Flexible Constructions for Distributed Matrix Multiplication

Weiqi Li, Zhen Chen, Zhiying Wang, Syed A. Jafar, Hamid Jafarkhani
Center for Pervasive Communications and Computing (CPCC)
University of California, Irvine, USA
$\{$weiqil4, zhenc4, zhiying, syed, hamidj$\}$@uci.edu

***Abstract*—The distributed matrix multiplication problem with unknown number of stragglers is considered, where the goal is to allow a master to efficiently and flexibly obtain the product of two massive matrices by distributing the computation across N servers. We assume there are at most $N - R$ stragglers but the exact number is not known a priori. Motivated by reducing the latency, a flexible solution is proposed to fully utilize the computation capability of available servers. The computing job for each server is separated into 2 layers, constructed based on Entangled Polynomial (EP) codes by Yu el al. The final results can be obtained when a larger number of servers complete the task from the first layer or a smaller number of servers complete the tasks from both 2 layers. The required finite field size of the proposed solution is less than $2N$. Moreover, the optimal partitioning of the input matrices is discussed. Our constructions can also be generalized to batch matrix multiplication.**

I. Introduction

Distributed matrix multiplication has received wide interest because of the huge amount of data computing required by many popular applications like machine learning. In particular, the following basic distributed matrix multiplication is considered: A master wishes to obtain the product 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. 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 master. After collecting enough results, the master can decode the desired product AB. To reduce the overall system latency caused by stragglers (servers that fail to respond or respond after the master executes the reconstruction), distributed matrix computing schemes with straggler tolerance are provided in [1]–[34]. Among the state-of-the-art schemes, some are based on matrix partitioning such as Polynomial codes [2], MatDot codes and PolyDot codes [3], Generalized PolyDot codes [4] and Entangled Polynomial (EP) codes [5], and others are based on batch processing such as Lagrange Coded Computing [6] and Cross Subspace Alignment codes [30]. The majority of the literature assumes a fixed number of stragglers, i.e., the data is distributed to N servers and after any R of them complete their computing, the final product can be obtained by the master. Here R is predetermined and called the *recovery threshold.* However, when the number of stragglers is smaller than $N - R$, the master still only uses the results from R servers, and the results of other servers are wasted. In [32]–[40], the authors consider a setting in which the number of stragglers is not known as a priori and design schemes that can cope with this setting. References [35], [39], [40] focus on the task scheduling for general distributed computing or distributed learning. The matrix-vector multiplication setting is considered in [32], [33]. Reference [34], [36]–[38] consider matrix-matrix multiplication, but they can only handle a special partitioning, i.e., A is split row-wisely and B is split column-wisely. 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 at each server and the amount of communication from servers to the master). When the number of stragglers is fixed, EP codes [5] provide an elegant solution for arbitrary partitioning by encoding the input matrix blocks into a carefully designed polynomial.

This paper proposes flexible distributed matrix multiplication in order to achieve low latency. The desired product AB can be decoded from collecting the results of a flexible number of servers. As long as the master collects enough results from servers, the computing is completed. This idea of multi-message is also considered in [37], [39], [40]. A naive solution to achieve flexibility is simply applying the EP code [5] with a recovery threshold of RK, where each server gets K pairs of shares instead of one pair of shares. The master can calculate the final results with any RN out of the KN computing results. Thus, each server only needs to compute RK/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 computation needs to be done in a field with minimum size of KN, and multiplication in a larger field results in a much bigger delay for each multiplication [41].

To obtain a smaller field size, we propose the following solution. The main idea is that non-stragglers can finish more tasks to compensate for the effect of the stragglers without knowing the pattern of the stragglers a priori. Specifically, the computation is divided into 2 layers, where the first layer has a larger recovery threshold and the second layer has a smaller recovery threshold. Each server keeps calculating and sending results to the master until enough servers send results to the master, which can be either a larger number of servers for the first layer or a smaller number of servers for both 2 layers. The remaining servers are viewed as stragglers. Our construction only requires a field size of less than $2N$. The computation load of each server can be reduced when there are fewer stragglers than $N-R$. Since computation load is one of the main reasons of delay, our scheme performs better than fixed EP codes with respect to delay, as shown in Fig. 1.

