal{A}}_i$ and $\widetilde{\mathcal{B}}_i$ contain

$$\widetilde{A}_{i,1} = A_{\alpha_i}, \quad \widetilde{A}_{i,2} = A_{\alpha_6,1} + \alpha_i A_{\alpha_6,2}, \quad \widetilde{A}_{i,3} = A_{\alpha_7,1} + \alpha_i A_{\alpha_7,2}, \tag{10}$$

$$\widetilde{B}_{i,1} = B_{\alpha_i}, \quad \widetilde{B}_{i,2} = B_{\alpha_6,1} + \alpha_i B_{\alpha_6,2}, \quad \widetilde{B}_{i,3} = B_{\alpha_7,1} + \alpha_i B_{\alpha_7,2}, \tag{11}$$

respectively. Each server needs to store all the above 6 coded matrices under the one-round communication model, but only 2 coded matrices at a time under the multi-round communication. Each server computes up to $\widetilde{\gamma} = 3$ tasks in order, independent of the progress of the other servers.

For Example 1, the computation load of each server is $L/3, L/2, 2L/3$ for the cases of no stragglers, 1 straggler and 2 stragglers, respectively. When there is no straggler (which is more likely in most practical systems), the computation load of each server is reduced by 33%, from $L/2$ to $L/3$. The resulting computation latency under an exponential model is plotted in Fig. 2.

In this example, if there is only one communication round from the sources to the servers, the storage size required for each server is $\frac{2\lambda\kappa}{3} + \frac{2\kappa\mu}{3}$ for our flexible construction and $\frac{\lambda\kappa}{2} + \frac{\kappa\mu}{2}$ for the EP code. We will discuss how to partition the matrices to obtain an advantageous computation load while maintaining the same storage size in Section IV.

Next, we present the general definitions and constructions of our flexible schemes. The key component is to generate extra parities during the encoding in each layer that will correspond to extra tasks to be completed by higher layers to compensate for more stragglers.

Define the *recovery profile* as a tuple of integers $(R_1, R_2, \cdots, R_a)$, where $N \geq R_1 > R_2 > ... > R_a = R$ and a is some integer termed the *number of layers*. Denote

$$\gamma_j = \begin{cases} 1, & j = 1, \\ (R_{j-1} - R_j) \sum_{J=1}^{j-1} \gamma_J, & 2 \leq j \leq a, \end{cases} \tag{12}$$

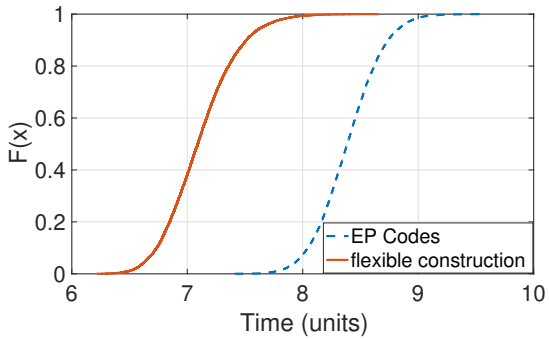

Fig. 2. CDF of computation latency for flexible construction and EP code in Example 1. $N = R_1 = 5, R_2 = R = 3$. We assume $\lambda = \kappa = \mu = 6U$, for some integer U, and the computation delay for multiplication of two $U \times U$ matrices in each server satisfy the exponential distribution with parameter 0.1, i.e., the expected computation delay is 10 time units. We use the time units here because the delay varies for different types of servers, and it should be noted that the delay is distributed in continuous time. The latency of the EP code is the delay of the 3rd quickest server, and the slowest 2 servers are viewed as stragglers. For the flexible construction, the computation is completed in the cases of 5 servers complete 1 task (no straggler), or 4 servers complete 2 tasks (1 straggler), or 3 servers complete 3 tasks (2 stragglers). The overall latency is the smallest latency of these 3 cases. The expected latency is 10.79 time units for EP code, and 8.20 time units for the flexible construction. Hence we save 24%.

which will be shown to be the number of tasks in each layer. For two matrices Φ, Ψ and partition parameters p_j, m_j, n_j, define functions $f_j, g_j, j \in [a]$, as

$$f_j(\alpha_i; \Phi) = \sum_{u=1}^{m_j} \sum_{v=1}^{p_j} \Phi_{(u,v)} \alpha_i^{v-1+p_j(u-1)}, \tag{13}$$

$$g_j(\alpha_i; \Psi) = \sum_{u=1}^{p_j} \sum_{v=1}^{n_j} \Psi_{(u,v)} \alpha_i^{p_j-u+p_j m_j(v-1)}, \tag{14}$$

where

$$\Phi = \begin{bmatrix} \Phi_{(1,1)} & \cdots & \Phi_{(1,p_j)} \\ \Phi_{(2,1)} & \cdots & \Phi_{(2,p_j)} \\ \vdots & \vdots & \vdots \\ \Phi_{(m_j,1)} & \cdots & \Phi_{(m_j,p_j)} \end{bmatrix}, \Psi = \begin{bmatrix} \Psi_{(1,1)} & \cdots & \Psi_{(1,n_j)} \\ \Psi_{(2,1)} & \cdots & \Psi_{(2,n_j)} \\ \vdots & \vdots & \vdots \\ \Psi_{(p_j,1)} & \cdots & \Psi_{(p_j,n_j)} \end{bmatrix}. \tag{15}$$

Note that (13) and (14) are the encoding functions of the EP codes [8] used in Layer j.

Construction 1. Given recovery profile $(R_1, R_2, \cdots, R_a)$ and partitioning parameters p_j, m_j, n_j such that $R_j = p_j m_j n_j + p_j - 1, j \in [a]$, the construction consists of a layers. Fix $N + R_1 - R_a$ distinct elements $\alpha_i, i \in [N + R_1 - R_a]$, in a finite field $\mathbb{F}$, $|\mathbb{F}| \geq N + R_1 - R_a$.

In Layer 1, set $A^{(1,1)} = A$ and $B^{(1,1)} = B$. A pair of coded matrices $f_1\left(\alpha_t; A^{(1,1)}\right)$ and $g_1\left(\alpha_t; B^{(1,1)}\right)$ are generated for Server t, $t \in [N]$. Moreover, extra $R_1 - R_a$ pairs of parities will be generated, i.e., $f_1\left(\alpha_{N+t}; A^{(1,1)}\right)$ and $g_1\left(\alpha_{N+t}; B^{(1,1)}\right)$, $t \in [R_1 - R_a]$. They will be used in higher layers.

In Layer $j, 2 \leq j \leq a$, the number of pairs of coded matrices is γ_j given by (12). For each $\delta_j \in [\gamma_j]$, a pair of coded matrices $f_j\left(\alpha_t; A^{(j,\delta_j)}\right)$ and $g_j\left(\alpha_t; B^{(j,\delta_j)}\right)$ are generated for Server t, $t \in [N]$. Besides, extra parities $f_j(\alpha_{N+t}; A^{(j,\delta_j)})$ and $f_j(\alpha_{N+t}; B^{(j,\delta_j)})$, $t \in [R_j - R_a]$, are produced for higher layers. Here, $A^{(j,\delta_j)}$ and $B^{(j,\delta_j)}$, $\delta_j \in [\gamma_j]$, are from the extra parities $f_J(\alpha_{N+t}; A^{(J,\delta_J)}), g_J(\alpha_{N+t}; B^{(J,\delta_J)})$ in Layer J for all $J \in [j-1]$ and

$$R_j - R_a + 1 \leq t \leq R_{j-1} - R_a, \delta_J \in [\gamma_J]. \tag{16}$$

Specifically, given j and δ_j, $A^{(j,\delta_j)}$ and $B^{(j,\delta_j)}$ are set as

$$A^{(j,\delta_j)} = f_J\left(\alpha_{N+t} : A^{(J,\delta_J)}\right), \tag{17}$$

$$B^{(j,\delta_j)} = g_J\left(\alpha_{N+t} : B^{(J,\delta_J)}\right), \tag{18}$$

where

$$t = \left\lfloor \delta_j \frac{R_{j-1} - R_j}{\gamma_j} \right\rfloor + R_j - R_a + 1, \tag{19}$$

and J is the integer satisfying

$$\sum_{x=1}^{J-1} \gamma_x < \delta_j \bmod \frac{\gamma_j}{(R_{j-1} - R_j)} \leq \sum_{x=1}^{J} \gamma_x \tag{20}$$

and

$$\delta_J = \delta_j \bmod \frac{\gamma_j}{(R_{j-1} - R_j)} - \sum_{x=1}^{J-1} \gamma_x. \tag{21}$$

Intuitively, the t-th extra parities in all previous layers are encoded in Layer j, for all t satisfying (16). Equations (19), (20), and (21) simply mean that these extra parities are ordered from left to right and from top to bottom (see Fig. 3 for an example).

Denote $\Gamma_{j,\delta_j}(\alpha_i)$ as the δ_j-th task in Layer j calculated in Server i, for $i \in [N], j \in [a], \delta_j \in [\gamma_j]$, where

$$\Gamma_{j,\delta_j}(\alpha_i) = f_j\left(\alpha_i; A^{(j,\delta_j)}\right) \cdot g_j\left(\alpha_i; B^{(j,\delta_j)}\right), \tag{22}$$

The calculation tasks of the construction are shown in Table II.

To summarize the construction, there are in total $\widetilde{\gamma} = \sum_{j=1}^{a} \gamma_j$ tasks. For Server i, the shares are constructed as

$$\widetilde{\mathcal{A}}_i = \{A^{(j,\delta_j)} \mid j \in [a], \delta_j \in [\gamma_j]\}, \tag{23}$$

$$\widetilde{\mathcal{B}}_i = \{B^{(j,\delta_j)} \mid j \in [a], \delta_j \in [\gamma_j]\}. \tag{24}$$

The k-th task of Server i is $\widetilde{\Gamma}_{i,k} = \Gamma_{j,\delta_j}(\alpha_i)$ where $k = \sum_{x=1}^{j-1} \gamma_x + \delta_j$.

