Coded Computing at Full Speed

Bin Tang†, Jiannong Cao§, Runze Cui†, Baoliu Ye†, Ye Li‡, and Sanglu Lu†
† National Key Laboratory for Novel Software Technology, Nanjing University, Nanjing, China
§ Department of Computing, The Hong Kong Polytechnic University, Hong Kong
‡ School of Electronics and Information, Nantong University, Nantong, China
Email: {tb, yebl, sanglu}@nju.edu.cn, csjcao@comp.polyu.edu.hk, MF1833013@smail.nju.edu.cn, yeli@ntu.edu.cn

***Abstract*—Distributed computing is the mainstream for large-scale machine learning and big data analytics, but its performance usually suffers from unpredictable stragglers, i.e., very slow nodes. Recently, coded computing has emerged as a new distributed computing paradigm that uses coding-theoretical approaches to mitigate the effect of stragglers. Most existing coding schemes only use the results from a certain number of fastest worker nodes to recover the output and completely ignore the partial work done by other worker nodes, leading to inferior performance. In this paper, for scenarios where each worker node transmits its local result to the master node only after it has finished the whole local computation, we introduce *communication at full speed* to characterize the full utilization of all the communication links between each worker node that has finished its local computation and the master node, and for scenarios where each worker node computes out the local result piece by piece and can forward each piece once available, we introduce *computation at full speed* to characterize the full utilization of the work done entirely or partially by all the worker nodes. Considering a general polynomial-based coding framework which encapsulates many advanced coding schemes for a variety of fundamental computing tasks, we propose a randomized approach where each worker node partitions its local result into pieces, generates and forwards random linear combinations of these pieces to the master node sequentially, and theoretically demonstrate that it can lead the coding framework to achieve communication at full speed. For some typical task scenarios, we further show that computation at full speed can be achieved by mapping the encoding operations in the randomized approach into a part of encoding on the input dataset. Experiments conducted on Alibaba Cloud as well as simulations show that our approaches can reduce the total runtime significantly.**

***Index Terms*—distributed computing, coded computing, polynomial-based coding, straggler tolerance**

I. Introduction

A. Background

In recent years, distributed computing has become the mainstream for large-scale machine learning and big data analytics [1]–[3]. One major issue in distributed computing is the existence of stragglers, i.e., the worker nodes that are very slow (can be 5 to 8 times slower than the average [4]) due to several factors including varying network condition, resource contention, etc., which are very common and unpredictable in modern distributed computing systems [5], [6]. Because the master node has to wait for all the worker nodes to finish, the whole computation can be much prolonged by the stragglers.

Recently, coded computing has become a new distributed computing paradigm to deal with the effect of stragglers [7]. By using coding-theoretical approaches to introduce proper redundant subtasks, the whole computation can be finished without waiting for the results from stragglers. Since then, many coding schemes have been proposed for a variety of computing tasks that are fundamental for large-scale machine learning and big data analytics, including matrix-vector multiplication [7]–[13], high-dimensional matrix-matrix multiplication [11], [14]–[19], distributed gradient descent [20]–[26], convolutions [16], [27], and multivariate polynomial evaluation [28], [29], etc.

One theme in the design of coding schemes is to bring down the recovery threshold under certain resource constraints, which is the number of local results of worker nodes required by the master node for recovering the desired output in the worst case, such that higher tolerance of stragglers can be provided. In these designs, only the results from the fastest k worker nodes are used by the master node for decoding, where k is the recovery threshold, while the partial work done by other worker nodes is completely ignored, even if they are slightly slower than the fastest k ones, leading to inferior performance.

B. Related Work

There are some very recent works focusing on exploiting the work done by those worker nodes that are not the fastest, but only for some specific computing tasks.

For matrix-vector multiplication, both Das *et al.* [10] and Kiani *et al.* [11] considered a fine grained model and take the sequence order of local computations at each worker node into account for the design of codes. However, in these schemes, it is inevitable that some redundant partial results would be forwarded to the master before decoding, which are wasteful. Mallick *et al.* [12] considered the use of rateless codes [30] which can leverage all the work done by worker nodes with small overhead. The approach of Kiani *et al.* [11] was also applied to product code [14] based high-dimensional matrix-matrix multiplication, but it has similar drawback as in the matrix-vector multiplication scenario. Besides, product code itself is not optimal in terms of recovery threshold [15].

One important idea in exploiting the work done by the worker nodes that are not the fastest is that, when the number of stragglers is less than that a coding scheme can tolerate, there would be redundant results among these non-stragglers which can be leveraged to reduce the communication cost per non-straggler. Ye and Abbe [21] first implemented this idea

Corresponding author: Baoliu Ye

in distributed gradient descent, and proposed a new gradient coding scheme that can achieve the best tradeoff between straggler tolerance and communication cost per non-straggler. Raviv *et al.* [29] applied the same idea to the Lagrange coding based multivariate polynomial evaluation, and proposed a technique named ideal decoding to reduce the communication cost per non-straggler. However, both schemes assume a fixed number of non-stragglers, which are not adaptive since the number of non-stragglers during execution is hard to predict accurately, so they cannot fully exploit all the non-stragglers. Besides, both schemes only exploit the non-stragglers that have finished their local computations. Hence, in scenarios where a worker node computes out the local result piece by piece, the partial computation results of other worker nodes will be completely ignored.

C. Our Contribution

In this paper, we target at fully utilizing all the work done, possibly partial, by each worker node to optimize the performance of coded computing. Two typical modes of performing the distributed computation are considered. One is called *communication-after-computation mode*, or *after-mode* for short, where each worker node starts the transmission of its local result to the master node only after it has finished the whole local computation. The other is called *communication-while-computation mode*, or *while-mode* for short, where the worker node computes out the local result piece by piece and can forward each piece to the master node once the piece is available.

We first introduce two important concepts to characterize the full utilization of all the work done by each worker node. One is referred to as *communication at full speed*, which characterizes the full utilization of all the communication links between each worker node that has finished its local computation and the master node in the after-mode. The other is called *computation at full speed*, which characterizes the full utilization of the work done entirely or partially by all the worker nodes in the while-mode.

Instead of focusing on some specific computing task, we consider a general polynomial-based coding framework that encapsulates many advanced coding schemes for a variety of fundamental computing tasks, including Reed-Solomon codes for matrix-vector multiplication [7], Polynomial codes [15], MatDot codes [17] and PolyDot codes [17] for high-dimensional matrix-matrix multiplication, Lagrange coding for multivariate polynomial evaluation [28], and Reed-Solomon codes for distributed gradient descent [23]. All these mentioned coding schemes are optimal in terms of recovery threshold under some certain constraints.

For the after-mode, we propose a randomized approach that can be applied to any polynomial-based coding scheme, where the local result of each worker node is partitioned into pieces and the worker nodes forward random linear combinations of these pieces to the master node sequentially, instead of the original ones. By analyzing the corresponding coefficient matrix underlying our approach, which is random but exhibits some inner structure posed by the polynomial-based coding scheme, we theoretically demonstrate that our approach can lead the polynomial-based coding scheme to achieve communication at full speed with probability 1 if the computation is over the real field or close to 1 if the computation is over a large finite field. Also, the extra computation cost incurred by our randomized approach is usually negligible.

For the while-mode, we show that, in many polynomial-based coding schemes, we can further map the linear combination operations on the local results in the proposed randomized approach into a part of the encoding on the input dataset, which is preprocessed before the computation starts, such that the pieces forwarded to the master node are the same as the ones in the after-mode with the randomized approach. In this way, all the partial results of worker nodes can be fully utilized, achieving computation at full speed.

The main contribution of this paper is summarized as follows:

- We introduce communication at full speed and computation at full speed to characterize the full utilization of works done entirely or partially by every worker node in the after-mode and in the while-mode, respectively.
- For general polynomial-based coding schemes applied in the after-mode, we propose a randomized approach that can lead the coding schemes to achieve communication at full speed, while the incurred extra computation cost is usually negligible.
- For polynomial-based coding schemes for some typical task scenarios applied in the while-mode, we propose an improvement on the randomized approach such that computation at full speed is achieved.
- We evaluate our proposed approaches by conducting experiments on Alibaba Cloud [31] as well as some additional simulations. The results show that our approaches can reduce the total runtime significantly.