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 cardinality and when M is partitioned into blocks, $M_{(i,j)}$ denotes the block in the i-th row and the j-th column.

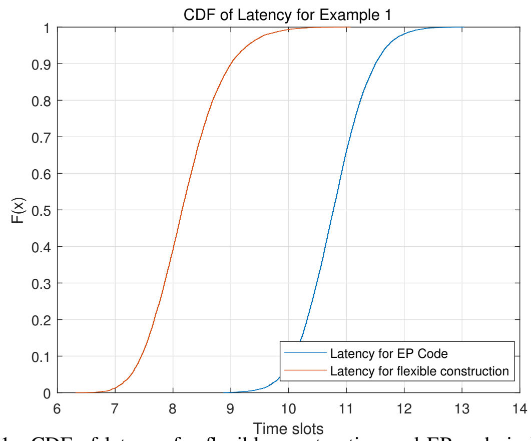

Fig. 1. CDF of latency for flexible construction and EP code in Example 1 of Section III. $N=R_1=5, R_2=R=3$. We assume $\lambda=\kappa=\mu=6U$, for some integer U, and the computation delay for multiplication of two $U\times U$ matrices in each server satisfy the exponential distribution with parameter 0.1. 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 of 3 servers complete 3 tasks (2 stragglers). The overall latency is the smallest latency of these 3 cases. The expected latency is 10.79 for EP code, and 8.20 for the flexible construction, hence we save 24%.

II. Problem Statement

We consider a problem of matrix multiplication 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 $S=AB$ in a distributed computing environment with 2 sources, a master, 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*. Given the flexibility parameters R_1, R_2, where $N\geq R_1>R_2=R$, 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 has R_1-R_2+1 coded matrices, which are divided into 2 layers. The first layer contains the first coded matrix, denoted by $\widetilde{A}_{i,1}$ or $\widetilde{B}_{i,1}$, and the second layer contains the remaining R_1-R_2 coded matrices, denoted by $\{\widetilde{A}_{i,2},\cdots,\widetilde{A}_{i,R_1-R_2+1}\}$, or $\{\widetilde{B}_{i,2},\cdots,\widetilde{B}_{i,R_1-R_2+1}\}$. For $i\in[N]$, the shares and the encoding functions are

$$\widetilde{\mathcal{A}}_i=\{\widetilde{A}_{i,j}\mid j\in[R_1-R_2+1]\}=f_i(A), \tag{1}$$
$$\widetilde{\mathcal{B}}_i=\{\widetilde{B}_{i,j}\mid j\in[R_1-R_2+1]\}=g_i(B). \tag{2}$$

Then $\widetilde{\mathcal{A}}_i$ and $\widetilde{\mathcal{B}}_i$ are sent to Server i from the sources before the computation starts. Each server is with a storage capacity C [1]. To satisfy the storage constraint, for each Server $i, i\in[N]$, $\sum_{M\in\widetilde{\mathcal{A}}_i\cup\widetilde{\mathcal{B}}_i}|M|\leq C$.

Server i computes R_1-R_2+1 tasks in order:

$$\widetilde{S}_{i,j}=h\left(\widetilde{A}_{i,j},\widetilde{B}_{i,j}\right)=\widetilde{A}_{i,j}\cdot\widetilde{B}_{i,j}, j\in[R_1-R_2+1],$$

and sends $\widetilde{S}_{i,j}$ to the master once its computation is finished. Since the results are computed in order, the master receives $\widetilde{S}_{i,j_1}$ before $\widetilde{S}_{i,j_2}$ for $\forall i\in[N], j_1<j_2$. Denote $\widetilde{S}_{i,[j]}=\left\{\widetilde{S}_{i,t}\mid t\in[j]\right\}$ and $\widetilde{S}_{\mathcal{K},[j]}=\left\{\widetilde{S}_{i,[j]}\mid i\in\mathcal{K}\right\}, \forall\mathcal{K}\subset[N]$.