TABLE II

CALCULATION TASKS IN EACH SERVER FOR THE MULTIPLE-LAYER CONSTRUCTION, WHERE δ_j RANGES BETWEEN 1 AND γ_j AS DEFINED IN (12), $j \in [a]$.

	Server 1	...	Server N	Extra parity 1	...	...	Extra parity $R_1 - R_a$
Layer 1	$\Gamma_{1,1}(\alpha_1)$	...	$\Gamma_{1,1}(\alpha_N)$	$\Gamma_{1,1}(\alpha_{N+1})$	...	...	$\Gamma_{1,1}(\alpha_{N+R_1-R_a})$
Layer 2	$\Gamma_{2,\delta_2}(\alpha_1)$	...	$\Gamma_{2,\delta_2}(\alpha_N)$	$\Gamma_{2,\delta_2}(\alpha_{N+1})$	...	$\Gamma_{2,\delta_2}(\alpha_{N+R_2-R_a})$	
$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\vdots$		
Layer a	$\Gamma_{a,\delta_a}(\alpha_1)$	...	$\Gamma_{a,\delta_a}(\alpha_N)$				

Example 2. An example of a 3-layer construction is shown in Fig. 3. We set $N = 5, R = 2, (R_1, R_2, R_3) = (5, 3, 2)$. In Fig. 3, only the coded matrices transmitted from Source 1 are listed, and those from Source 2 are similar. In Layer 1 ($A^{(1,1)} = A$), the coded matrices $f_1(\alpha_i; A^{(1,1)})$ are transmitted to Server i, $i \in [5]$, and $f_1(\alpha_{5+t}; A^{(1,1)}), t \in [3]$ are the extra parities. These parities are used in Layers 2 and 3. Specifically, $A^{(2,1)} = f_1(\alpha_7; A^{(1,1)})$ and $A^{(2,2)} = f_1(\alpha_8; A^{(1,1)})$ are used in Layer 2 and $A^{(3,1)} = f_1(\alpha_6; A^{(1,1)})$ is used in Layer 3. In Layer 2, $f_2(\alpha_i; A^{(2,\delta_2)}), \delta_2 \in [2], i \in [5]$ are encoded using the above extra parities from Layer 1. The generated extra parities $A^{(3,2)} = f_2(\alpha_6; A^{(2,1)})$ and $A^{(3,3)} = f_2(\alpha_6; A^{(2,2)})$ are used in Layer 3.

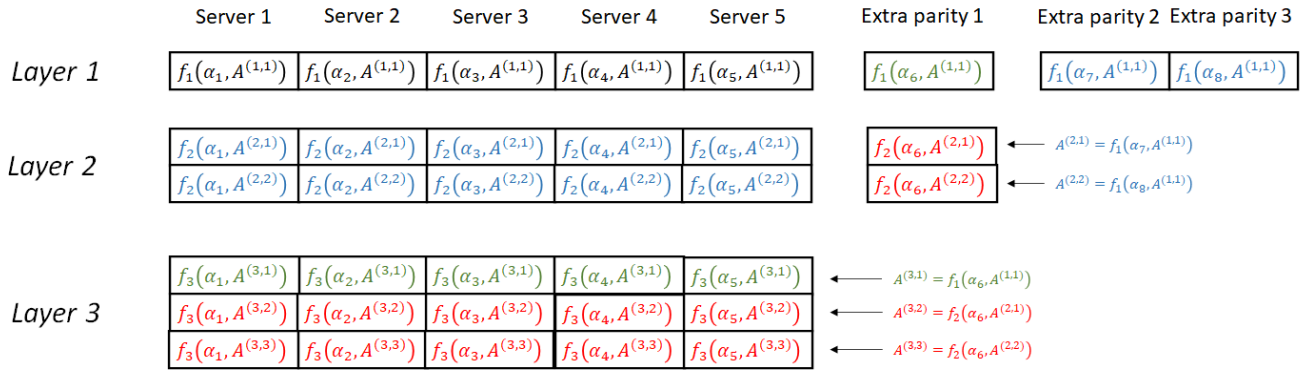

Fig. 3. Example of coded matrices for 3-layer construction, $N = R_1 = 5, R_2 = 3, R_3 = R = 2$.

Theorem 1, below, states the performance of the flexible construction in terms of storage and computation. This result is based on the following decoding strategy: in the presence of $\hat{R}$ available servers, $R_j \leq \hat{R} < R_{j-1}$, all tasks in Layers $1, 2, \ldots, j-1$ and some tasks in Layer j are executed. It

should be noted that the sum of storage sizes in all layers corresponds to the one-round communication model. However, under the multi-round communication model, the server storage size is only the maximum over the pairs of coded matrices. Since the coded matrix in a layer is encoded from sub-matrices in the previous layer, the higher the layer is, the smaller the size becomes. Hence, the storage size is just that of the first pair of coded matrices.

Theorem 1. In Construction 1, assume we have $\hat{R}$ available servers and $R \leq \hat{R} \leq N$, we need

$$L_{\text{flex}} = \begin{cases} L_1, & \hat{R} \geq R_1, \\ \left(1 + \frac{R_{j-1} - \hat{R}}{p_j m_j n_j}\right) \sum_{J=1}^{j-1} L_J, & R_j \leq \hat{R} < R_{j-1}, j \geq 2, \end{cases} \tag{25}$$

computation load at each server to obtain the final result, where

$$L_j = \begin{cases} \frac{\lambda\kappa\mu}{m_1 p_1 n_1}, & j = 1, \\ \frac{R_{j-1} - R_j}{p_j m_j n_j} \sum_{J=1}^{j-1} L_J, & j \geq 2, \end{cases} \tag{26}$$

is the total computation load at each server in Layer j. The server storage size required in Layer j is

$$C_j = C_{j,A} + C_{j,B}, \tag{27}$$

where

$$C_{j,A} = \begin{cases} \frac{\lambda\kappa}{p_1 m_1}, & j = 1, \\ \frac{R_{j-1} - R_j}{p_j m_j} \sum_{J=1}^{j-1} C_{J,A}, & j \geq 2. \end{cases} \tag{28}$$

$$C_{j,B} = \begin{cases} \frac{\kappa\mu}{p_1 n_1}, & j = 1, \\ \frac{R_{j-1} - R_j}{p_j n_j} \sum_{J=1}^{j-1} C_{J,B}, & j \geq 2. \end{cases} \tag{29}$$

Proof: In the following, we first prove (26). Then, we show that with the computation load in (25), the central node is able to obtain the matrix product. At last, we prove the storage size required in each layer.

In Layer $j = 1$, from (15), we know that $f_1(\alpha_i; A^{(1,1)})$ and $f_1(\alpha_i; B^{(1,1)})$ have sizes $\frac{\lambda}{m_1} \times \frac{\kappa}{p_1}$ and $\frac{\kappa}{p_1} \times \frac{\mu}{n_1}$, respectively. Thus, the computation load in Layer 1 is

$$L_1 = \frac{\lambda\kappa\mu}{m_1 p_1 n_1}. \tag{30}$$

In Layer j, according to (13), (14), (15), and (22), the computation load of $\{\Gamma_{j,\delta_j}(\alpha_i) = f_j(\alpha_i; A^{(j,\delta_j)}) \cdot g_j(\alpha_i; B^{(j,\delta_j)}) : \delta_j \in [\gamma_j]\}$ is $1/(p_j m_j n_j)$ fraction of that of $\mathcal{X} \triangleq \{A^{(j,\delta_j)} \cdot B^{(j,\delta_j)} : \delta_j \in [\gamma_j]\}$. Moreover,

the computation load of $\mathcal{Y}(J,t) \triangleq \{f_J(\alpha_{N+t} : A^{(J,\delta_J)}) \cdot f_J(\alpha_{N+t} : B^{(J,\delta_J)}) : \delta_J \in [\gamma_J]\}$ is equal to the load (per server) at the J-th layer, which is L_J. Based on (16), (17) and (18), the computation load of $\mathcal{X}$ is equal to the load of $\mathcal{Y}(J,t)$ for all $J \in [j-1], R_j - R_a + 1 \leq t \leq R_{j-1} - R_a$, which is $(R_{j-1} - R_j) \sum_{J=1}^{j-1} L_J$. Therefore, (26) is satisfied.

In the case that the number of available servers $\hat{R} \geq R_1$, according to the correctness of EP codes [8], the required results can be obtained by collecting R_1 evaluation points of $\Gamma_{1,1}(\alpha_i)$. Thus, we only need the computation in Layer 1.

In the case that $R_j \leq \hat{R} < R_{j-1}$, we first calculate all the tasks in Layers 1 to $j-1$, whose computation load is $\sum_{J=1}^{j-1} L_J$. Then, in Layer j, Server i calculates $\frac{R_{j-1}-\hat{R}}{R_{j-1}-R_j}\gamma_j$ tasks, i.e., $\Gamma_{j,\delta_j}(\alpha_i), \delta_j = 1, 2, ..., \frac{R_{j-1}-\hat{R}}{R_{j-1}-R_j}\gamma_j, i \in [N]$. Thus, the total computation is $\left(1 + \frac{R_{j-1}-\hat{R}}{p_j m_j n_j}\right) \sum_{J=1}^{j-1} L_J$.

Claim: R_J evaluations from $\{\Gamma_{J,\delta_J}(\alpha_i), i \in [N]\}$ can be obtained by the above calculations for $J = j, j-1, \ldots, 2, 1$.

We prove it by induction on J. As a consequence, the polynomial $\Gamma_{J,\delta_J}(\cdot)$ is decoded due to the correctness of EP codes [8]. Hence, the final result can be decoded with $J = 1$ and (25) is proved.