The rest of the paper is organized as follows. Sec. II presents the model of coded computing and the formalization of two key concepts. Sec. III introduces the general polynomial-based coding framework. In Sec. IV and Sec. V, we introduce approaches for achieving communication at full speed and computation at full speed, respectively. Sec. VI is dedicated to the proof of our main theorem. Sec. VII presents the experimental results. Finally, Sec. VIII gives the concluding remarks.

II. Model and Objectives

In this section, we introduce the basic model for coded computing, and then formalize the two important concepts, communication at full speed and computation at full speed, towards fully utilizing the results, entirely or partially, at each worker node.

A. The Model of Coded Computing

For any positive integer j, let $[j]$ denote the set $\{1, 2, \ldots, j\}$. Let $\mathbb{F}$ be a field, that could be a finite field $\mathbb{F}_q$ with size q or the real field $\mathbb{R}$ that depends on the specific application scenario.

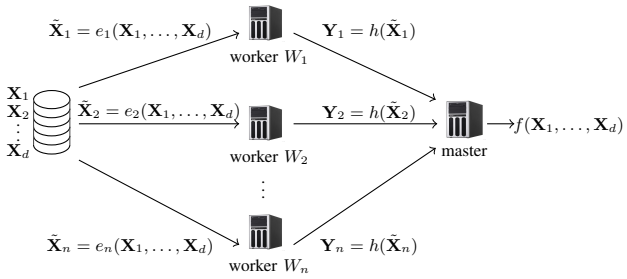


Fig. 1: An illustration of coded computing over a distributed master-worker system.

Given an input dataset $(\mathbf{X}_1, \mathbf{X}_2, \ldots, \mathbf{X}_d)$ where $\mathbf{X}_i \in \mathbb{F}^{v \times w}$, we consider the task of computing a function over the dataset, denoted by $f(\mathbf{X}_1, \mathbf{X}_2, \ldots, \mathbf{X}_d)$, in a distributed computing environment with a master node and n worker nodes. See Fig. 1 for an illustration.

Let $\mathcal{W} = \{W_1, W_2, \ldots, W_n\}$ denote the set of worker nodes. Prior to computation, each worker node W_i stores a possibly coded fraction of the dataset, denoted by $\tilde{\mathbf{X}}_i = e_i(\mathbf{X}_1, \mathbf{X}_2, \ldots, \mathbf{X}_d)$, where e_i is referred to as the *encoding function* of worker node W_i. Then worker node W_i computes a function of $\tilde{\mathbf{X}}_i$, denoted by $\mathbf{Y}_i = h(\tilde{\mathbf{X}}_i) \in \mathbb{F}^{r \times s}$, and returns $\mathbf{Y}_i$ to the master node. We refer to h as the *local computing* function and $\mathbf{Y}_i$ the local result of worker node W_i. After collecting a subset of (possibly partial) local results from these worker nodes, the master node finally decodes $f(\mathbf{X}_1, \mathbf{X}_2, \ldots, \mathbf{X}_d)$ by using some decoding algorithm $\mathcal{A}$. The tuple $(\{e_i\}_{i\in[n]}, h, \mathcal{A})$ is referred to as a coding scheme.

The use of coded computing provides the capability of tolerating *stragglers*, which are worker nodes that are much slower than other nodes in their computations or communications with the master node. One key concept about such capability of a coding scheme is as follows [15], [17], [24].

Definition 1 (Recovery Threshold)**.** The recovery threshold of a coding scheme $\Psi = (\{e_i\}_{i\in[n]}, h, \mathcal{A})$, denoted by $\mathsf{rec}(\Psi)$, is defined as the minimum number j such that the master can uniquely decode the desired result $f(\mathbf{X}_1, \mathbf{X}_2, \ldots, \mathbf{X}_d)$ from the local results of any j workers.

In other words, a coding scheme Ψ can tolerate any $n - \mathsf{rec}(\Psi)$ arbitrary stragglers.

B. Computation Modes

Note that the above coded computing model does not specify how the worker nodes perform local computation of function h and how the worker nodes send their local results to the master node. Regarding this, there are two typical modes as follows.

- *Communication after computation mode (after-node)*, where each worker node starts the transmission of its local result to the master node only after it has finished the whole local computation of $\mathbf{Y}_i = h(\tilde{\mathbf{X}}_i)$. This mode is very common, as in many cases the local result is computed out as a whole.
- *Communication while computation mode (while-mode)*, where the worker node computes out the local result piece by piece and can forward each piece to the master node once the piece is available.

Here gives an example to illustrate the two modes.

Example 1. *Consider that a worker node is to compute* $\mathbf{P} = \mathbf{MN}$ *for two matrices* $\mathbf{M}$ *and* $\mathbf{N}$*, which happens in many task scenarios. One approach is to partition* $\mathbf{M}$ *along the row side into* m *pieces* $\mathbf{M}_1, \ldots, \mathbf{M}_m$*, and then compute* $\mathbf{P}_1 = \mathbf{M}_1\mathbf{N}, \ldots, \mathbf{P}_m = \mathbf{M}_m\mathbf{N}$ *sequentially, which put together gives* $\mathbf{P}$*. In this way, the worker node can forward each piece* $\mathbf{P}_i$ *of the local result to the master once the piece has been computed out, no need for waiting others. Such computation is in the while-mode. If the worker node adopts the faster Strassen algorithm [32] to perform the matrix-matrix multiplication, then the result of* $\mathbf{P}$ *will be computed out as a whole other than piece by piece. Then it is in after-mode.*

C. Design Objectives

As mentioned in Sec. I, in most of the existing coding schemes for distributed computing, the master node only uses the local results from the fastest k worker nodes for decoding where k is the recovery threshold of the scheme, which ignores the partial work done by other worker nodes. One promising approach to improving the performance is to leverage the work done by every worker node, no matter the work done is complete or partial. To capture the full utilization of the work done by every worker node, we will formalize two design objectives: communication at full speed for the after-mode and computation at full speed for the while-mode.

1) Communication at Full Speed: We first consider a coding scheme Ψ used in the after-mode. Most often the number of worker nodes that have completed their local computations is larger than $\mathsf{rec}(\Psi)$ before the master node collects the entire local results from $\mathsf{rec}(\Psi)$ worker nodes. One way to improve the performance is to leverage all the communication links to forward useful information to the master for recovering the output result. Since there are redundancies among the local results of such worker nodes, one key issue is to keep the worker nodes from sending redundant data before the master node can decode.

To deal with this issue, two important features should be taken into account. One is that, which worker nodes and how many worker nodes would finish their local computations before the master can decode are unknown beforehand. The other is that, during execution, the worker nodes can have various computation and communication speeds, even if they are not stragglers [7]. In order to be adaptive to different situations that may happen during execution, one natural approach is to split the local result of each worker node into multiple pieces. One desired property about the pieces forwarded to the master node before the master can decode is that there is no information redundancy among them. This leads to the following definition.

Definition 2 (Communication at Full Speed)**.** Suppose a coding scheme Ψ is performed in the after-mode, where each local result is split into m ($m > 1$) equal-sized pieces. If the master can recover the output result from any $m \cdot \mathsf{rec}(\Psi)$ pieces of the total mn ones (with probability p, resp.), then we say that Ψ achieves communication at full speed (with probability p resp.).

Note that if there are only $\mathsf{rec}(\Psi)$ non-stragglers, each of them should send the whole local result to the master node. Hence, $m \cdot \mathsf{rec}(\Psi)$ is the minimum number of pieces required by the master node to decode successfully in the worst case. So, if a coding scheme Ψ achieves communication at full speed, then it can fully utilize all the parallel communication links between the master node and the worker nodes that have completed their local computations, while no redundant transmission is incurred before the master node can recover the result.