The decoding function $d_{\mathcal{K},[j]}$ of the master for recovering S satisfies

$$\begin{aligned}&S=d_{\mathcal{K},[j]}\left(\widetilde{S}_{\mathcal{K},[j]}\right),\\&\forall R_2\leq|\mathcal{K}|=R^*\leq R_1, j=R_1-R^*+1.\end{aligned} \tag{3}$$

The function set $\{f_i,g_i,h,d_{\mathcal{K},[j]}\mid 1\leq i\leq N, R_2\leq|\mathcal{K}|=R^*\leq R_1, j=R_1-R^*+1\}$ is called the *flexible constructions for distributed matrix multiplication*.

In other words, the sources send all R_1-R_2+1 coded matrices to each server. Then, each server keeps calculating and sending results to the master until the master obtains enough results – either when the quickest R_1 servers complete the first task, or when the quickest R^* servers complete the first R_1-R^*+1 tasks, $R_2\leq R^*<R_1$. The remaining servers are viewed as stragglers. The *latency* is defined as the time required for the master to collect enough results from the start of the computation. For simplicity, in the analysis of this paper, we assume a small failure probability at each server and a constant time for a unit computation at each server if it is not a straggler.

We want to find flexible constructions with the *storage capacity* C and the *computation load* (i.e., the number of multiplications) at each server as small as possible.

III. Construction

In this section, we present our flexible constructions. We start from a motivating example.

Example 1. Consider the matrix multiplication of A and B, for $A\in\mathbb{F}^{\lambda\times\kappa}, B\in\mathbb{F}^{\kappa\times\mu}$, using $N=5$ servers with at most $N-R=2$ stragglers. Assume A is partitioned column-wisely and B is partitioned row-wisely: $A=[A_1,A_2], B=\begin{bmatrix}B_1\\B_2\end{bmatrix}$, and the master requires $AB=A_1B_1+A_2B_2$.

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

Applying the EP code [5], server $i, i \in [5]$ receives coded matrices $A_1 + \alpha_i A_2$ and $\alpha_i B_1 + B_2$, and calculates

$$\begin{aligned}&(A_1 + \alpha_i A_2) \cdot (\alpha_i B_1 + B_2) \\ =&A_1B_2 + \alpha_i(A_1B_1 + A_2B_2) + \alpha_i^2 A_2B_1,\end{aligned} \tag{4}$$

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 Lagrange interpolation. The total computation load of directly multiplying A and B is $L = \lambda\kappa\mu$, and with 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 R^* available servers can complete the computation, $3 = R_2 \leq R^* \leq R_1 = 5$. First, we partition the matrices and get $A = [A_1, A_2, A_3], B = [B_1^T, B_2^T, B_3^T]^T$, and thus the master 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

$$\begin{aligned}&(A_1 + \alpha_i A_2 + \alpha_i^2 A_3) \cdot (\alpha_i^2 B_1 + \alpha_i B_2 + B_3) \\ =&A_1B_3 + \alpha_i(A_2B_3 + A_1B_2) \\ &+\alpha_i^2(A_1B_1 + A_2B_2 + A_3B_3) \\ &+\alpha_i^3(A_2B_1 + A_3B_2) + \alpha_i^4 A_3B_1.\end{aligned} \tag{5}$$

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, B are divided into smaller pieces compared to 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 $A_{\alpha_i} = (A_1 + \alpha_i A_2 + \alpha_i^2 A_3), B_{\alpha_i} = (\alpha_i^2 B_1 + \alpha_i B_2 + B_3)$ and we further partition them into 2 parts,

$$A_{\alpha_i} = [A_{\alpha_i,1}, A_{\alpha_i,2}], B_{\alpha_i} = \begin{bmatrix} B_{\alpha_i,1} \\ B_{\alpha_i,2} \end{bmatrix}. \tag{6}$$

The calculation of each server is shown in Table I.

Since Layer 2 has a similar structure as (4), 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 master 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 master 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.