Base case: In Layer j, since $\hat{R} \geq R_j$, the claim holds trivially.

Induction step: Suppose the claim holds for Layers $j, j-1, \ldots, J+1$. We show that it will hold for Layer J. Note that $J < j$. The associated polynomials are decoded in Layers $j, j-1, \ldots, J+1$. Then, from Eqs. (16), (17) and (18), one can calculate $\Gamma_{J,\delta_J}(\alpha_{N+t})$, for $R_j - R_a + 1 \leq t \leq R_{j-1} - R_a - (\hat{R} - R_j)$ from Layer j and $R_{J'} - R_a + 1 \leq t \leq R_{J'-1} - R_a$ from Layers $J' = j-1, \ldots, J+1$. In total, $R_J - \hat{R}$ extra parities are obtained for the polynomial $\Gamma_{J,\delta_J}(\cdot)$. Thus, together with $\hat{R}$ available nodes, R_J evaluation points of $\Gamma_{J,\delta_J}(\cdot)$ are obtained, for all $\delta_J \in [\gamma_J]$.

The proof of the storage size is similar to the proof of (26). The proof sketch is as follows.

In Layer 1, the server needs to store $f_1(\alpha_i; A^{(1,1)}), f_1(\alpha_i; B^{(1,1)}), i \in [N]$, then

$$C_1 = \frac{1}{p_1}\left(\frac{\lambda\kappa}{m_1} + \frac{\kappa\mu}{n_1}\right). \tag{31}$$

In Layer $j \geq 2$, from (12), (17) and (18), the γ_j tasks in Layer j are encoded from the extra parities in Layers 1 to $j-1$. Based on (13), (14), (15), and (22), the size of $f_j(\alpha_i; A^{(j,\delta_j)}), i \in [N]$ is $p_j m_j$ fractions of $A^{(j,\delta_j)}$, and the size of $f_j(\alpha_i; B^{(j,\delta_j)}), i \in [N]$ is $p_j n_j$ fractions of $B^{(j,\delta_j)}$. Thus, (28) and (29) are obtained. ■

Remark 1. In Fig. 3, partial computation results can be also utilized to accelerate the computation in several cases such that the servers contribute different number of results depending on their speed. For

example, when Servers 1 and 2 complete their first 4 tasks and Server 3 completes its first 2 tasks, we are able to obtain $f_1(\alpha_i, A^{(1,1)})$ for $i = 1, 2, 3, 6, 7$, thus obtain the final results. Similar partial results utilization can be found in our general constructions, but in this paper we assume a server is either available or not able to provide any results.

Remark 2. CSA codes and GCSA codes [33] are designed to handle batch processing of matrix multiplication, namely, the multiplication of two sequences of matrices. They also provide solutions for secure distributed computation. Combined with these codes, our construction can be easily modified to handle batch processing and secure distributed computation.

Remark 3. In [8], the authors proposed an improved version of the EP code, which achieves a recovery threshold of $2r(p, m, n) - 1$. Here $r(p, m, n)$ denotes the bilinear complexity [56] for multiplying two matrices of sizes $m \times p$ and $p \times n$. It is well known that $r(p, m, n)$ is subcubic, i.e, $r(p, m, n) = o(pmn)$ when p, m, and n are large. The code in our construction can be replaced by the improved subcubic method. In this case, the recovery profile $(R_1, R_2, \cdots, R_a)$ and partitioning parameters p_j, m_j, n_j satisfy $R_j = 2r(p_j, m_j, n_j), j \in [a]$.

The following corollary states a special case of the computation load that will be useful in the optimization discussed in Section IV under the multi-round communication model.

Corollary 2. In the case of $p_j = 1, j \geq 2$, we have $m_j n_j = R_j$ in Construction 1. The j-th layer's computation load of each server is

$$L_j = \begin{cases} \frac{\lambda\kappa\mu}{m_1 p_1 n_1}, & j = 1, \\ \frac{R_1(R_{j-1} - R_j)}{R_{j-1} R_j} L_1, & j \geq 2, \end{cases} \tag{32}$$

and the total computation of each server is

$$L_{\text{flex}} = \begin{cases} L_1, & R_1 \leq \hat{R}, \\ \frac{R_1(R_j + R_{j-1} - \hat{R})}{R_{j-1} R_j} L_1, & R_j \leq \hat{R} < R_{j-1}, j \geq 2, \end{cases} \tag{33}$$

where $\hat{R}$ is the number of non-straggler servers. Specifically, when $\hat{R} = R_j$,

$$L_{\text{flex}} = \frac{R_1}{R_j} L_1. \tag{34}$$

Proof: We first prove (32) by induction.

Base case: When $j = 2$, we get $L_2 = \frac{R_1 - R_2}{R_2} L_1$ from (26) and it satisfies (32).

Induction step: Suppose $L_2, ..., L_j$ satisfy (32). From (26) and $p_j m_j n_j = R_j$ we know that

$$L_j = \frac{R_{j-1} - R_j}{R_j} \sum_{J=1}^{j-1} L_J. \tag{35}$$

Then, we have

$$\begin{aligned} L_{j+1} &= \frac{R_j - R_{j+1}}{R_{j+1}} \sum_{J=1}^{j-1} L_J + \frac{R_j - R_{j+1}}{R_{j+1}} L_j \\ &= \frac{R_j - R_{j+1}}{R_{j+1}} \frac{R_j}{R_{j-1} - R_j} L_j + \frac{R_j - R_{j+1}}{R_{j+1}} L_j \\ &= \frac{R_{j-1}(R_j - R_{j+1})}{R_{j+1}(R_{j-1} - R_j)} L_j \\ &= \frac{R_1(R_j - R_{j+1})}{R_j R_{j+1}} L_1, \end{aligned} \tag{36}$$

which satisfies (32).

Then, for the total computation, from (25) we can easily check that for $R_j \leq \hat{R} < R_{j-1}, j \in [a]$, we have

$$\begin{aligned} L_{\text{flex}} &= \left(1 + \frac{R_{j-1} - \hat{R}}{R_j}\right) \sum_{J=1}^{j-1} L_J \\ &= \left(1 + \frac{R_{j-1} - \hat{R}}{R_j}\right) \frac{R_j}{R_{j-1} - R_j} L_j \\ &= \frac{R_1(R_j + R_{j-1} - \hat{R})}{R_{j-1} R_j} L_1. \end{aligned} \tag{37}$$

The proof is completed. ■

IV. Computation Load Optimization

In this section, we discuss how to pick the matrix partition parameters and the recovery profile to optimize the computation load given the storage capacity. Under the one-round communication model, we find the optimal parameters for the 2-layer flexible construction and show that when the storage capacity is above a threshold, the flexible construction outperforms the fixed EP code. For the multi-round communication model, we show that all layers except the first layer reduce to block-wise matrix-vector multiplication, and when the straggler probability is small, the most number of layers is optimal.

Recall $\hat{R}$ is the number of non-straggler servers, $R \leq \hat{R} \leq N$. We consider the expected computation load over the realizations of $\hat{R}$. Assume for each instance of computing, $\hat{R}$ is independent and identically distributed. Denote q_j as the probability of j stragglers in the system. Formally,

$$q_j = P(\hat{R} = N - j), \quad \forall j \in \{0, , 1, \cdots, N - R\}. \tag{38}$$

Here, R is chosen such that the probability of having more than $N - R$ stragglers is negligible. Therefore, j is assumed to be in the range between 0 and $N - R$, and

$$\sum_{j=0}^{N-R} q_j = 1. \tag{39}$$

The expectation of the computation load is

$$E[L_{\text{flex}}] = \sum_{j=0}^{N-R} q_j L_{\text{flex}}(\hat{R} = N - j), \tag{40}$$

where $L_{\text{flex}}(\hat{R})$ is the computation load for $\hat{R}$ non-straggler servers. The goal is to minimize $E[L_{\text{flex}}]$ over the partitioning parameters p_j, m_j, n_j, $j \in [a]$ and the recovery profile $\{R_1, \cdots, R_a\}$, given the recovery threshold $R_a = R$ and the storage constraint C in each server. Although in practical systems $p_j, m_j, n_j, R_j, j \in [a]$ are required to be integers, in this section, we only assume them as real numbers to simplify the optimization analysis. To find an integer solution (not necessarily optimal), we pick the parameters close to the optimal real values that satisfy the recovery threshold and the storage constraint.

A. Optimization on Entangled Polynomial Codes

As a warm-up, let us start with an EP code with a fixed recovery threshold R, which satisfies $R = m_0 p_0 n_0 + p_0 - 1$ according to [8], for some undetermined partition parameters p_0, m_0, n_0. The computation load of EP codes remains the same if the number of stragglers is no greater than $N - R$. According to [8], the computation load and the required storage size are

$$L_{\text{EP}}(\hat{R}) = \frac{\lambda \kappa \mu}{m_0 p_0 n_0}, \hat{R} \geq R, \tag{41}$$

$$C_{\text{EP}} = \frac{1}{p_0} \left(\frac{\lambda \kappa}{m_0} + \frac{\kappa \mu}{n_0} \right). \tag{42}$$

Thus, the optimization problem can be formulated as

$$\begin{aligned} \min_{p_0,m_0,n_0} \quad & L_{\text{EP}} = \frac{\lambda\kappa\mu}{m_0 p_0 n_0}, \\ \text{s.t.} \quad & R = p_0 m_0 n_0 + p_0 - 1, \\ & \frac{\lambda\kappa}{p_0 m_0} + \frac{\kappa\mu}{p_0 n_0} \leq C. \end{aligned} \tag{43}$$

Theorem 2. The solution of the EP code optimization problem in (43) is

$$L_{\text{EP}}^* = \frac{2C\lambda\kappa\mu}{C(R+1) + \sqrt{C^2(R+1)^2 - 16\lambda\kappa^2\mu}} \tag{44}$$

with

$$p_0^* = \frac{1}{2}(R+1) - \frac{1}{2}\sqrt{(R+1)^2 - 16\frac{\lambda\kappa^2\mu}{C^2}}, \tag{45}$$

and m_0^*, n_0^* are given by $m_0^* n_0^* = \frac{R+1}{p_0^*} - 1$ and $\lambda\kappa n_0^* = \kappa\mu m_0^*$.