2) Computation at Full Speed: For the while-mode, we aim at fully exploiting all the partial results computed out by all the worker nodes. The desired property, which is very similar to the one in the after-mode, is defined as follows.

Definition 3 (Computation at Full Speed)**.** Consider a coding scheme Ψ performed in the while-mode, where the local computation process consists m homogeneous consecutive sub-processes each of which produces an equal-sized piece of the local result. If the master can recover the output result from any $m \cdot \mathsf{rec}(\Psi)$ pieces of the total mn ones (with probability p, resp.), then we say that it achieves computation at full speed (with probability p, resp.).

It is worth mentioning that, if a coding scheme achieves computation at full speed in the while-mode, then applying it in the after-mode, it can achieve communication at full speed. But the converse does not hold in general.

III. Polynomial-based Coding Framework

In this paper, we are not going to design completely new coding schemes, but focus on improving many existing nicely-behaved coding schemes for a variety of task scenarios such that they can achieve communication at full speed, and in some cases achieve computation at full speed. Specifically, we will consider a general polynomial-based coding framework, where the encoding functions e_i and the local computing functions h_i, $i \in [n]$, satisfy the following properties:

- for each $W_i \in \mathcal{W}$, its local result $\mathbf{Y}_i$ is equal to the evaluation of a polynomial given as

$$\mathbf{Y}_i = g(\alpha) = \mathbf{Z}_1 + \mathbf{Z}_2\alpha + \cdots + \mathbf{Z}_k\alpha^{k-1} \qquad (1)$$

  for some integer k (which is its recovery threshold as explained below) over a point $\alpha_i \in \mathbb{F}$, i.e., $\mathbf{Y}_i = g(\alpha_i)$, where $\mathbf{Z}_i \in \mathbb{F}^{r\times s}$ and α_i, $i \in [n]$, are distinct and non-zero, and
- the required output $f(\mathbf{X}_1, \mathbf{X}_2, \ldots, \mathbf{X}_d)$ can be decoded uniquely and efficiently from the coefficients $\mathbf{Z}_1, \mathbf{Z}_2, \ldots, \mathbf{Z}_k$ of the polynomial.

In fact, $\mathbf{Y}_1, \ldots, \mathbf{Y}_n$ can be viewed as a codeword of an (n, k) Reed-Solomon code corresponding to the source word $\mathbf{Z}_1, \ldots, \mathbf{Z}_k$. So $\mathbf{Z}_1, \mathbf{Z}_2, \ldots, \mathbf{Z}_k$ can be decoded by applying any decoding algorithm of Reed-Solomon codes, or equivalently, by interpolating of a polynomial with degree $k-1$. Hence, the master node only needs to collect the local results of any k out of n worker nodes. In other words, a coding scheme that follows this framework has a recovery threshold of k, and we refer to such a coding scheme as an (n, k) *polynomial-based coding scheme*.

The above framework encapsulates many advanced coding schemes for a variety of computing tasks that are fundamental to large-scale machine learning and big data analytics, including:

- Reed-Solomon codes for matrix-vector multiplication or matrix-matrix multiplication [7];
- Polynomial codes [15], MatDot codes [17], and PolyDot codes [17] for high dimensional matrix-matrix multiplication, where both matrices are too large to store in a single worker node.
- Lagrange coding for multivariate polynomial evaluation [28];
- Reed-Solomon codes for distributed gradient descent [23].

Here we introduce the Polynomial codes [15] for high-dimensional matrix-matrix multiplication as an example, which will be further considered in Sec. V.

Example 2. *Consider the task of computing* $\mathbf{MN}$ *for two matrices* $\mathbf{M}$ *and* $\mathbf{N}$*, where both* $\mathbf{M}$ *and* $\mathbf{N}$ *are too large to store in a single worker node. A Polynomial code works as follows. The matrix* $\mathbf{M}$ *is partitioned along the row side into* M *parts* $\mathbf{M}_1, \ldots, \mathbf{M}_M$*, and the matrix* $\mathbf{N}$ *is partitioned along the column side into* N *parts* $\mathbf{N}_1, \ldots, \mathbf{N}_N$*. Each worker node* W_i*,* $i \in [n]$ *is assigned with* $\tilde{\mathbf{M}}_i$ *and* $\tilde{\mathbf{N}}_i$*, where*

$$\tilde{\mathbf{M}}_i = \sum_{j=1}^{M} \mathbf{M}_j \alpha_i^{j-1}, \quad \tilde{\mathbf{N}}_i = \sum_{\ell=1}^{N} \mathbf{N}_\ell \alpha_i^{(\ell-1)M},$$

and computes

$$\mathbf{Y}_i = \tilde{\mathbf{M}}_i \tilde{\mathbf{N}}_i = \sum_{j=1}^{M}\sum_{\ell=1}^{N} \mathbf{M}_j \mathbf{N}_\ell \alpha_i^{j-1+(\ell-1)M}.$$

Since $\mathbf{Y}_i$ *can be viewed as an evaluation of a polynomial with degree* $MN-1$ *and coefficients* $\{\mathbf{M}_j\mathbf{N}_\ell\}_{j\in[M],\ell\in[N]}$ *at point* α_i*, when* α_i *are all distinct,* $\{\mathbf{M}_j\mathbf{N}_\ell\}_{j\in[M],\ell\in[N]}$*, which put together is just* $\mathbf{MN}$*, can be recovered uniquely from any (and at least)* MN *local results. So the recovery threshold is* MN*, which is proven to be optimal when each worker node can only store* $\frac{1}{M}$*-th of* $\mathbf{M}$ *and* $\frac{1}{N}$*-th of* $\mathbf{N}$*, and its communication traffic is at most* $\frac{1}{MN}$*-th of* $\mathbf{MN}$ *[15], [17].*

IV. Achieving Communication at Full Speed

In this section, we will present a randomized approach that can be applied to any (n, k) polynomial-based coding scheme so that the coding scheme can achieve communication

at full speed. The total runtime, i.e., the time required for the master node to recover the output, is also analyzed under a probabilistic model.

A. Description of the Randomized Approach

Consider an (n, k) polynomial-based coding scheme. Suppose that worker node W_i has completed its local computation and got $\mathbf{Y}_i$. Instead of sending $\mathbf{Y}_i$ directly to the master, worker i first partitions $\mathbf{Y}_i$ into m equal-sized pieces for some positive integer m.[1] The partition of $\mathbf{Y}_i$ can be done along the row side, along the column side or in other ways. For ease of presentation, we consider a partition of $\mathbf{Y}_i$ along the row side, i.e., $\mathbf{Y}_i = [\mathbf{Y}_{i,1}^T, \mathbf{Y}_{i,2}^\mathsf{T}, \ldots, \mathbf{Y}_{i,m}^\mathsf{T}]^\mathsf{T}$, where the superscript T denotes the transpose of a matrix. For each $i \in [k]$, we write $\mathbf{Z}_i = [\mathbf{Z}_{i,1}^\mathsf{T}, \mathbf{Z}_{i,2}^\mathsf{T}, \ldots, \mathbf{Z}_{i,m}^\mathsf{T}]^\mathsf{T}$ and for each $j \in [m]$, define

$$g_j(\alpha) = \mathbf{Z}_{1,j} + \mathbf{Z}_{2,j}\alpha + \cdots + \mathbf{Z}_{k,j}\alpha^{k-1}.$$

Then we have $\mathbf{Y}_{i,j} = g_j(\alpha_i)$.