Note that in this example, there are $R_1 - R_2 + 1 = 3$ coded matrices in a share. That is, $\widetilde{A}_{i,1} = A_{\alpha_i}, \widetilde{A}_{i,2} = A_{\alpha_6,1} + \alpha_i A_{\alpha_6,2}, \widetilde{A}_{i,3} = A_{\alpha_7,1} + \alpha_i A_{\alpha_7,2}, \widetilde{B}_{i,1} = B_{\alpha_i}, \widetilde{B}_{i,2} = B_{\alpha_6,1} + \alpha_i B_{\alpha_6,2}, \widetilde{B}_{i,3} = B_{\alpha_7,1} + \alpha_i B_{\alpha_7,2}$, for $i \in [N]$. Server i needs to store $\widetilde{\mathcal{A}}_i$ and $\widetilde{\mathcal{B}}_i$ before the computation steps. Each server computes the $R_1 - R_2 + 1 = 3$ tasks in order independent of the progress of the other servers.

From Example 1, when there is no straggler (which is more likely in most practical systems), we can reduce the computation load of each server from $L/2$ to $L/3$. In the worst case, we can tolerate 2 stragglers and get the desired results. The resulting latency under an exponential model is plotted in Fig. 1.

In this example, 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 a good performance on storage size in Section IV.

Next, we present the general construction of our flexible schemes.

Construction 1. Assume we have $N \geq R_1 > R_2 = R$, $R_j = p_j m_j n_j + p_j - 1, j \in [2]$, and distinct elements $\{\alpha_i \mid i \in [N + R_1 - R_2]\}$ from the finite field $\mathbb{F}$. With p_1, m_1, n_1, matrices A, B are partitioned as

$$\begin{bmatrix} A_{(1,1)} & \cdots & A_{(1,p_1)} \\ A_{(2,1)} & \cdots & A_{(2,p_1)} \\ \vdots & \vdots & \vdots \\ A_{(m_1,1)} & \cdots & A_{(m_1,p_1)} \end{bmatrix}, \begin{bmatrix} B_{(1,1)} & \cdots & B_{(1,n_1)} \\ B_{(2,1)} & \cdots & B_{(2,n_1)} \\ \vdots & \vdots & \vdots \\ B_{(p_1,1)} & \cdots & B_{(p_1,n_1)} \end{bmatrix}. \tag{7}$$

In Layer 1, set $A^{(1)} = A, B^{(1)} = B$, we calculate $f_{1,A^{(1)}}(\alpha_i) \cdot f_{1,B^{(1)}}(\alpha_i)$ in server i, where

$$f_{1,A^{(1)}}(\alpha_i) = \sum_{u=1}^{m_1}\sum_{v=1}^{p_1} A^{(1)}_{(u,v)} \alpha_i^{v-1+p_1(u-1)}, \tag{8}$$

$$f_{1,B^{(1)}}(\alpha_i) = \sum_{u=1}^{p_1}\sum_{v=1}^{n_1} B^{(1)}_{(u,v)} \alpha_i^{p_1-u+p_1m_1(v-1)}, \tag{9}$$

are shares based on EP codes [5]. Here, for Server i, $\widetilde{A}_{i,1} = f_{1,A^{(1)}}(\alpha_i)$, $\widetilde{B}_{i,1} = f_{1,B^{(1)}}(\alpha_i)$.

In Layer 2, we partition matrices $f_{1,A^{(1)}}(\alpha_{N+t})$, $f_{1,B^{(1)}}(\alpha_{N+t})$, $t \in [R_1 - R_2]$, with parameters p_2, m_2, n_2. Server i calculates $f_{2,A^{(2)}}(\alpha_i) \cdot f_{2,B^{(2)}}(\alpha_i)$, where $\left(A^{(2)}, B^{(2)}\right) \in \{\left(f_{1,A^{(1)}}(\alpha_{N+t}), f_{1,B^{(1)}}(\alpha_{N+t}\right) \mid t \in [R_1 - R_2]\}$ and

$$f_{2,A^{(2)}}(\alpha_i) = \sum_{u=1}^{m_2}\sum_{v=1}^{p_2} A^{(2)}_{(u,v)} \alpha_i^{v-1+p_2(u-1)}, \tag{10}$$

$$f_{2,B^{(2)}}(\alpha_i) = \sum_{u=1}^{p_2}\sum_{v=1}^{n_2} B^{(2)}_{(u,v)} \alpha_i^{p_2-u+p_2m_2(v-1)}. \tag{11}$$

In Layer 2, for Server $i \in [N]$, index $t \in [R_1 - R_2]$, $A^{(2)} = f_{1,A^{(1)}}(\alpha_{N+t})$, and $B^{(2)} = f_{1,B^{(1)}}(\alpha_{N+t})$, the corresponding coded matrices are

$$\begin{aligned}\widetilde{A}_{i,t+1} &= f_{2,A^{(2)}}(\alpha_i), \\ \widetilde{B}_{i,t+1} &= f_{2,B^{(2)}}(\alpha_i).\end{aligned}$$

The calculation tasks in both layers are shown in Table II. Since we only use $N + R_1 - R_2$ distinct α_i values, the required field size is $|\mathbb{F}| \geq N + R_1 - R_2$.