Proof: We will show that the computation load increases with p_0, so we will find the minimal p_0 satisfying the storage constraint.

Using the threshold constraint

$$m_0 n_0 = \frac{R+1}{p_0} - 1, \tag{46}$$

we have $L_{\text{EP}} = \frac{\lambda\kappa\mu}{R+1-p_0}$, which is an increasing function of p_0. So, we minimize p_0 under the constraint that

$$\frac{\lambda\kappa n_0 + \kappa\mu m_0}{R+1-p_0} \leq C. \tag{47}$$

Note that

$$\lambda\kappa n_0 + \kappa\mu m_0 \geq 2\sqrt{\lambda\kappa^2\mu m_0 n_0} = 2\sqrt{\lambda\kappa^2\mu\frac{R+1-p_0}{p_0}} \tag{48}$$

and it holds with equality if and only if $\lambda\kappa n_0 = \kappa\mu m_0$. Combining (47) with (48) results in

$$2\sqrt{\frac{\lambda\kappa^2\mu}{(R+1-p_0)p_0}} \leq C. \tag{49}$$

Note that $2\sqrt{\frac{\lambda\kappa^2\mu}{(R+1-p_0)p_0}}$ decreases with p_0 because the derivative

$$\frac{\mathrm{d}\,(R+1-p_0)p_0}{\mathrm{d}\,p_0} = R+1-2p_0 = p_0 m_0 n_0 - p_0 \geq 0, \tag{50}$$

here we consider m_0, n_0 as integers thus $m_0 n_0 \geq 1$. Therefore, L_{EP} reaches its optimal value when $\lambda\kappa n_0 = \kappa\mu m_0$ and (49) holds with equality, i.e., $p_0^* = \frac{1}{2}(R+1) - \frac{1}{2}\sqrt{(R+1)^2 - 16\frac{\lambda\kappa^2\mu}{C^2}}$. As a result, m_0^*, n_0^* can be obtained by $m_0^* n_0^* = \frac{R+1}{p_0^*} - 1$ and $\lambda\kappa n_0^* = \kappa\mu m_0^*$. The optimal computation load is

$$L_{EP}^* = \frac{2C\lambda\kappa\mu}{C(R+1) + \sqrt{C^2(R+1)^2 - 16\lambda\kappa^2\mu}}. \tag{51}$$

The theorem is proved. ■

Remark 4. In Theorem 2, the storage capacity is required to satisfy

$$C \geq \frac{4\kappa\sqrt{\lambda\mu}}{1+R}. \tag{52}$$

to have a valid L_{EP}^* in (44). In addition, by combining (47) and (48), we obtain the minimum storage required as $2\sqrt{\frac{\lambda\kappa^2\mu}{(R+1-p_0)p_0}}$ in (49). Since $p_0 = \frac{R+1}{m_0 n_0 + 1}$ and m_0, n_0 are at least 1, we conclude that (52) is the minimum storage constraint requirement to use EP codes for distributed matrix multiplication.

B. Optimization for the One-round Communication Model

Next, we consider the flexible constructions with the one-round communication model. In this model, all the tasks are sent to the server in one communication round. Thus, the sum of the task sizes should not exceed the storage constraint. Since the more layers, the larger the total size of the tasks is, only the 2-layer construction is considered. We first optimize the partition parameters with predetermined recovery profile $(R_1, R_2 = R)$ and capacity C. After that, R_1 is optimized given $R_2 = R$ and C.

By the expression of the computation load in Theorem 1, the expectation of the computation load in (40) becomes

$$E[L_{\text{flex}}] = \sum_{j=0}^{N-R_2} q_j \frac{\lambda\kappa\mu}{p_1 m_1 n_1} + \sum_{j=N-R_1+1}^{N-R_2} q_j \frac{\lambda\kappa\mu(R_1+j-N)}{m_1 m_2 p_1 p_2 n_1 n_2}. \tag{53}$$

In practical systems, the probability of having many stragglers is usually small. For instance, less than 110 failures occur over a 3000-node production cluster of Facebook per day [57]. So, we ignore the second term in (53) and use the approximation $L_{\text{flex}} = \frac{\lambda\kappa\mu}{p_1 m_1 n_1}$ in our optimization problem. Combined with (27), the optimization problem can be formulated as

$$\begin{aligned} \min_{p_1,m_1,n_1,p_2,m_2,n_2} \quad & L_{\text{flex}} = \frac{\lambda\kappa\mu}{p_1 m_1 n_1}, \\ \text{s.t.} \quad & R_1 = p_1 m_1 n_1 + p_1 - 1, \\ & R_2 = p_2 m_2 n_2 + p_2 - 1, \\ & \frac{1}{p_1}\left(\frac{\lambda\kappa}{m_1} + \frac{\kappa\mu}{n_1}\right) + \frac{(R_1 - R_2)}{p_1 p_2}\left(\frac{\lambda\kappa}{m_1 m_2} + \frac{\kappa\mu}{n_1 n_2}\right) \leq C. \end{aligned} \tag{54}$$

It should be noted that when $R_1 = R_2 = R$, the 2-layer flexible construction reduces to the fixed EP code and the optimal partition parameters remain the same.

Theorem 3. Fix $R_1, R_2 = R$. The solution of the 2-layer flexible construction optimization (54) is

$$L^*_{\text{flex}} = \frac{2C(R_2+1)\lambda\kappa\mu}{C(R_1+1)(R_2+1) + \sqrt{C^2(R_1+1)^2(R_2+1)^2 - 16\lambda\kappa^2\mu(2R_1-R_2+1)^2}}, \tag{55}$$

with

$$p_1^* = \frac{1}{2}(R_1+1) - \frac{1}{2}\sqrt{(R_1+1)^2 - \frac{16\lambda\kappa^2\mu(2R_1-R_2+1)^2}{C^2(R_2+1)^2}}, \tag{56}$$

and m_1^*, n_1^* are given by $m_1^* n_1^* = \frac{R_1+1}{p_1^*} - 1$ and $\lambda\kappa n_1^* = \kappa\mu m_1^*$, and $p_2^* = \frac{R_2+1}{2}, m_2^* = 1, n_2^* = 1$.

Proof: Using $m_1 n_1 = \frac{R_1+1}{p_1} - 1$, we have $L_{\text{flex}} = \frac{\lambda\kappa\mu}{R_1+1-p_1}$, which is an increasing function of p_1. Therefore, to maximize L_{flex} we need to minimize p_1.

Using $m_1 n_1 = \frac{R_1+1}{p_1} - 1$ and $m_2 n_2 = \frac{R_2+1}{p_2} - 1$, similar to (47) and (48), we have:

$$\begin{aligned} & C \\ \geq & \frac{1}{p_1}\left(\frac{\lambda\kappa}{m_1} + \frac{\kappa\mu}{n_1}\right) + \frac{(R_1-R_2)}{p_1 p_2}\left(\frac{\lambda\kappa}{m_1 m_2} + \frac{\kappa\mu}{n_1 n_2}\right) && (57) \\ \geq & 2\sqrt{\frac{\lambda\kappa^2\mu}{(R_1+1-p_1)p_1}} + 2(R_1-R_2)\sqrt{\frac{\lambda\kappa^2\mu}{(R_1+1-p_1)(R_2+1-p_2)p_1 p_2}}. && (58) \end{aligned}$$

Here (57) holds with equality when $\lambda\kappa n_1 = \kappa\mu m_1$ and $\lambda\kappa n_1 n_2 = \kappa\mu m_1 m_2$ or $n_2 = m_2$. Similar to (50), it is easy to show that (58) is a decreasing function of p_1 and p_2. For any fixed p_2, to obtain the minimum p_1, we should set (58) equal to C. When (58) is fixed, p_1 is minimized when p_2 reaches its maximum because a bigger p_2 results in a smaller p_1 when (58) is equal to C. Noticing that $p_2 = \frac{R_2+1}{m_2 n_2+1}$ and m_2, n_2 are at least 1, we set $p_2^* = \frac{R_2+1}{2}, m_2^* = 1, n_2^* = 1$. The optimal p_1^* and L^*_{flex} are obtained accordingly. ■

If R_2 is odd, then the choices of p_2, m_2, n_2 in the above theorem are the exact optimal integer parameters. Otherwise, we pick m_2, n_2 to set p_2 as the largest integer solution of $p_2 = \frac{R_2+1}{m_2 n_2+1}$.

Remark 5. In Theorem 3, the storage capacity is required to satisfy

$$C \geq \frac{4\kappa\sqrt{\lambda\mu}(2R_1 - R_2 + 1)}{(1+R_2)(1+R_1)} \tag{59}$$

to have a valid L^*_{flex} in (55). In addition, we obtain the minimum storage required in (58). Since $p_1 = \frac{R_1+1}{m_1 n_1+1}, p_2 = \frac{R_2+1}{m_2 n_2+1}$, and m_1, n_1, m_2, n_2 are at least 1, we conclude that (59) is the minimum

storage constraint requirement to use our 2-layer flexible codes for the distributed matrix multiplication. When $R_1 = R_2$, (59) is the same as (52).

Next, we provide an example for the optimal integer solutions of the partition parameters.