Encoding for communication: Instead of forwarding $\mathbf{Y}_{i,j}$ directly to the master node, worker node W_i will generate m coded pieces that are *random linear combinations* of pieces $\mathbf{Y}_{i,j}$ defined as

$$\tilde{\mathbf{Y}}_{i,\ell} = \sum_{j=1}^{m} c_{i,\ell,j}\mathbf{Y}_{i,j},\ \ell \in [m],$$

where $c_{i,\ell,1} = 1$ for each $i \in [n], \ell \in [m]$, and each $c_{i,\ell,j}$, $i \in [n], \ell \in [m], j \in [m] \setminus \{1\}$ is independently and uniformly chosen from $\mathbb{F}$ at random if $\mathbb{F} = \mathbb{F}_q$ or from the interval $(0, 1)$ if $\mathbb{F} = \mathbb{R}$. This encoding procedure can be integrated into the local computation function.

Decoding at the master node: Since

$$\tilde{\mathbf{Y}}_{i,\ell} = \sum_{j=1}^{m} c_{i,\ell,j}\left(\sum_{p=1}^{k}\mathbf{Z}_{p,j}\alpha_i^{p-1}\right) = \sum_{j=1}^{m}\sum_{p=1}^{k} c_{i,\ell,j}\alpha_i^{p-1}\mathbf{Z}_{p,j},$$

each coded piece can be viewed as a linear combination of all $\{\mathbf{Z}_{i,j}\}_{i\in[n],j\in[m]}$. Hence, when the master node collects enough coded pieces from all the workers, it can recover $\{\mathbf{Z}_{i,j}\}_{i\in[k],j\in[m]}$ by solving the corresponding linear system, which is formally explained in the following.

For a row vector $\mathbf{r} = [r_1, \ldots, r_p]$ and a matrix $\mathbf{M} = [\mathbf{m}_1^\mathsf{T}, \ldots, \mathbf{m}_p^\mathsf{T}]^\mathsf{T}$ with p rows, define

$$\mathbf{r} \circ \mathbf{M} = [r_1\mathbf{m}_1^\mathsf{T}, \ldots, r_p\mathbf{m}_p^\mathsf{T}]^\mathsf{T}.$$

Let

$$\mathbf{A} = \begin{bmatrix} \alpha_1^0 & \alpha_1^1 & \cdots & \alpha_1^{k-1} \\ \vdots & \vdots & & \vdots \\ \alpha_n^0 & \alpha_n^1 & \cdots & \alpha_n^{k-1} \end{bmatrix} \tag{2}$$

be an $n \times k$ Vandermonde matrix defined over points $\alpha_1, \alpha_2, \ldots, \alpha_n$. Further define

$$\mathbf{B} = \begin{bmatrix} \mathbf{A} & \mathbf{c}_{1,2} \circ \mathbf{A} & \cdots & \mathbf{c}_{1,m} \circ \mathbf{A} \\ \vdots & \vdots & & \vdots \\ \mathbf{A} & \mathbf{c}_{m,2} \circ \mathbf{A} & \cdots & \mathbf{c}_{m,m} \circ \mathbf{A} \end{bmatrix}, \tag{3}$$

[1]We assume that m can be divided by the number of elements in $\mathbf{Y}_i$. If not, the assumption can be satisfied by appending some zeros.

where $\mathbf{c}_{i,j} = [c_{1,i,j}, c_{2,i,j}, \ldots, c_{m,i,j}]$, $i \in [m], j \in [m] \setminus \{1\}$. It is straightforward to check that

$$\begin{aligned}&[\tilde{\mathbf{Y}}_{1,1}^\mathsf{T}, \tilde{\mathbf{Y}}_{2,1}^\mathsf{T}, \ldots, \tilde{\mathbf{Y}}_{n,1}^\mathsf{T}, \ldots, \tilde{\mathbf{Y}}_{1,m}^\mathsf{T}, \tilde{\mathbf{Y}}_{2,m}^\mathsf{T}, \ldots, \tilde{\mathbf{Y}}_{n,m}^\mathsf{T}]^\mathsf{T} \\ &= \mathbf{B}[\mathbf{Z}_{1,1}^\mathsf{T}, \ldots, \mathbf{Z}_{k,1}^\mathsf{T}, \ldots, \mathbf{Z}_{1,m}^\mathsf{T}, \ldots, \mathbf{Z}_{k,m}^\mathsf{T}]^\mathsf{T},\end{aligned} \tag{4}$$

which is a linear system containing mk unknowns, i.e., $\mathbf{Z}_{i,j}$, $i \in [k], j \in [m]$. Note that each piece received by the master corresponds to an equation in the linear system. When the master receives enough pieces, it can recover $\{\mathbf{Z}_{i,j}\}_{i\in[k],j\in[m]}$ by solving (4) using *Gaussian elimination*.

Additional computation cost: It is straightforward to see that the above encoding for communication scheme incurs a computation cost of $O(m|\mathbf{Y}_i|)$, where $|\mathbf{Y}_i|$ denotes the size of $\mathbf{Y}_i$, and the Gaussian elimination based decoding of $\mathbf{Z}_i$, $i \in [k]$, will incur a computational cost of $O((mk)^3 + mk^2|\mathbf{Z}_i|)$, where the first term is the time cost for inverting a matrix and the second term is the time cost for using the matrix inverse to recover the result. As demonstrated by our experiments (c.f. Sec. VII), it is usually good enough to set m as a very small integer (e.g., 10). Hence, the computation cost for encoding is usually much lower than that of the local computing function and thus negligible. On the other hand, compared to the original polynomial-based scheme which admits an efficient decoding algorithm with complexity $O(k|\mathbf{Z}_i| \log^2 k \log\log k)$ based on polynomial interpolation [33], our scheme increases the computational cost of decoding roughly by a factor of $O(\frac{mk}{\log^2 k \log\log k})$. When k is not large as in many practical scenarios, this will not cause a significant delay on the whole computation.

B. Theoretical Result

Despite its simplicity, our randomized approach is very powerful since it can lead the polynomial-based coding scheme to achieve communication at full speed with probability 1 (close to 1, resp.) if $\mathbb{F} = \mathbb{R}$ (if $\mathbb{F} = \mathbb{F}_q$ and q is large enough, resp.), as formally stated in the following theorem, where $\mathsf{rk}(\cdot)$ denotes the rank of a matrix.

Theorem 1. *Let $\tilde{\mathbf{B}}$ be a matrix formed by any arbitrary mk rows of $\mathbf{B}$.*

- *If $\mathbb{F} = \mathbb{F}_q$, then $\Pr(\mathsf{rk}(\tilde{\mathbf{B}}) = mk) \geq 1 - \frac{(m-1)(2^k-1)}{q}$.*
- *If $\mathbb{F} = \mathbb{R}$, then $\Pr(\mathsf{rk}(\tilde{\mathbf{B}}) = mk) = 1$.*

This implies that an (n, k) polynomial-based coding scheme, after applying our approach, achieves communication at full speed with probability at least $1 - \frac{(m-1)(2^k-1)}{q}$ if $\mathbb{F} = \mathbb{F}_q$, or with probability 1 if $\mathbb{F} = \mathbb{R}$.

Proof. Since the proof of the main result is technical and lengthy, Sec. VI is dedicated to presenting the proof. □

Remark 1. As mentioned earlier, it is good enough to choose m as a small integer. So when k is not large and a large finite field is used, the probability of achieving communication at full speed can be very close to 1. For example, consider $k = 50$, $m = 10$. If we use a field size of 2^{64}, this probability is at least 0.99945.

C. Analysis of the Total Runtime

In this subsection, we analyze the total runtime of our approach based on a widely used probabilistic model. Same as [7], [21], we assume that both the computation time and the communication time for the whole result follow shifted exponential distributions, each of which is the sum of a constant and an exponential random variable, and for each worker node, the communication time for different pieces of the result is the same. Formally, for worker node W_i, let T_i^{comp} and T_i^{comm} denote its computation time and its communication time for transmitting the whole result. Their distributions are characterized by

$$T_i^{\mathsf{comp}} - \tau_1 \sim E(\lambda_1), \text{ and } T_i^{\mathsf{comm}} - \tau_2 \sim E(\lambda_2) \quad (5)$$

where $E(\lambda)$ denotes the exponential distribution with mean $1/\lambda$. Also, T_i^{comp} and T_i^{comm}, $i = 1, 2, \ldots, n$ are mutually independent. Moreover, the time cost for worker node W_i transmitting a single piece of the result is given by T_i^{comm}/m.