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

TABLE II
CALCULATION TASKS IN EACH SERVER FOR THE GENERAL CONSTRUCTION. IN LAYER 1, $A^{(1)} = A, B^{(1)} = B$. LAYER 2 CALCULATES FOR ALL PAIRS OF $(A^{(2)}, B^{(2)}) \in \{\big(f_{1,A^{(1)}}(\alpha_{N+t}), f_{1,B^{(1)}}(\alpha_{N+t}\big) \mid t \in [R_1 - R_2]\}$.

	Server 1	Server 2	...	Server N
Layer 1	$f_{1,A^{(1)}}(\alpha_1) \cdot f_{1,B^{(1)}}(\alpha_1)$	$f_{1,A^{(1)}}(\alpha_2) \cdot f_{1,B^{(1)}}(\alpha_2)$	...	$f_{1,A^{(1)}}(\alpha_N) \cdot f_{1,B^{(1)}}(\alpha_N)$
Layer 2	$f_{2,A^{(2)}}(\alpha_1) \cdot f_{2,B^{(2)}}(\alpha_1)$	$f_{2,A^{(2)}}(\alpha_2) \cdot f_{2,B^{(2)}}(\alpha_2)$	...	$f_{2,A^{(2)}}(\alpha_N) \cdot f_{2,B^{(2)}}(\alpha_N)$

Theorem 1. In Construction 1, assume we have R^* available servers and $R_2 \le R^* \le N$, we only need

$$L_{\text{flex}} = \begin{cases} \frac{\lambda\kappa\mu}{m_1 p_1 n_1}, & R^* \ge R_1, \\ \frac{\lambda\kappa\mu}{m_1 p_1 n_1} + \frac{\lambda\kappa\mu(R_1 - R^*)}{m_1 m_2 p_1 p_2 n_1 n_2}, & R_2 \le R^* < R_1. \end{cases} \tag{12}$$

computation load in each server to obtain the final product, and the storage capacity required is

$$C_{\text{flex}} = \frac{1}{p_1}\left(\frac{\lambda\kappa}{m_1} + \frac{\kappa\mu}{n_1}\right) + \frac{R_1 - R_2}{p_1 p_2}\left(\frac{\lambda\kappa}{m_1 m_2} + \frac{\kappa\mu}{n_1 n_2}\right). \tag{13}$$

Proof: We first look at the computation load.

In the case that the number of available servers $R^* \ge R_1$, according to the correctness of EP codes [5], the required results $A \times B$ can be obtained by collecting R_1 evaluation points of $f_{1,A^{(1)}}(\alpha_i) \times f_{1,B^{(1)}}(\alpha_i)$. Thus, we only need the computation in Layer 1. In Layer 1, we calculate $f_{1,A^{(1)}}(\alpha_i) \cdot f_{1,B^{(1)}}(\alpha_i)$. From (7), (8) and (9) we know that $f_{1,A^{(1)}}(\alpha_i)$ has size $\frac{\lambda}{m_1} \cdot \frac{\kappa}{p_1}$ and $f_{1,B^{(1)}}(\alpha_i)$ has size $\frac{\kappa}{p_1} \cdot \frac{\mu}{n_1}$. Thus, normalized by the cost of a single multiplication operation, the computation in Layer 1 is