Example 3. Assume there are $N = 8$ servers and we need to tolerate $N - R = 1$ straggler. $\lambda = \kappa = \mu$ and the storage size of each server is limited by $C = \frac{8}{7}\lambda\kappa$. Using the EP code, the optimal choice of $\{p_0, m_0, n_0\}$ is $\{1, 1, 7\}$, which results in a storage size of $\frac{8}{7}\lambda\kappa$ and a computation load per server of $\frac{1}{7}\lambda\kappa\mu = 0.143\lambda\kappa\mu$. Using the 2-layer flexible construction with $R_1 = 8$ and $R_2 = 7$, the optimal parameters are chosen as $p_1 = 1, m_1 = 2, n_1 = 4, p_2 = 4, m_2 = 1, n_2 = 1$, which requires a storage size of $\frac{15}{16}\lambda\kappa$ and a computation load of $\frac{1}{8}\lambda\kappa\mu$ when there is no straggler, with an additional computation load of $\frac{1}{32}\lambda\kappa\mu$ when there is one straggler. Assuming the probability of one straggler to be 10%, the average computation load is $0.128\lambda\kappa\mu$. In this example, we save both storage size and average computation load while maintaining one straggler tolerance.

Having found the best computation load for a fixed recovery profile as in Theorem 2, next, we discuss the optimization of the recovery profile. Given the straggler tolerance level $N - R_2$, we just need to optimize R_1, such that $R_2 \leq R_1 \leq N$.

Theorem 4. To minimize the 2-layer computation load L^*_{flex} in (55), the optimal R_1^* is

$$R_1^* = \begin{cases} N, & C \geq \frac{8\kappa\sqrt{\lambda\mu}}{R_2+1}, \\ \min\left(N, \frac{C^2(R_2+1)^2(R_2+3)+64\lambda\kappa^2\mu(R_2-1)}{2(64\lambda\kappa^2\mu-C^2(R_2+1)^2)}\right), & \frac{8\kappa}{R_2+1}\sqrt{\frac{\lambda\mu}{3}} < C < \frac{8\kappa\sqrt{\lambda\mu}}{R_2+1}, \\ R_2, & C \leq \frac{8\kappa}{R_2+1}\sqrt{\frac{\lambda\mu}{3}}. \end{cases} \tag{60}$$

Proof: The optimal computation given R_1 is shown in (55). Since the numerator is a constant not related to R_1, we set Y as the denominator and L^*_{flex} has the minimum value when Y reaches its maximum.

$$\frac{dY}{dR_1} = C(R_2+1) + \frac{C^2(R_2+1)^2(R_1+1) - 32\lambda\kappa^2\mu(2R_1 - R_2 + 1)}{\sqrt{C^2(R_1+1)^2(R_2+1)^2 - 16\lambda\kappa^2\mu(2R_1 - R_2 + 1)^2}}. \tag{61}$$

Setting $\frac{dY}{dR_1} = 0$, we have

$$\left(\frac{32\lambda\kappa^2\mu(2R_1 - R_2 + 1)}{C(R_2+1)}\right)^2 = 64\lambda\kappa^2\mu(2R_1 - R_2 + 1)(R_1 + 1) - 16\lambda\kappa^2\mu(2R_1 - R_2 + 1)^2. \tag{62}$$

Since $R_1 \geq R_2$, the term $\lambda\kappa^2\mu(2R_1 - R_2 + 1) \neq 0$ can be cancelled and the solution to the above equation is

$$\hat{R_1} \triangleq \frac{C^2(R_2+1)^2(R_2+3) + 64\lambda\kappa^2\mu(R_2-1)}{2(64\lambda\kappa^2\mu - C^2(R_2+1)^2)}. \tag{63}$$

Let $X = C^2(R_1+1)^2(R_2+1)^2 - 16\lambda\kappa^2\mu(2R_1 - R_2 + 1)^2$. We have $X \geq 0$ due to the minimum storage constraint in Remark 5. We simplify (61) as

$$\frac{dY}{dR_1} = C(R_2+1) + \frac{\frac{dX}{dR_1}}{2\sqrt{X}}, \tag{64}$$

where

$$\frac{dX}{dR_1} = 2(C^2(R_2+1)^2 - 64\lambda\kappa^2\mu)R_1 + 2C^2(R_2+1)^2 + 64\lambda\kappa^2\mu(R_2-1) \tag{65}$$

is a linear function of R_1 and the constant term $2C^2(R_2+1)^2 + 64\lambda\kappa^2\mu(R_2-1) > 0$ since R_2 is at least 1.

In the case of $C \geq \frac{8\kappa\sqrt{\lambda\mu}}{R_2+1}$, we have

$$C^2(R_2+1)^2 - 64\lambda\kappa^2\mu \geq 0 \Rightarrow \frac{dX}{dR_1} > 0 \Rightarrow \frac{dY}{dR_1} > 0. \tag{66}$$

Thus, we should pick $R_1^* = N$.

In the case of $C < \frac{8\kappa\sqrt{\lambda\mu}}{R_2+1}$, we get

$$C^2(R_2+1)^2 - 64\lambda\kappa^2\mu < 0 \Rightarrow \frac{dX}{dR_1} \text{ is a decreasing linear function.} \tag{67}$$

We discuss $\frac{dY}{dR_1}$ when R_1 varies between $(0, \frac{C^2(R_2+1)^2+32\lambda\kappa^2\mu}{64\lambda\kappa^2\mu - C^2(R_2+1)^2}]$ and $(\frac{C^2(R_2+1)^2+32\lambda\kappa^2\mu}{64\lambda\kappa^2\mu - C^2(R_2+1)^2}, +\infty)$ separately. In the first region, we have

$$R_1 \leq \frac{C^2(R_2+1)^2 + 32\lambda\kappa^2\mu}{64\lambda\kappa^2\mu - C^2(R_2+1)^2} \Rightarrow \frac{dX}{dR_1} \geq 0 \Rightarrow \frac{dY}{dR_1} > 0, \tag{68}$$

Y reach its maximum when $R_1 = \frac{C^2(R_2+1)^2+32\lambda\kappa^2\mu}{64\lambda\kappa^2\mu - C^2(R_2+1)^2}$. In the second region, we have

$$R_1 > \frac{C^2(R_2+1)^2 + 32\lambda\kappa^2\mu}{64\lambda\kappa^2\mu - C^2(R_2+1)^2} \Rightarrow \frac{dX}{dR_1} < 0. \tag{69}$$

Clearly, $\frac{\frac{dX}{dR_1}}{2\sqrt{X}}$ is a decreasing function of R_1, because $\frac{dX}{dR_1}$ is a negative decreasing function of R_1 by (67) and (69), and $\sqrt{X}$ is a positive decreasing function of R_1 by (69). Then, with (64) we can conclude that

$$\frac{d^2Y}{dR_1^2} = \frac{d\frac{\frac{dX}{dR_1}}{2\sqrt{X}}}{dR_1} < 0. \tag{70}$$

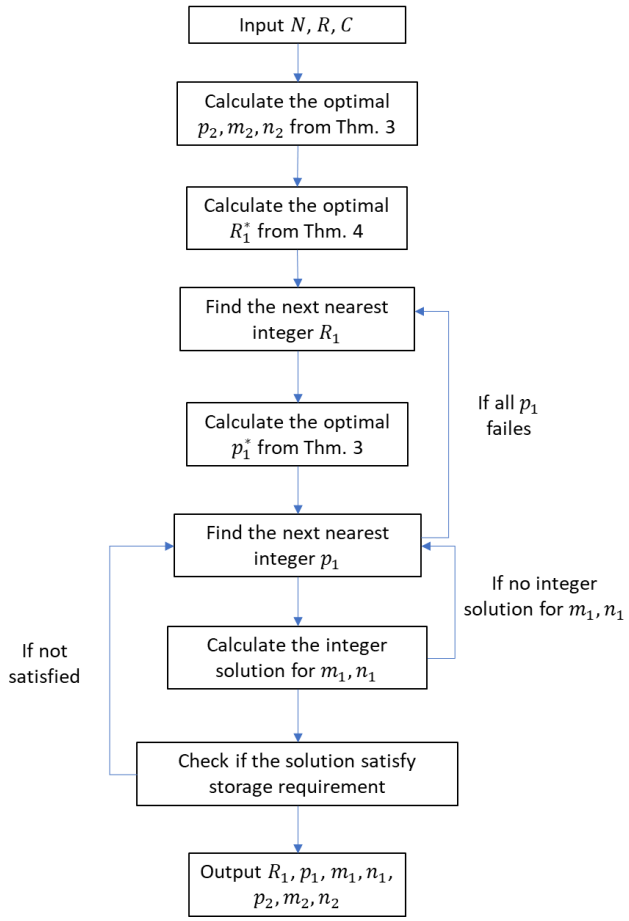

Fig. 4. Flowchart of choosing parameters for the two-layer flexible code.

In addition, we know from (63) that $\hat{R}_1$ is located in $(\frac{C^2(R_2+1)^2+32\lambda\kappa^2\mu}{64\lambda\kappa^2\mu-C^2(R_2+1)^2}, +\infty)$ when $R_2 \geq 2$, and hence is a local maximum. Therefore, combining the 2 ranges of R_1, we conclude that Y reaches its maximum in (63). Finally, the proof is completed considering the requirement that $R_2 \leq R_1 \leq N$, and the fact that $\hat{R}_1 \geq R_2$ is satisfied when $C \geq \frac{8\kappa}{R_2+1}\sqrt{\frac{\lambda\mu}{3}}$. ■

Now, we find the optimal solution of $p_j, m_j, n_j, j \in [2]$ and R_1 in Theorem 3 and Theorem 4, respectively. In practical systems, these parameters have to be chosen as integers, and the flowchart to choose the parameters is shown in Fig. 4.