Let T^{total} denote the total runtime. To ease the presentation, we assume that the master node can recover the desired output upon receiving mk pieces of the results according to Theorem 1. In other words, T^{total} is the time required for the master node to collect mk pieces of the results from all the worker nodes. Let $N_i(t)$ denote the number of pieces received by the master node from worker node W_i before time t. Define $\sigma(t) = \min\left\{\frac{t-\tau_1}{\tau_2}, 1\right\}$. It is straightforward to check that $N_i(t) \leq \lfloor \sigma(t)m \rfloor$. For any positive integer $j \leq \lfloor \sigma(t)m \rfloor$, let $t_j = t - \tau_1 - j\tau_2/m$. We have

$$\begin{aligned}\Pr(N_i(t) \geq j) &= \Pr(t \geq T_i^{\mathsf{comp}} + jT_i^{\mathsf{comm}}/m) \\ &= \iint\limits_{0<x+jy/m<t_j} \lambda_1 e^{-\lambda_1 x} \lambda_2 e^{-\lambda_2 y} \mathrm{d}x\mathrm{d}y \\ &= 1 - \frac{\lambda_1 j e^{-\lambda_2 m t_j/j} - \lambda_2 m e^{-\lambda_1 t_j}}{\lambda_1 j - \lambda_2 m}\end{aligned}$$

Since $N_i(t)$ only takes non-negative integer values,

$$\begin{aligned}\mathsf{E}[N_i(t)] &= \sum_{j=1}^{\infty} \Pr(N_i(t) \geq j) = \sum_{j=1}^{\lfloor \sigma(t)m \rfloor} \Pr(N_i(t) \geq j) \\ &= \sum_{j=1}^{\lfloor \sigma(t)m \rfloor} \left(1 - \frac{\lambda_1 j e^{-\lambda_2 m t_j/j} - \lambda_2 m e^{-\lambda_1 t_j}}{\lambda_1 j - \lambda_2 m}\right)\end{aligned}$$

It is straightforward to show that $\mathsf{E}[N_i(t)]$ satisfies the following properties:

- $\mathsf{E}[N_i(t)]$ is continuous over (τ_1, ∞).
- $\mathsf{E}[N_i(t)]$ is strictly increasing over (τ_1, ∞).
- When t goes to infinity, $\mathsf{E}[N_i(t)]$ approaches m.

Hence, there must be a unique $t^* > \tau_1$ such that

$$\mathsf{E}[N_i(t^*)] = mk/n. \quad (6)$$

In practice, the recovery threshold k and the number of worker nodes n have the same order. Without loss of generality, we assume that $\delta = k/n$ is a fixed constant. Then we can have a tight characterization of T^{total} as stated in the following theorem.

Theorem 2. *For any positive constant ε, there exists some constant $\varepsilon' > 0$ that depends on $\tau_1, \tau_2, \lambda_1, \lambda_2, \delta$ but not n, such that*

$$\Pr(|T^{\mathsf{total}} - t^*| > \varepsilon t^*) \leq 2e^{-2\varepsilon'^2 \delta^2 n}. \quad (7)$$

Proof. Without loss of generality, we can assume that $\varepsilon < 1 - \frac{\tau_1}{t^*}$. Since $\mathsf{E}[N_i(t)]$ is continuous and strictly increasing over (τ_1, ∞), for any positive constant $\epsilon < 1 - \tau_1/t^*$, there exists some constant $\varepsilon_1 > 0$ that may depend on $\tau_1, \tau_2, \lambda_1, \lambda_2, \delta$ but not n such that $\mathsf{E}[N_i((1+\varepsilon)t^*)] = (1+\varepsilon_1)mk/n$. By the linearity of expectation, we have

$$\mathsf{E}\left[\sum_{i=1}^{n} N_i((1+\varepsilon)t^*)\right] = (1+\varepsilon_1)mk.$$

Since $N_i((1+\varepsilon)t^*)$, $i \in [n]$ are identically and independently distributed, and $0 \leq N_i((1+\varepsilon)t^*) \leq m$, by the Hoeffding's inequality, we have

$$\begin{aligned}&\Pr\left(\sum_{i=1}^{n} N_i((1+\varepsilon)t^*) < mk\right) \\ &= \Pr\left(\sum_{i=1}^{n} N_i((1+\varepsilon)t^*) - \mathsf{E}\left[\sum_{i=1}^{n} N_i((1+\varepsilon)t^*)\right] < -\varepsilon_1 mk\right) \\ &\leq \exp\left(-\frac{2(\varepsilon_1 mk)^2}{nm^2}\right) = e^{-2\varepsilon_1^2\delta^2 n}.\end{aligned}$$

Equivalently,

$$\Pr(T^{\mathsf{total}} > (1+\varepsilon)t^*) \leq e^{-2\varepsilon_1^2\delta^2 n}.$$

Similarly, we can show that there exists some constant $\varepsilon_2 > 0$ that may depend on $\tau_1, \tau_2, \lambda_1, \lambda_2, \delta$ but not n such that

$$\Pr(T^{\mathsf{total}} < (1-\varepsilon)t^*) \leq e^{-2\varepsilon_2^2\delta^2 n}.$$

By setting $\varepsilon' = \min\{\varepsilon_1, \varepsilon_2\}$ and applying the union bound, we can get (7). The proof is accomplished. □

V. Achieving Computation at Full Speed

In the approach proposed for achieving communication at full speed in Sec. IV, every transmitted piece is a linear combination of all the pieces. Evidently, this requires all the pieces to be computed out. So this approach cannot lead the polynomial-based coding scheme to achieve computation at full speed in the while-mode. Nevertheless, in this section, we show that for some typical task scenarios, we can modify this approach to achieve computation at full speed. The total runtime of such coding scheme is also analyzed.

A. The Approach

The key idea of our approach is that, for some computing tasks, the linear encoding of the pieces of the local result can be mapped into a part of the encoding on the dataset, so that each piece of the new local result can be viewed as a linear combination of the pieces of the original local

result. In the following, we will take the Polynomial code for high dimensional matrix-matrix multiplication described in Example 2 as an example to elaborate on this idea. Our approach can also be applied to other coding schemes, e.g., Reed-Solomon codes for matrix-vector multiplication/matrix-matrix multiplication [7], MatDot codes and PolyDot codes for high dimensional matrix-matrix multiplication [17], and the Lagrange coding for evaluation over some proper multivariate polynomials [28], in almost the same way.

Consider the approach proposed in Sec. IV to make the coding scheme achieve communication at full speed. We will further use the related notations given in Sec. III and Sec. IV-A. Now suppose that each $\mathbf{M}_j$, $j \in [x]$ and $\tilde{\mathbf{M}}_i$, $i \in [x]$ are partitioned along the row side into m pieces denoted by $\mathbf{M}_{j,1}, \mathbf{M}_{j,2}, \ldots, \mathbf{M}_{j,m}$, and $\tilde{\mathbf{M}}_{i,1}, \tilde{\mathbf{M}}_{i,2}, \ldots, \tilde{\mathbf{M}}_{i,m}$, respectively. It is straightforward to see that, for any $\ell \in [m]$,

$$\mathbf{Y}_{i,\ell} = \tilde{\mathbf{M}}_{i,\ell}\tilde{\mathbf{N}}_i = \left(\sum_{j=1}^{x} \mathbf{M}_{j,\ell}\alpha_i^{j-1}\right)\tilde{\mathbf{N}}_i.$$