$$L_1 = \frac{\lambda\kappa\mu}{m_1 p_1 n_1}. \tag{14}$$

When $R_2 \le R^* < R_1$, we only have R^* evaluation points of $f_{1,A^{(1)}}(\alpha_i) \cdot f_{1,B^{(1)}}(\alpha_i)$ calculated in Layer 1. Then, we need to obtain additional $R_1 - R^*$ evaluation points. In Layer 2, $f_{2,A^{(2)}}(\alpha_i) \cdot f_{2,B^{(2)}}(\alpha_i)$, $i \in [N]$, are calculated at the servers with $(A^{(2)}, B^{(2)})$ chosen from $\{\big(f_{1,A^{(1)}}(\alpha_{N+t}), f_{1,B^{(1)}}(\alpha_{N+t})\big)\}, t \in [R_1 - R_2]\}$. With each pair of $(A^{(2)}, B^{(2)})$, the master can calculate one evaluation point of $f_{1,A^{(1)}}(\alpha_{N+t}) \cdot f_{1,B^{(1)}}(\alpha_{N+t})$ since (10) and (11) are exactly the EP code [5]. From (10) and (11), we know that $f_{2,A^{(2)}}(\alpha_i)$ has size $\frac{\lambda}{m_1 m_2} \cdot \frac{\kappa}{p_1 p_2}$ and $f_{2,B^{(2)}}(\alpha_i)$ has size $\frac{\kappa}{p_1 p_2} \cdot \frac{\mu}{n_1 n_2}$. Thus, the total computation in Layer 2 is

$$L_2 = \frac{(R_1 - R^*) L_1}{p_2 m_2 n_2}. \tag{15}$$

Combining (14) and (15), the computation load is given as (12).

For the storage, we first look at the storage size required for each layer. In Layer 1, we need to store $f_{1,A^{(1)}}(\alpha_i), f_{1,B^{(1)}}(\alpha_i)$, then

$$C_1 = \frac{1}{p_1}\left(\frac{\lambda\kappa}{m_1} + \frac{\kappa\mu}{n_1}\right). \tag{16}$$

In Layer 2, we need to store all pairs of $f_{2,A^{(2)}}(\alpha_i), f_{2,B^{(2)}}(\alpha_i)$, for $R_1 - R_2$ choices of $(A^{(2)}, B^{(2)})$. The required storage size is

$$C_2 = \frac{(R_1 - R_2)}{p_1 p_2}\left(\frac{\lambda\kappa}{m_1 m_2} + \frac{\kappa\mu}{n_1 n_2}\right). \tag{17}$$

Thus, we obtain the total required storage size as (13). ■

Remark. Cross Subspace Alignment codes and Generalized Cross Subspace Alignment codes [30] are designed to handle batch processing of matrix multiplication. Our construction can also be easily modified to handle batch processing based on these two codes.

IV. OPTIMIZATION

In this section, we discuss how to pick partitioning parameters p, m, n, to improve the system performance, i.e., to minimize the computation load given the storage constraint C in each server.

We first discuss fixed EP code with a fixed recovery threshold R, which satisfies $R = m_0 p_0 n_0 + p_0 - 1$ according to [5], for some undetermined p_0, m_0, n_0. The computation load and the storage size required are shown in [5] as

$$L_{\text{EP}} = \frac{\lambda\kappa\mu}{m_0 p_0 n_0}, \quad C_{\text{EP}} = \frac{1}{p_0}\left(\frac{\lambda\kappa}{m_0} + \frac{\kappa\mu}{n_0}\right). \tag{18}$$

Thus, the optimization problem can be formulated as

$$\begin{aligned} \min_{p_0, m_0, n_0} \quad & L_{\text{EP}} = \frac{\lambda\kappa\mu}{m_0 p_0 n_0}, \\ \text{s.t.} \quad & R = p_0 m_0 n_0 + p_0 - 1, \\ & \frac{\lambda\kappa}{p_0 m_0} + \frac{\kappa\mu}{p_0 n_0} \le C, \\ & p_0, m_0, n_0 \text{ are integers.} \end{aligned} \tag{19}$$

Theorem 2. The optimization in (19) without the integer constraint has solution

$$p_0^* = \frac{1}{2}(R+1) - \frac{1}{2}\sqrt{(R+1)^2 - 16\frac{\lambda\kappa^2\mu}{C^2}}, \tag{20}$$