Corollary 3. The flexible construction with 2 layers is better than a fixed EP code in terms of the computation load when the storage constraint C satisfy:

$$C > \frac{8\kappa}{R_2+1}\sqrt{\frac{\lambda\mu}{3}}. \tag{71}$$

TABLE III

OPTIMAL CHOICES OF THE FLEXIBLE CONSTRUCTIONS GIVEN THE NUMBER OF SERVERS N, THE FAILURE TOLERANCE $N-R$ AND THE STORAGE CONSTRAINT C.

Storage constraint C	Optimal constructions	Optimal matrix partition
$C < \frac{4\kappa\sqrt{\lambda\mu}}{1+R_2}$	Not available	Not available
$\frac{4\kappa\sqrt{\lambda\mu}}{1+R_2} \leq C \leq \frac{8\kappa}{R_2+1}\sqrt{\frac{\lambda\mu}{3}}$	Fixed EP codes	p_0, m_0, n_0 chosen in Theorem 2
$C > \frac{8\kappa}{R_2+1}\sqrt{\frac{\lambda\mu}{3}}$	Flexible codes with R_1 chosen in Theorem 4	$p_j, m_j, n_j, j \in [2]$ chosen in Theorem 3

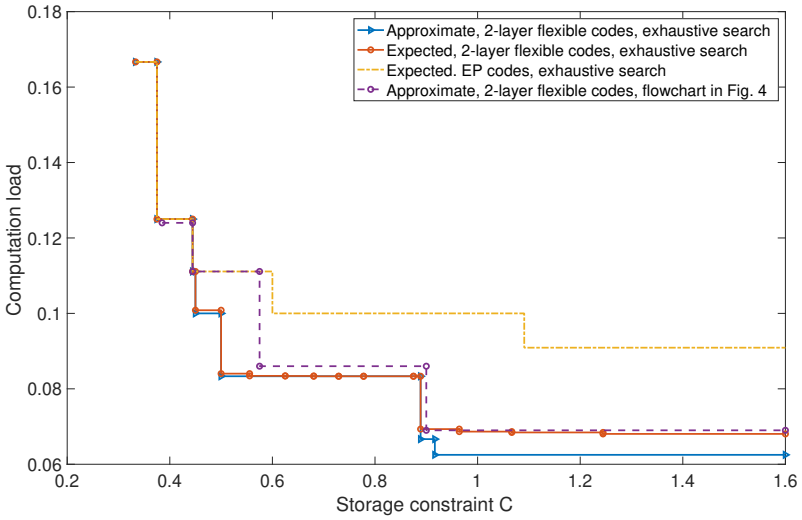

Fig. 5. Computation load comparison of our 2-layer flexible codes and fixed EP codes under different storage constraints. $N = 16, R = 11$, and $\lambda = \kappa = \mu = 1$ unit. Each server has a failure probability of 0.05. The approximate computation load in (55) and the expected computation load in (53) are both shown in the figure. The approximate computation load with parameters chosen from Fig. 4 is also shown.

Proof: From Theorems 2 and 3 we have

$$L^*_{\text{flex}}|_{R_1=R_2} = L^*_{\text{EP}}. \tag{72}$$

Also, it is easy to check that $R_1^* > R_2$ in (60) when (71) is satisfied. Then, combining Theorem 4 we conclude that

$$L^*_{\text{flex}}|_{R_1=R_1^*} < L^*_{\text{flex}}|_{R_1=R_2} = L^*_{\text{EP}}. \tag{73}$$

The proof is completed. ■

We summarize how to choose the optimal constructions in different situations in Table III.

Fig. 5 shows a comparison of our 2-layer flexible codes and the fixed EP codes. For the approximate computation load, only the computation load in the first layer is considered as in (54). The expected computation load is computed based on (53) with a truncated binomial distribution

$$q_j = \frac{1}{\theta}\binom{N}{j}(1-\epsilon)^{N-j}\epsilon^j, 0 \leq j \leq N-R, \tag{74}$$

where $\epsilon = 0.05$ is the probability that each server is a straggler. To limit the number of stragglers below $N-R$, we truncate the binomial distribution below $N-R$ and $\theta = \sum_{i=0}^{N-R}\binom{N}{i}\epsilon^i(1-\epsilon)^{N-i}$ is the probability that there are at least $N-R$ available nodes.[3] In Fig. 5, we have $1-\theta < 10^{-4}$. The minimum required storage constraint is $C = 0.33$. When $C < 0.45$, our 2-layer construction reduces to the EP code. When the storage constraint $C \geq 0.45$, our 2-layer constructions have better performance. For example, when $C = 0.9$, the optimal EP code has $p_0 = 2, m_0 = 1, n_0 = 5$ and its expected computation is $L_{\text{EP}} = 0.1$. However, our 2-layer optimal flexible code has $p_1 = 1, m_1 = 3, n_1 = 5, p_2 = 6, m_2 = 1, n_2 = 1, R_1 = 15,$ and its expected computation load is $L_{\text{flex}} = 0.069$, i.e., we save more than 30% in terms of computation load. In addition, the approximate computation load of the 2-layer flexible code in this case is 0.067, which is very close to the expected computation load when the computation in both layers are considered. The computation load with parameters chosen from Fig. 4 is also shown. It does not exactly match the exhaustive search results because we need integer solution for all the parameters. However, it shows a similar trend as exhaustive search.

C. Optimization for the Multi-round Communication Model

Now let us consider the multi-round communication model where coded matrices are sent sequentially from the source to the server. In this case, it is only required that the maximum size of the coded matrices does not exceed the storage size. As mentioned before Theorem 1, the storage capacity just needs to exceed the size of the first pair of coded matrices. We first optimize the partitioning parameters for a fixed recovery profile and then optimize the number of layers and the recovery profile.

Let us consider the construction with a layers and predetermined $R_j, j \in [a]$ such that $N \geq R_1 > R_2 > ... > R_a = R$. Assuming $R_j, j \in [a]$, and the storage constraint C are given, we first minimize

[3]In practice, R is chosen such that the probability of having more than $N-R$ stragglers is negligible.

the computation load in each layer:

$$\begin{aligned} \min_{p_j,m_j,n_j} \quad & L_j \\ \text{s.t.} \quad & R_j = p_j m_j n_j + p_j - 1 \\ & \frac{\lambda\kappa}{p_1 m_1} + \frac{\kappa\mu}{p_1 n_1} \leq C, \end{aligned} \tag{75}$$

where L_j is shown in (26). Note that once $L_j, j \in [a]$, are minimized, by Theorem 1 the computation load $L_{\text{flex}}(\hat{R})$ for any number of non-stragglers $\hat{R}$ is also minimized. Hence the optimization in (75) is stronger than optimizing the expected computation load defined in (40).

Theorem 5. The optimal solution of (75) for the flexible construction under the multi-round communication model is

$$L_j^* = \begin{cases} \frac{2C\lambda\kappa\mu}{C(R+1)+\sqrt{C^2(R+1)^2-16\lambda\kappa^2\mu}}, & j = 1, \\ \frac{R_1(R_{j-1}-R_j)}{R_{j-1}R_j} L_1, & j \geq 2, \end{cases} \tag{76}$$

with

$$p_1^* = \frac{1}{2}(R+1) - \frac{1}{2}\sqrt{(R+1)^2 - \frac{16\lambda\kappa^2\mu}{C^2}}, \tag{77}$$

m_1^*, n_1^* are given by $m_1^* n_1^* = \frac{R+1}{p_1^*} - 1$ and $\lambda\kappa n_1^* = \kappa\mu m_1^*$ and $p_j^* = 1, m_j^* n_j^* = R_j$ for $j \geq 2$.

Proof: When $j = 1$, it is the same optimization problem in Theorem 2.

For $j \geq 2$, We prove by induction.

Base case: For $j = 2$, since there is no constraint on storage size of Layer 2, by Theorem 1 we have

$$L_2 = \frac{R_1 - R_2}{p_2 m_2 n_2} L_1 = \frac{R_1 - R_2}{R_2 - p_2 + 1} L_1, \tag{78}$$

which is an increasing function of p_2. Thus, we have $p_2^* = 1, m_2^* n_2^* = R_2$.

Induction step: Assume the minimum L_J^* is achieved when $p_J^* = 1, m_J^* n_J^* = R_J$ for $J = 2, 3, ..., j-1$. For $J = j$, we have

$$L_j = \frac{R_{j-1} - R_j}{p_j m_j n_j} \sum_{J=1}^{j-1} L_J = \frac{R_{j-1} - R_j}{R_j - p_j + 1} \sum_{J=1}^{j-1} L_J, \tag{79}$$

which is an increasing function of p_j and $L_J, J \in [j-1]$, respectively. Hence, we should pick the minimum $p_j^* = 1$ and the minimum $L_J^*, 1 \leq J \leq j-1$ to optimize L_j. Therefore, $p_J^* = 1, m_J^* n_J^* = R_J$ for all $2 \leq J \leq j$. ■

Notice that in the case of $j \geq 2$, we have $R_j = m_j n_j$, there is at least one integer solution with $m_j = R_j, n_j = 1$, which simplifies the problem to be matrix-vector multiplication.

Next, we discuss how to set the number of layers and the recovery profile to minimize the computation load. First, we state a lemma to show that adding more layers does not increase the computation load. Then, a theorem is presented on show how to set R_1.

Lemma 1. Given R, R_1, and $p_j = 1, j \geq 2$, adding another layer does not increase the computation load of each server.

Proof: Let us add a layer between Layers $j-1$ and j. When $R_{j-1} = R_j + 1$, no layers can be added. When $R_{j-1} > R_j + 1$, consider adding one extra layer with $R_{\text{add}} = R_j + 1$ between Layer $j-1$ and j so that $R_j < R_{\text{add}} < R_{j-1}$. Denote the computation load of the new construction by L_{add}, which is a function of the number of non-stragglers, $\hat{R}$.