Then for each $\ell \in [m]$, $\tilde{\mathbf{Y}}_{i,\ell}$ is equal to

$$\begin{aligned}\tilde{\mathbf{Y}}_{i,\ell} &= \sum_{j=1}^{m} c_{i,\ell,j}\left(\sum_{p=1}^{x} \mathbf{M}_{j,p}\alpha_i^{j-1}\right)\tilde{\mathbf{N}}_i \\ &= \left(\sum_{j=1}^{m}\sum_{p=1}^{x} c_{i,\ell,j}\mathbf{M}_{j,p}\alpha_i^{j-1}\right)\tilde{\mathbf{N}}_i.\end{aligned}$$

Denote $\sum_{j=1}^{m}\sum_{p=1}^{x} c_{i,\ell,j}\mathbf{M}_{j,p}\alpha_i^{j-1}$ by $\bar{\mathbf{M}}_{i,\ell}$ so that $\tilde{\mathbf{Y}}_{i,\ell} = \bar{\mathbf{M}}_{i,\ell}\tilde{\mathbf{N}}_i$. Note that $\bar{\mathbf{M}}_{i,\ell}$ can be obtained by linearly encoding the input of $\mathbf{M}$. Now we achieve at the following coding scheme: *each worker node W_i is assigned with $\{\bar{\mathbf{M}}_{i,\ell}\}_{\ell\in[m]}$ and $\tilde{\mathbf{N}}_i$, and it computes and forwards $\bar{\mathbf{M}}_{i,\ell}\tilde{\mathbf{N}}_i$ sequentially.* As the pieces each worker node W_i forwards to the master node are just $\tilde{\mathbf{Y}}_{i,1}, \tilde{\mathbf{Y}}_{i,2}, \ldots, \tilde{\mathbf{Y}}_{i,m}$, the decoding can be done in the same way as in the randomized approach for communication at full speed.

Theorem 3. *The above coding scheme for matrix-matrix multiplication in the while-mode achieves computation at full speed with probability at least $1 - \frac{(m-1)(2^k-1)}{q}$ if $\mathbb{F} = \mathbb{F}_q$, or with probability 1 if $\mathbb{F} = \mathbb{R}$.*

Proof. According to the above analysis, this is a direct consequence of Theorem 1. □

1) Comparison with a Finer Granularity Approach: There is another finer granularity approach for achieving computation at full speed: each of the n worker nodes is viewed as a set of m virtual worker nodes, and we apply an (mn, mk) polynomial-based coding scheme to the mn virtual worker nodes. Hence, each worker is assigned with m pieces of encoded data, and each worker computes over each piece of data and sends the result to the master sequentially. It is straightforward to see that this scheme can achieve computation at full speed.

However, this finer granularity based approach can incur some cost that scales with m. To see this, let us also consider the application of this finer granularity based approach to the Polynomial code based matrix-matrix multiplication. When applying this approach, the numbers of partitions of matrices $\mathbf{M}$ and $\mathbf{N}$, say M and N respectively, satisfy $MN = mk$. Hence, the storage cost of each worker node under this approach is $m\left(\frac{|\mathbf{M}|}{M} + \frac{|\mathbf{N}|}{N}\right)$, where $|\mathbf{M}|$ and $|\mathbf{N}|$ denote their respective sizes. Assuming that $\mathbf{M}$ and $\mathbf{N}$ have the same size, the storage cost is

$$m\frac{M+N}{MN}|\mathbf{M}| \geq \frac{2m\sqrt{MN}}{MN}|\mathbf{M}| = 2|\mathbf{M}|\sqrt{\frac{m}{k}}.$$

In contrast, both the original polynomial code as well as our approach have the same storage cost of $\frac{|\mathbf{M}|}{M} + \frac{|\mathbf{N}|}{N}$ where $MN = k$. If we set $M = N$, then the storage cost is $2|M|/\sqrt{k}$. Now we see that the finer granularity based approach incurs $\sqrt{m}$ times storage cost of our approach, which is costly even when m is a small integer.

Besides, both the finer granularity approach and our approach can be applied to MatDot codes [17] for the high-dimensional matrix-matrix multiplication. The same communication traffic of each worker node in our approach is the same as in the original MatDot code. In contrast, the finer granularity approach makes the communication traffic m times as the original one, which makes it hardly to apply in practice. The details are omitted.

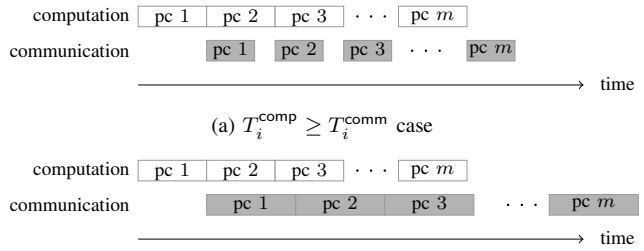
B. Analysis of the Runtime

Now we analyze the total runtime of this approach based on a probability model which is almost the same as used in Sec IV-C. We also use T_i^{comp} and T_i^{comm} to denote the whole computation time and the whole communication time for the local result, whose distributions are given in (5), respectively. Also, T_i^{comp} and T_i^{comm}, $i \in [n]$ are mutually independent. Moreover, the time cost for worker i computing (transmitting, resp.) a single piece of the result is given by T_i^{comp}/m (T_i^{comm}/m, resp.).

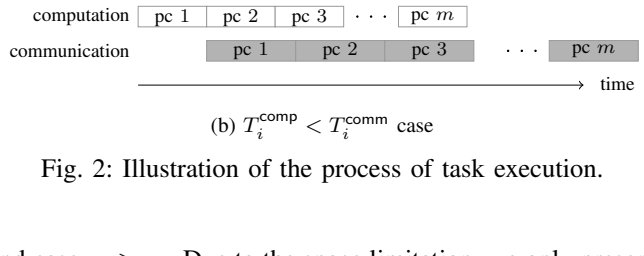
Since the analysis is similar to the one in Sec. IV-C, many details are omitted to save the space. Let $N_i'(t)$ denote the number of pieces received by the master from worker node W_i before time t. Denote $\lfloor \frac{mt}{\max\{\tau_1,\tau_2\}} \rfloor - 1$ by $\sigma'(t)$. We have that $N_i'(t) \leq \sigma'(t)$. For any positive integer j such that $j \leq \sigma'(t)$,

$$\begin{aligned}&\Pr(N_i'(t) \geq j) \\ &= \Pr(N_i'(t) \geq j, T_i^{\mathsf{comp}} \geq T_i^{\mathsf{comm}}) \\ &\quad + \Pr(N_i'(t) \geq j, T_i^{\mathsf{comp}} < T_i^{\mathsf{comm}}) \\ &= \Pr\left(t \geq \frac{jT_i^{\mathsf{comp}} + T_i^{\mathsf{comm}}}{m}, T_i^{\mathsf{comp}} \geq T_i^{\mathsf{comm}}\right) \\ &\quad + \Pr\left(t \geq \frac{T_i^{\mathsf{comp}} + jT_i^{\mathsf{comm}}}{m}, T_i^{\mathsf{comp}} < T_i^{\mathsf{comm}}\right)\end{aligned}$$

where the second step holds as illustrated in Fig. 2. The probability is calculated considering two cases, case $\tau_1 \leq \tau_2$



(a) $T_i^{\mathsf{comp}} \geq T_i^{\mathsf{comm}}$ case



(b) $T_i^{\mathsf{comp}} < T_i^{\mathsf{comm}}$ case

Fig. 2: Illustration of the process of task execution.

and case $\tau_1 > \tau_2$. Due to the space limitation, we only present the result for case $\tau_1 \leq \tau_2$. We get that, in this case,

$$\begin{aligned}\Pr(N_i'(t) \geq j) =& \lambda_1 a(t) \left(\frac{j e^{-\frac{\lambda_2 mt}{j}}}{j\lambda_1 - \lambda_2} b(t,j) + \frac{e^{-\lambda_2 mt}}{\lambda_2 j - \lambda_1} c(t,j) \right) \\ & - e^{\lambda_1 \left(\tau_1 - \frac{mt-\tau_2}{j}\right)} + 1,\end{aligned}$$

where $a(t) = e^{\lambda_1\tau_1 + \lambda_2\tau_2}$, $b(t,j) = e^{(\frac{\lambda_2}{j} - \lambda_1)\frac{mt}{j+1}} - e^{(\frac{\lambda_2}{j} - \lambda_1)\tau_1}$, and $c(t,j) = e^{(\lambda_2 j - \lambda_1)\frac{mt-\tau_2}{j}} - e^{(\lambda_2 j - \lambda_1)\frac{mt}{j+1}}$. We can show that, there exists a unique $\hat{t}$ such that

$$\mathsf{E}[N_i'(\hat{t})] = \sum_{j=1}^{\sigma'(t)} \Pr(N_i'(t) \geq j) = mk/n.$$

Let $\hat{T}^{\mathsf{total}}$ denote the total runtime under our approach. We then have the following result.