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: Using the threshold constraint

$$p_0 m_0 n_0 = R + 1 - p_0, \tag{21}$$

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

$$\frac{(\lambda\kappa n_0 + \kappa\mu m_0)}{R+1-p_0} \leq C. \tag{22}$$

Also, we have

$$\lambda\kappa n_0 + \kappa\mu m_0 \geq 2\sqrt{\lambda\kappa^2\mu m_0 n_0} = 2\sqrt{\lambda\kappa^2\mu\left(\frac{R+1}{p_0} - 1\right)} \tag{23}$$

and it holds with equality if and only if $\lambda\kappa n_0 = \kappa\mu m_0$. Thus, we have (22) as

$$2\sqrt{\frac{\lambda\kappa^2\mu}{(R+1-p_0)p_0}} \leq C, \tag{24}$$

which decreases with p_0 since the derivative satisfies

$$\frac{\mathrm{d}\,(R+1-p_0)p_0}{\mathrm{d}\,p_0} = R+1-2p_0 = p_0 m_0 n_0 - p_0 \geq 0. \tag{25}$$

Thus, L_{EP} reaches its optimal value when (24) holds with equality and $\lambda\kappa n_0 = \kappa\mu m_0$. Combining (21), the optimal p_0^* is given by (20), and then m_0^*, n_0^* can be obtained accordingly. ■

Notice that p_0, m_0, n_0 are integers, we pick these 3 parameters close to the optimal values that satisfy all the constraints in (19).

Next, we consider the flexible constructions with predetermined $R_1, R_2 = R$. Assume that the probability that each server is a straggler is ϵ. The average computation load is

$$E[L_{\text{flex}}] = \sum_{R^*=R_1}^{N} \binom{N}{R^*}(1-\epsilon)^{R^*}\epsilon^{N-R^*}\frac{\lambda\kappa\mu}{p_1 m_1 n_1} + \sum_{R^*=R_2}^{R_1-1}\binom{N}{R^*}(1-\epsilon)^{R^*}\epsilon^{N-R^*}\frac{\lambda\kappa\mu(R_1-R^*)}{m_1 m_2 p_1 p_2 n_1 n_2}. \tag{26}$$

In practical systems, ϵ is small (e.g., less than 110 failures over 3000-node production clusters of Facebook per day [42]), so we ignore the second term in (26) and use the approximation $L_{\text{flex}} = \frac{\lambda\kappa\mu}{p_1 m_1 n_1}$ in our optimization problem. Combined with (13), the optimization problem can be formulated as

$$\begin{aligned} &\min_{p_1,m_1,n_1,p_2,m_2,n_2} \quad L_{\text{flex}} = \frac{\lambda\kappa\mu}{p_1 m_1 n_1}, \\ \text{s.t.} \quad & R_j = p_j m_j n_j + p_j - 1, j \in [2], \\ & \frac{1}{p_1}\left(\frac{\lambda\kappa}{m_1} + \frac{\kappa\mu}{n_1}\right) + \frac{(R_1-R_2)}{p_1 p_2}\left(\frac{\lambda\kappa}{m_1 m_2} + \frac{\kappa\mu}{n_1 n_2}\right) \leq C, \\ & p_1, m_1, n_1, p_2, m_2, n_2 \text{ are integers.} \end{aligned} \tag{27}$$

Theorem 3. The solution to (27) without the integer constraint for p_1, m_1, n_1, p_2 is

$$p_1^* = \frac{R_1+1}{2} - \sqrt{\frac{(R_1+1)^2}{4} - \frac{4\lambda\kappa^2\mu(2R_1-R_2+1)^2}{C^2(R_2+1)^2}}, \tag{28}$$

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

Using $m_j n_j = \frac{R_j+1}{p_j} - 1$, similar to (23), we have:

$$\frac{1}{p_1}\left(\frac{\lambda\kappa}{m_1} + \frac{\kappa\mu}{n_1}\right) \geq 2\sqrt{\frac{\lambda\kappa^2\mu}{(R_1+1-p_1)p_1}}, \tag{29}$$

$$\frac{(R_1-R_2)}{p_1 p_2}\left(\frac{\lambda\kappa}{m_1 m_2} + \frac{\kappa\mu}{n_1 n_2}\right) \geq 2(R_1-R_2)\sqrt{\frac{\lambda\kappa^2\mu}{(R_1+1-p_1)(R_2+1-p_2)p_1 p_2}}. \tag{30}$$