When $\hat{R} \leq R_j$ or $\hat{R} \geq R_{j-1}$, based on (33), the computation load of each server does not change, i.e., $L_{\text{add}} = L_{\text{flex}}$.

When $R_{\text{add}} \leq \hat{R} < R_{j-1}$, by Corollary 2, the new computation load is

$$\begin{aligned} L_{\text{add}} &= \frac{R_1(R_{\text{add}} + R_{j-1} - \hat{R})}{R_{j-1}R_{\text{add}}} L_1 \\ &= \frac{R_1}{R_{j-1}} L_1 + \frac{R_1(R_{j-1} - \hat{R})}{R_{j-1}R_{\text{add}}} L_1, \\ &< \frac{R_1}{R_{j-1}} L_1 + \frac{R_1(R_{j-1} - \hat{R})}{R_{j-1}R_j} L_1 \\ &= L_{\text{flex}}, \end{aligned} \tag{80}$$

where the inequality results from the fact that $R_{\text{add}} > R_j$. Thus, by adding one layer with $R_{\text{add}} = R_j + 1$, computation load does not increase. Similarly, more layers can be added between Layer $j-1$ and j. Therefore, adding more layers between R_1 and R does not increase the computation load of each server. ■

Based on Lemma 1, given R_1 and R, the optimal scheme is to add one layer for each value between R_1 and R. Thus, the recovery profile should be chosen to be $(R_1, R_1 - 1, R_1 - 2, \ldots, R)$. The only problem left is how to set R_1. According to Corollary 2 and Theorem 5,

$$L_{\text{flex}} = \begin{cases} L_1 = \frac{2\lambda\kappa\mu}{(R+1)+\sqrt{(R+1)^2 - \frac{16\lambda\kappa^2\mu}{C^2}}}, & \text{if } \hat{R} > R_1, \\ \frac{R_1}{N-j} L_1, & \text{if } \hat{R} = N - j, N - R_1 \leq j \leq N - R. \end{cases} \tag{81}$$

Based on (81), we see $\frac{\lambda\kappa\mu}{R_1+1} < L_1 < \frac{2\lambda\kappa\mu}{R_1+1}$. Denote $L_1 = \eta\frac{\lambda\kappa\mu}{R_1+1}$, where

$$\eta = \frac{2}{1+\sqrt{1-\frac{16\lambda\kappa^2\mu}{C^2(1+R_1)^2}}}. \tag{82}$$

Note that given λ, κ, μ, C, the value of η decreases as R_1 increases. Then, for fixed R_1, the expectation of the computation load is

$$E\left[L_{\text{flex}}^{(R_1)}\right] = L_1 \sum_{j=0}^{N-R_1-1} q_j + \sum_{j=N-R_1}^{N-R} q_j \frac{R_1}{N-j} L_1 \tag{83}$$

$$= \lambda\kappa\mu\eta \left(\sum_{j=0}^{N-R_1-1} \frac{q_j}{1+R_1} + \sum_{j=N-R_1}^{N-R} \frac{q_j R_1}{(N-j)(1+R_1)} \right). \tag{84}$$

Here, the superscript R_1 indicates that the computation load depends on R_1. The goal is to minimize $E\left[L_{\text{flex}}^{(R_1)}\right]$ over R_1 where $R \le R_1 \le N$.

The theorem below states a sufficient condition for which we should set $R_1 = N$ and use the maximum number of layers. In particular, the recovery profile should be $(N, N-1, N-2, \ldots, R)$.

Theorem 6. When $q_0 > \sum_{j=1}^{N-R} \frac{q_j}{N-j}$, the optimal R_1 to minimize (84) is achieved when $R_1^* = N$.

Proof: Denote the term in the parentheses of (84) as

$$h(R_1) = \frac{1}{1+R_1} \sum_{j=0}^{N-R_1-1} q_j + \frac{R_1}{1+R_1} \sum_{j=N-R_1}^{N-R} \frac{q_j}{N-j}. \tag{85}$$

When $R_1 = N$,

$$h(N) = \frac{N}{1+N} \sum_{j=0}^{N-R} \frac{q_j}{N-j}. \tag{86}$$

When $R_1 = N-k, k \in [1, N-R]$,

$$h(N-k) = \frac{1}{N-k+1} \sum_{j=0}^{k-1} q_j + \frac{N-k}{N-k+1} \sum_{j=k}^{N-R} \frac{q_j}{N-j}. \tag{87}$$

Then, their difference is

$$h(N-k) - h(N) \tag{88}$$

$$= \frac{1}{N-k+1} \sum_{j=0}^{k-1} q_j + \frac{N-k}{N-k+1} \sum_{j=k}^{N-R} \frac{q_j}{N-j} - \frac{N}{1+N} \sum_{j=0}^{N-R} \frac{q_j}{N-j} \tag{89}$$

$$= \frac{1}{(N+1)(N-k+1)} \left(\sum_{j=0}^{k-1} \frac{(k-j)N-j}{N-j} q_j - k \sum_{j=k}^{N-R} \frac{q_j}{N-j} \right). \tag{90}$$

When $q_0 > \sum_{j=1}^{N-R} \frac{q_j}{N-j}$, the first term in the parentheses of (90) is

$$\sum_{j=0}^{k-1} \frac{(k-j)N-j}{N-j} q_j \geq kq_0 > k \sum_{j=1}^{N-R} \frac{q_j}{N-j} \geq k \sum_{j=k}^{N-R} \frac{q_j}{N-j}. \tag{91}$$

Thus, $h(N-k) > h(N)$. Since η increases as R_1 decreases, by (84) we conclude that $E\left[L_{\text{flex}}^{(N)}\right] < E\left[L_{\text{flex}}^{(N-k)}\right]$. Therefore, R_1^* should be set as N. ■

Example 4. Suppose $N = 50, R = 40$ and assume the number of stragglers follows a truncated binomial distribution similar to (74), i.e., $q_j = \theta\binom{N}{j}\epsilon^j(1-\epsilon)^{N-j}$, for the constant factor $\theta = \frac{1}{\sum_{i=0}^{N-R}\binom{N}{i}\epsilon^i(1-\epsilon)^{N-i}}$. According to Theorem 6, R_1 can be set as N as long as $\epsilon < 7.4\%$.

V. Conclusion

In this paper, we consider coded distributed matrix multiplication. A flexible construction for distributed matrix multiplication is proposed and the optimal parameters are discussed. The construction can also be generalized to batch processing of matrix multiplication and secure distributed computation. Flexible constructions are also found in other problems such as communication-efficient secret sharing [58], adaptive gradient codes [59], coded elastic computing [60] and flexible storage [61], [62]. It is worthwhile to explore more applications of flexible constructions, such as distributed machine learning and secure multi-party computation.

References

[1] W. Li, Z. Chen, Z. Wang, S. A. Jafar, and H. Jafarkhani, "Flexible constructions for distributed matrix multiplication," in *Proceedings of International Symposium on Information Theory (ISIT)*, 2021.

[2] J. Dean and L. A. Barroso, "The tail at scale," *Communications of the ACM*, vol. 56, pp. 74–80, 2013.

[3] G. Ananthanarayanan, S. Kandula, A. Greenberg, I. Stoica, Y. Lu, B. Saha, and E. Harris, "Reining in the outliers in map-reduce clusters using mantri," in *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation*, ser. OSDI'10. USA: USENIX Association, 2010, p. 265–278.

[4] K. Lee, M. Lam, R. Pedarsani, D. Papailiopoulos, and K. Ramchandran, "Speeding up distributed machine learning using codes," *IEEE Transactions on Information Theory*, vol. 64, no. 3, pp. 1514–1529, 2017.

[5] Q. Yu, M. Maddah-Ali, and S. Avestimehr, "Polynomial codes: an optimal design for high-dimensional coded matrix multiplication," *Advances in Neural Information Processing Systems*, vol. 30, 2017.

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

[7] S. Dutta, Z. Bai, H. Jeong, T. M. Low, and P. Grover, "A unified coded deep neural network training strategy based on generalized PolyDot codes," in *2018 IEEE International Symposium on Information Theory (ISIT)*. IEEE, 2018, pp. 1585–1589.

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

[9] 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 *The 22nd International Conference on Artificial Intelligence and Statistics*. PMLR, 2019, pp. 1215–1225.

[10] A. Reisizadeh, S. Prakash, R. Pedarsani, and A. S. Avestimehr, "Coded computation over heterogeneous clusters," *IEEE Transactions on Information Theory*, vol. 65, no. 7, pp. 4227–4242, 2019.

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

[12] S. Dutta, V. Cadambe, and P. Grover, "Short-dot: Computing large linear transforms distributedly using coded short dot products," in *Advances In Neural Information Processing Systems*, 2016, pp. 2100–2108.

[13] ——, "Coded convolution for parallel and distributed computing within a deadline," in *2017 IEEE International Symposium on Information Theory (ISIT)*. IEEE, 2017, pp. 2403–2407.

[14] Q. Yu, M. A. Maddah-Ali, and A. S. Avestimehr, "Coded Fourier transform," in *2017 55th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*. IEEE, 2017, pp. 494–501.

[15] T. Jahani-Nezhad and M. A. Maddah-Ali, "Codedsketch: A coding scheme for distributed computation of approximated matrix multiplication," *IEEE Transactions on Information Theory*, vol. 67, no. 6, pp. 4185–4196, 2021.

[16] T. Baharav, K. Lee, O. Ocal, and K. Ramchandran, "Straggler-proofing massive-scale distributed matrix multiplication with d-dimensional product codes," in *2018 IEEE International Symposium on Information Theory (ISIT)*. IEEE, 2018, pp. 1993–1997.

[17] G. Suh, K. Lee, and C. Suh, "Matrix sparsification for coded matrix multiplication," in *2017 55th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*. IEEE, 2017, pp. 1271–1278.