Theorem 4. *For any positive constant ε, there exists some constant $\varepsilon' > 0$ that depends on $\tau_1, \tau_2, \lambda_1, \lambda_2, \delta$ but not n, such that*

$$\Pr(|\hat{T}^{\mathsf{total}} - \hat{t}| > \varepsilon\hat{t}) \leq 2e^{-2\varepsilon'^2\delta^2 n}.$$

Proof. The proof is done similarly to the one for Theorem 2, and hence omitted. □

VI. Proof of Theorem 1

Due to the space limitation, we will only prove Theorem 1 for the case $\mathbb{F} = \mathbb{F}_q$. The technical challenge for showing this result comes from the fact that $\tilde{\mathbf{B}}$ is a random matrix while exhibiting some inner structure since it is built on $\mathbf{A}$ (c.f., (3)). We will overcome this challenge by conducting proper linear transformations as well as counting based arguments.

We start with a basic property of the matrix $\tilde{\mathbf{B}}$.

Lemma 1. *By permuting rows of $\tilde{\mathbf{B}}$, we can write $\tilde{\mathbf{B}}$ as*

$$\tilde{\mathbf{B}} = \begin{bmatrix} \mathbf{A}_1 & \mathbf{w}_{1,2} \circ \mathbf{A}_1 & \cdots & \mathbf{w}_{1,m} \circ \mathbf{A}_1 \\ \vdots & \vdots & & \vdots \\ \mathbf{A}_m & \mathbf{w}_{m,2} \circ \mathbf{A}_m & \cdots & \mathbf{w}_{m,m} \circ \mathbf{A}_m \end{bmatrix},$$

where each $\mathbf{A}_i$, $i \in [m]$ is a submatrix of $\mathbf{A}$ with k rows, and each vector $\mathbf{w}_{i,j}$, $i \in [m]$, $j \in [k] \setminus \{0\}$ has m entries which are picked from $\mathbb{F}$ independently and uniformly at random.

Proof. Due to the space limitation, the proof is omitted. □

Define

$$\tilde{\mathbf{B}}_j = \begin{bmatrix} \mathbf{A}_1 & \mathbf{w}_{1,2} \circ \mathbf{A}_1 & \cdots & \mathbf{w}_{1,j} \circ \mathbf{A}_1 \\ \vdots & \vdots & & \vdots \\ \mathbf{A}_j & \mathbf{w}_{j,2} \circ \mathbf{A}_j & \cdots & \mathbf{w}_{j,j} \circ \mathbf{A}_j \end{bmatrix}.$$

In particular, $\tilde{\mathbf{B}}_1 = \mathbf{A}_1$, and $\tilde{\mathbf{B}}_m = \tilde{\mathbf{B}}$. If $\Pr(\mathsf{rk}(\tilde{\mathbf{B}}_j) = jk) > 0$ for each $j \in [m-1]$, then we have

$$\begin{aligned}&\Pr(\mathsf{rk}(\tilde{\mathbf{B}}_m) = mk) \\ &\geq \prod_{j=2}^{m} \Pr(\mathsf{rk}(\tilde{\mathbf{B}}_j) = jk \mid \mathsf{rk}(\tilde{\mathbf{B}}_{j-1}) = (j-1)k) \\ &\qquad \cdot \Pr(\mathsf{rk}(\tilde{\mathbf{B}}_1) = k) \\ &= \prod_{j=2}^{m} \Pr(\mathsf{rk}(\tilde{\mathbf{B}}_j) = jk \mid \mathsf{rk}(\tilde{\mathbf{B}}_{j-1}) = (j-1)k)\end{aligned} \tag{8}$$

where the second step holds since $\mathbf{B}_1 = \mathbf{A}_1$ has rank k.

In the following, we will establish a lower bound for the $\Pr(\mathsf{rk}(\tilde{\mathbf{B}}_j) = jk \mid \mathsf{rk}(\tilde{\mathbf{B}}_{j-1}) = (j-1)k)$. Define

$$\begin{aligned}\mathbf{C}_j &= \begin{bmatrix} \mathbf{A}_j & \mathbf{w}_{j,2} \circ \mathbf{A}_j & \cdots & \mathbf{w}_{j,j-1} \circ \mathbf{A}_j \end{bmatrix} \\ \tilde{\mathbf{w}}_{j-1} &= [\mathbf{w}_{1,j}^{\mathsf{T}}, \ldots, \mathbf{w}_{j-1,j}^{\mathsf{T}}]^{\mathsf{T}} \\ \tilde{\mathbf{A}}_{j-1} &= [\mathbf{A}_1^{\mathsf{T}}, \ldots, \mathbf{A}_{j-1}^{\mathsf{T}}]^{\mathsf{T}}.\end{aligned}$$

Then,

$$\tilde{\mathbf{B}}_j = \begin{bmatrix} \tilde{\mathbf{B}}_{j-1} & \tilde{\mathbf{w}}_{j-1} \circ \tilde{\mathbf{A}}_{j-1} \\ \mathbf{C}_j & \mathbf{w}_{j,j} \circ \mathbf{A}_j \end{bmatrix}.$$

Assume that $\mathsf{rk}(\tilde{\mathbf{B}}_{j-1}) = (j-1)k$. Since $\tilde{\mathbf{B}}_{j-1}$ has full rank of $(j-1)k$, its rows form a basis of the whole row vector space $\mathbb{F}^{(j-1)k}$. Hence there exists a $k \times (j-1)k$ matrix $\mathbf{U}$ such that $\mathbf{C}_j = \mathbf{U}\tilde{\mathbf{B}}_{j-1}$. Also, since $\mathbf{A}_j$ has full rank of k, its rows form a basis of the whole row vector space $\mathbb{F}^k$. Hence there exists a $(j-1)k \times k$ matrix $\mathbf{V}$ such that $\tilde{\mathbf{A}}_{j-1} = \mathbf{V}\mathbf{A}_j$. Then

$$\tilde{\mathbf{w}}_{j-1} \circ \tilde{\mathbf{A}}_{j-1} = \tilde{\mathbf{w}}_{j-1} \circ (\mathbf{V}\mathbf{A}_j) = (\tilde{\mathbf{w}}_{j-1} \circ \mathbf{V})\mathbf{A}_j,$$

where the second step can be easily verified by definition. By performing some elementary row operations on $\tilde{\mathbf{B}}_j$, we can obtain

$$\tilde{\mathbf{B}}_j' = \begin{bmatrix} \tilde{\mathbf{B}}_{j-1} & \tilde{\mathbf{w}}_{j-1} \circ \tilde{\mathbf{A}}_{j-1} \\ \mathbf{O} & \mathbf{w}_{j,j} \circ \mathbf{A}_j - (\mathbf{U}(\tilde{\mathbf{w}}_{j-1} \circ \mathbf{V}))\mathbf{A}_j \end{bmatrix},$$

where $\mathbf{O}$ is the zero matrix.