Similar to (25), we know that (30) is a decreasing function of p_1 and p_2. Thus, when p_2 reaches its maximum, p_1 is minimized. Noticing that $p_2 = \frac{R_2+1}{m_2 n_2 + 1}$ and m_2, n_2 are integers, we set $p_2^* = \frac{R_2+1}{2}, m_2^* = 1, n_2^* = 1$. The optimal p_1^* is obtained from (29) and (30). ■

Again, we pick $p_1, m_1, n_1, p_2, m_2, n_2$ as integers around the optimal value satisfying (27) as our final choice.

Example 2. Assume we have $N = 8$ 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 codes 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 cost 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.

V. Conclusion

In this paper, 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.

References

[1] 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.

[2] Q. Yu, M. A. Maddah-Ali, and A. S. Avestimehr, “Polynomial codes: an optimal design for high-dimensional coded matrix multiplication,” *arXiv preprint arXiv:1705.10464*, 2017.

[3] 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.

[4] S. Dutta, Z. Bai, H. Jeong, T. Low, and P. Grover, “A unified coded deep neural network training strategy based on generalized polydot codes for matrix multiplication,” *ArXiv:1811.10751*, Nov. 2018.

[5] 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.

[6] Q. Yu, S. Li, N. Raviv, S. M. M. Kalan, M. Soltanolkotabi, and S. Avestimehr, “Lagrange coded computing: Optimal design for resiliency, security and privacy,” *ArXiv:1806.00939*, 2018.

[7] 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.

[8] 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.

[9] 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.

[10] ——, “Coded convolution for parallel and distributed computing within a deadline,” *arXiv preprint arXiv:1705.03875*, 2017.

[11] Q. Yu, M. A. Maddah-Ali, and A. S. Avestimehr, “Coded fourier transform,” *arXiv preprint arXiv:1710.06471*, 2017.

[12] T. Jahani-Nezhad and M. A. Maddah-Ali, “Codedsketch: A coding scheme for distributed computation of approximated matrix multiplications,” *arXiv preprint arXiv:1812.10460*, 2018.

[13] 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.

[14] 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.

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

[16] 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.

[17] S. Wang, J. Liu, and N. Shroff, “Coded sparse matrix multiplication,” *arXiv preprint arXiv:1802.03430*, 2018.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

[22] M. Kim, J.-y. Sohn, and J. Moon, “Coded matrix multiplication on a group-based model,” *arXiv preprint arXiv:1901.05162*, 2019.

[23] H. Park, K. Lee, J.-y. Sohn, C. Suh, and J. Moon, “Hierarchical coding for distributed computing,” *arXiv preprint arXiv:1801.04686*, 2018.

[24] 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.

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

[26] 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.

[27] R. G. DOliveira, S. E. Rouayheb, and D. Karpuk, “Gasp codes for secure distributed matrix multiplication,” *IEEE Transactions on Information Theory*, 2020, early access, DOI: 10.1109/TIT.2020.2975021.

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

[29] M. Aliasgari, O. Simeone, and J. Kliewer, “Distributed and private coded matrix computation with flexible communication load,” *arXiv preprint arXiv:1901.07705*, 2019.

[30] Z. Jia and S. Jafar, “Cross-subspace alignment codes for coded distributed batch computation,” *ArXiv:1909.13873*, 2019.

[31] 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.

[32] 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.

[33] R. Bitar, Y. Xing, Y. Keshtkarjahromi, V. Dasari, S. E. Rouayheb, and H. Seferoglu, “Private and rateless adaptive coded matrix-vector multiplication,” *arXiv preprint arXiv:1909.12611*, 2019.

[34] R. Bitar, M. Xhemrishi, and A. Wachter-Zeh, “Adaptive private distributed matrix multiplication,” *arXiv preprint arXiv:2101.05681*, 2021.

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

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

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

[38] 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.

[39] 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.

[40] E. Ozfatura, S. Ulukus, and D. Gündüz, “Straggler-aware distributed learning: Communication computation latency trade-off,” *arXiv:2004.04948*, 2020.

[41] 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.

[42] 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.