[18] S. Wang, J. Liu, N. Shroff, and P. Yang, "Fundamental limits of coded linear transform," *arXiv preprint arXiv:1804.09791*, 2018.

[19] A. Mallick, M. Chaudhari, U. Sheth, G. Palanikumar, and G. Joshi, "Rateless codes for near-perfect load balancing in distributed matrix-vector multiplication," *Proc. ACM Meas. Anal. Comput. Syst.*, vol. 3, no. 3, 2019.

[20] S. Wang, J. Liu, and N. Shroff, "Coded sparse matrix multiplication," in *International Conference on Machine Learning*. PMLR, 2018, pp. 5152–5160.

[21] A. Severinson, A. G. i Amat, and E. Rosnes, "Block-diagonal and lt codes for distributed computing with straggling servers," *IEEE Transactions on Communications*, vol. 67, no. 3, pp. 1739–1753, 2018.

[22] F. Haddadpour and V. R. Cadambe, "Codes for distributed finite alphabet matrix-vector multiplication," in *2018 IEEE International Symposium on Information Theory (ISIT)*. IEEE, 2018, pp. 1625–1629.

[23] U. Sheth, S. Dutta, M. Chaudhari, H. Jeong, Y. Yang, J. Kohonen, T. Roos, and P. Grover, "An application of storage-optimal MatDot codes for coded matrix multiplication: Fast k-nearest neighbors estimation," in *2018 IEEE International Conference on Big Data (Big Data)*. IEEE, 2018, pp. 1113–1120.

[24] H. Jeong, F. Ye, and P. Grover, "Locally recoverable coded matrix multiplication," in *2018 56th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*. IEEE, 2018, pp. 715–722.

[25] M. Kim, J.-y. Sohn, and J. Moon, "Coded matrix multiplication on a group-based model," in *2019 IEEE International Symposium on Information Theory (ISIT)*. IEEE, 2019, pp. 722–726.

[26] H. Park, K. Lee, J.-y. Sohn, C. Suh, and J. Moon, "Hierarchical coding for distributed computing," in *2018 IEEE International Symposium on Information Theory (ISIT)*. IEEE, 2018, pp. 1630–1634.

[27] S. Li, M. A. Maddah-Ali, and A. S. Avestimehr, "Coding for distributed fog computing," *IEEE Communications Magazine*, vol. 55, no. 4, pp. 34–40, 2017.

[28] W. Chang and R. Tandon, "On the capacity of secure distributed matrix multiplication," *IEEE Global Communications Conference (GLOBECOM)*, pp. 1–6, 2018.

[29] J. Kakar, S. Ebadifar, and A. Sezgin, "On the capacity and straggler-robustness of distributed secure matrix multiplication," *IEEE Access*, vol. 7, pp. 45 783–45 799, 2019.

[30] R. G. D'Oliveira, S. E. Rouayheb, and D. Karpuk, "Gasp codes for secure distributed matrix multiplication," *IEEE Transactions on Information Theory*, 2020.

[31] M. Kim and J. Lee, "Private secure coded computation," *IEEE Communications Letters*, vol. 23, no. 11, pp. 1918–1921, 2019.

[32] M. Aliasgari, O. Simeone, and J. Kliewer, "Distributed and private coded matrix computation with flexible communication load," in *2019 IEEE International Symposium on Information Theory (ISIT)*. IEEE, 2019, pp. 1092–1096.

[33] Z. Jia and S. A. Jafar, "Cross subspace alignment codes for coded distributed batch computation," *IEEE Transactions on Information Theory*, vol. 67, no. 5, pp. 2821–2846, 2021.

[34] Z. Chen, Z. Jia, Z. Wang, and S. A. Jafar, "Gcsa codes with noise alignment for secure coded multi-party batch matrix multiplication," *IEEE Journal on Selected Areas in Information Theory, Early Access*, 2021.

[35] A. K. Pradhan, A. Heidarzadeh, and K. R. Narayanan, "Factored LT and factored Raptor codes for large-scale distributed matrix multiplication," *IEEE Journal on Selected Areas in Information Theory*, vol. 2, no. 3, pp. 893–906, 2021.

[36] N. Ferdinand and S. C. Draper, "Hierarchical coded computations," *IEEE International Symposium on Information Theory*, 2018.

[37] M. M. Amiri and D. Gündüz, "Computation scheduling for distributed machine learning with straggling workers," *IEEE Transactions on Signal Processing*, vol. 67, no. 24, pp. 6270–6284, 2019.

[38] R. Bitar, P. Parag, and S. E. Rouayheb, "Minimizing latency for secure coded computing using secret sharing via staircase codes," *IEEE Transactions on Communications*, vol. 68, no. 8, pp. 4609–4619, 2020.

[39] R. Bitar, Y. Xing, Y. Keshtkarjahromi, V. Dasari, S. El Rouayheb, and H. Seferoglu, "Private and rateless adaptive coded matrix-vector multiplication," *EURASIP Journal on Wireless Communications and Networking*, vol. 2021, no. 1, pp. 1–25, 2021.

[40] A. Ramamoorthy, L. Tang, and P. O. Vontobel, "Universally decodable matrices for distributed matrix-vector multiplication," in *2019 IEEE International Symposium on Information Theory (ISIT)*. IEEE, 2019, pp. 1777–1781.

[41] A. B. Das, L. Tang, and A. Ramamoorthy, "C3les: Codes for coded computation that leverage stragglers," in *2018 IEEE Information Theory Workshop (ITW)*, 2018, pp. 1–5.

[42] R. Bitar, M. Xhemrishi, and A. Wachter-Zeh, "Adaptive private distributed matrix multiplication," *IEEE Transactions on Information Theory*, 2022.

[43] S. Kiani, N. Ferdinand, and S. C. Draper, "Exploitation of stragglers in coded computation," *IEEE International Symposium on Information Theory*, 2018.

[44] B. Hasırcıoglu, J. Gómez-Vilardebó, and D. Gündüz, "Bivariate polynomial coding for exploiting stragglers in heterogeneous coded computing systems," *ArXiv:2001.07227*, 2020.

[45] B. Hasırcıoğlu, J. Gómez-Vilardebó, and D. Gündüz, "Bivariate Hermitian polynomial coding for efficient distributed matrix multiplicationn," *2020 IEEE Global Communications Conference*, pp. 1–6, 2020.

[46] X. Fan, P. Soto, X. Zhong, D. Xi, Y. Wang, and J. Li, "Leveraging stragglers in coded computing with heterogeneous servers," in *2020 IEEE/ACM 28th International Symposium on Quality of Service (IWQoS)*, 2020, pp. 1–10.

[47] A. B. Das and A. Ramamoorthy, "Coded sparse matrix computation schemes that leverage partial stragglers," in *2021 IEEE International Symposium on Information Theory (ISIT)*. IEEE, 2021, pp. 1570–1575.

[48] S. Kianidehkordi, N. Ferdinand, and S. C. Draper, "Hierarchical coded matrix multiplication," *IEEE Transactions on Information Theory*, vol. 67, no. 2, pp. 726–754, 2021.

[49] A. Ramamoorthy and L. Tang, "Numerically stable coded matrix computations via circulant and rotation matrix embeddings," in *2021 IEEE International Symposium on Information Theory (ISIT)*, 2021, pp. 1712–1717.

[50] W. Li, Z. Wang, and H. Jafarkhani, "On the sub-packetization size and the repair bandwidth of reed-solomon codes," *IEEE Transactions on Information Theory*, vol. 65, no. 9, pp. 5484–5502, 2019.

[51] K. M. Greenan, E. L. Miller, and T. J. S. SJ, "Optimizing galois field arithmetic for diverse processor architectures and applications," in *2008 IEEE International Symposium on Modeling, Analysis and Simulation of Computers and Telecommunication Systems*. IEEE, 2008, pp. 1–10.

[52] J. Luo, K. D. Bowers, A. Oprea, and L. Xu, "Efficient software implementations of large finite fields gf (2 n) for secure storage applications," *ACM Transactions on Storage (TOS)*, vol. 8, no. 1, pp. 1–27, 2012.

[53] J. S. Plank, K. M. Greenan, and E. L. Miller, "Screaming fast galois field arithmetic using intel simd instructions." in *FAST*, 2013, pp. 299–306.

[54] S. B. Gashkov and I. S. Sergeev, "Complexity of computation in finite fields," *Journal of Mathematical Sciences*, vol. 191, no. 5, pp. 661–685, 2013.

[55] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[56] M. Bläser, *Fast Matrix Multiplication*, ser. Graduate Surveys. Theory of Computing Library, 2013, no. 5.

[57] M. Sathiamoorthy, M. Asteris, D. Papailiopoulos, A. G. Dimakis, R. Vadali, S. Chen, and D. Borthakur, "Xoring elephants: Novel erasure codes for big data," in *39th Int. Conf. Very Large Data Bases*, vol. 6, no. 5, 2013, pp. 325–336.

[58] W. Huang, M. Langberg, J. Kliewer, and J. Bruck, "Communication efficient secret sharing," *IEEE Transactions on Information Theory*, vol. 62, no. 12, pp. 7195–7206, 2016.

[59] Y. Malitsky and K. Mishchenko, "Adaptive gradient descent without descent," *arXiv preprint arXiv:1910.09529*, 2019.

[60] Y. Yang, M. Interlandi, P. Grover, S. Kar, S. Amizadeh, and M. Weimer, "Coded elastic computing," in *2019 IEEE International Symposium on Information Theory (ISIT)*. IEEE, 2019, pp. 2654–2658.

[61] H. Jafarkhani and M. Hajiaghayi, "Cost-efficient repair for storage systems using progressive engagement," US Patent 10,187,088., Jan. 2019.

[62] W. Li, Z. Wang, T. Lu, and H. Jafarkhani, "Storage codes with flexible number of nodes," *arXiv preprint arXiv:2106.11336*, 2021.