Let $\mathbf{a}_i$, $i \in [k]$ be the i-th row of $\mathbf{A}_j$. For any $\mathbf{y} = (y_1, \ldots, y_k) \in \mathbb{F}^k$ and $\mathbf{Z} = \{z_{\ell,p}\}_{\ell,p \in [k]} \in \mathbb{F}^{k \times k}$, define a function

$$\begin{aligned}\mathbf{D}(\mathbf{y}, \mathbf{Z}) =& \mathbf{y} \circ \mathbf{A}_j - \mathbf{Z}\mathbf{A}_j \\ =& \begin{bmatrix} y_1\mathbf{a}_1 - \sum_{p=1}^{k} z_{1,p}\mathbf{a}_p \\ \vdots \\ y_\ell\mathbf{a}_\ell - \sum_{p=1}^{k} z_{\ell,p}\mathbf{a}_p \\ \vdots \\ y_k\mathbf{a}_k - \sum_{p=1}^{k} z_{k,p}\mathbf{a}_p \end{bmatrix} \triangleq \begin{bmatrix} \mathbf{d}_1(\mathbf{y}, \mathbf{Z}) \\ \vdots \\ \mathbf{d}_\ell(\mathbf{y}, \mathbf{Z}) \\ \vdots \\ \mathbf{d}_k(\mathbf{y}, \mathbf{Z}) \end{bmatrix}\end{aligned}$$

Further define $\mathcal{Y}$ as the set

$$\left\{\mathbf{y}\in\mathbb{F}^k \mid \exists(x_1,\ldots,x_k)\in\mathbb{F}^k\setminus\{\mathbf{0}\} \text{ s.t., } \sum_{\ell=1}^{k} x_\ell \mathbf{d}_\ell(\mathbf{y},\mathbf{Z})=\mathbf{0}\right\},$$

and for each nonempty subset $\mathcal{S}\subseteq[k]$, define

$$\mathcal{Y}(\mathcal{S})=\left\{\mathbf{y}\in\mathbb{F}^k \mid \exists(x_1,\ldots,x_k)\in\mathbb{F}^k \right.$$
$$\left. \text{s.t., } \sum_{\ell=1}^{k} x_\ell \mathbf{d}_\ell(\mathbf{y},\mathbf{Z})=\mathbf{0}, \text{and } x_\ell\neq 0 \text{ iff } \ell\in\mathcal{S}\right\}$$

We will establish a bound on $|\mathcal{Y}(\mathcal{S})|$, the size of $\mathcal{Y}(\mathcal{S})$. Suppose $(x_1,\ldots,x_k)\in\mathbb{F}^k$, $x_\ell\neq 0$ iff $\ell\in\mathcal{S}$, and $\sum_{\ell=1}^{k} x_\ell\mathbf{d}_\ell(\mathbf{y},\mathbf{Z})=\mathbf{0}$. Since

$$\begin{aligned}
\sum_{\ell=1}^{k} x_\ell \mathbf{d}_\ell(\mathbf{y},\mathbf{Z}) &= \sum_{\ell=1}^{k} x_\ell\left(y_\ell\mathbf{a}_\ell-\sum_{p=1}^{k} z_{\ell,p}\mathbf{a}_p\right)\\
&= \sum_{\ell=1}^{k} x_\ell y_\ell\mathbf{a}_\ell-\sum_{\ell=1}^{k} x_\ell\sum_{p=1}^{k} z_{\ell,p}\mathbf{a}_p\\
&= \sum_{\ell=1}^{k} x_\ell y_\ell\mathbf{a}_\ell-\sum_{p=1}^{k} x_p\sum_{\ell=1}^{k} z_{p,\ell}\mathbf{a}_\ell\\
&= \sum_{\ell=1}^{k}\left(x_\ell y_\ell-\sum_{p=1}^{k} x_p z_{p,\ell}\right)\mathbf{a}_\ell\\
&= \mathbf{0}.
\end{aligned}$$

and $\mathbf{a}_1,\ldots,\mathbf{a}_k$ are linearly independent, $x_\ell y_\ell-\sum_{p=1}^{k} x_p z_{p,\ell}=0$ for each $\ell\in[k]$. Since $\mathcal{S}$ is nonempty, there exists some $i\in\mathcal{S}$ such that $x_i\neq 0$. So we have

$$y_\ell=\frac{x_i}{x_\ell}\sum_{p\in\mathcal{S}}\frac{x_p}{x_i}z_{p,\ell}, \text{for each } \ell\in\mathcal{S}. \tag{9}$$

From (9), we can see that $(y_\ell)_{\ell\in\mathcal{S}}$ is a function of $(\frac{x_p}{x_i})_{p\in\mathcal{S}}$, or equivalently, a function of $(\frac{x_p}{x_i})_{p\in\mathcal{S}\setminus\{i\}}$. Note that $(\frac{x_p}{x_i})_{p\in\mathcal{S}\setminus\{i\}}$ have at most $(q-1)^{|\mathcal{S}|-1}$ possible values. Hence $(y_1,y_2,\ldots,y_k)$ that satisfies (9) has at most $(q-1)^{|\mathcal{S}|-1}q^{k-|\mathcal{S}|}$ possibilities. Therefore,

$$|\mathcal{Y}(\mathcal{S})|\leq(q-1)^{|\mathcal{S}|-1}q^{k-|\mathcal{S}|}\leq q^{k-1}.$$

Because $\mathcal{Y}=\cup_{\mathcal{S}\subseteq[k],\mathcal{S}\neq\emptyset}\mathcal{Y}(\mathcal{S})$,

$$|\mathcal{Y}|\leq\sum_{\mathcal{S}\subseteq[k],\mathcal{S}\neq\emptyset}|\mathcal{Y}(\mathcal{S})|\leq(2^k-1)q^{k-1}.$$

Now we have

$$\Pr(\mathsf{rk}(\mathbf{D}(\mathbf{w}_{j,j},\mathbf{Z}))\neq k)=\Pr(\mathbf{w}_{j,j}\in\mathcal{Y})=\frac{|\mathcal{Y}|}{q^k}\leq\frac{2^k-1}{q},$$

where the second equality holds since each entry of $\mathbf{w}_{j,j}$ is chosen from $\mathbb{F}$ independently and uniformly at random. By the total law of probability, we can get

$$\Pr(\mathsf{rk}(\mathbf{D}(\mathbf{w}_{j,j},\mathbf{U}(\tilde{\mathbf{w}}_{j-1}\circ\mathbf{V})))\neq k \mid \mathsf{rk}(\tilde{\mathbf{B}}_{j-1})=(j-1)k)$$
$$\leq\frac{2^k-1}{q}.$$

Notice that $\mathsf{rk}(\tilde{\mathbf{B}}_j)=\mathsf{rk}(\tilde{\mathbf{B}}'_j)=\mathsf{rk}(\tilde{\mathbf{B}}_{j-1})+\mathsf{rk}(\mathbf{w}_{j,j}\circ\mathbf{A}_j-(\mathbf{U}(\tilde{\mathbf{w}}_{j-1}\circ\mathbf{V}))\mathbf{A}_j))$. Hence,

$$\Pr(\mathsf{rk}(\tilde{\mathbf{B}}_j)=jk \mid \mathsf{rk}(\tilde{\mathbf{B}}_{j-1})=(j-1)k)\geq 1-\frac{2^k-1}{q}.$$

By (8), we have

$$\begin{aligned}
\Pr(\mathsf{rk}(\tilde{\mathbf{B}}_m)=mk) &\geq\left(1-\frac{2^k-1}{q}\right)^{m-1}\\
&\geq 1-\frac{(m-1)(2^k-1)}{q}.
\end{aligned}$$

The proof of Theorem 1 is completed.

VII. Experimental Results

In this section, we conduct experiments on Alibaba Cloud [31] and also simulations to see how the runtime of an (n,k) polynomial-based coding scheme can be improved by our approaches. Specifically, we consider the following approaches for performance comparison:

- the original polynomial-based coding scheme, which is labeled as Original,
- the application of the approach proposed in Sec. IV to achieve communication at full speed, which is labeled as COMM-FS(m),
- and the application of the approach proposed in Sec. V to achieve computation at full speed, which is labeled as COMP-FS(m),

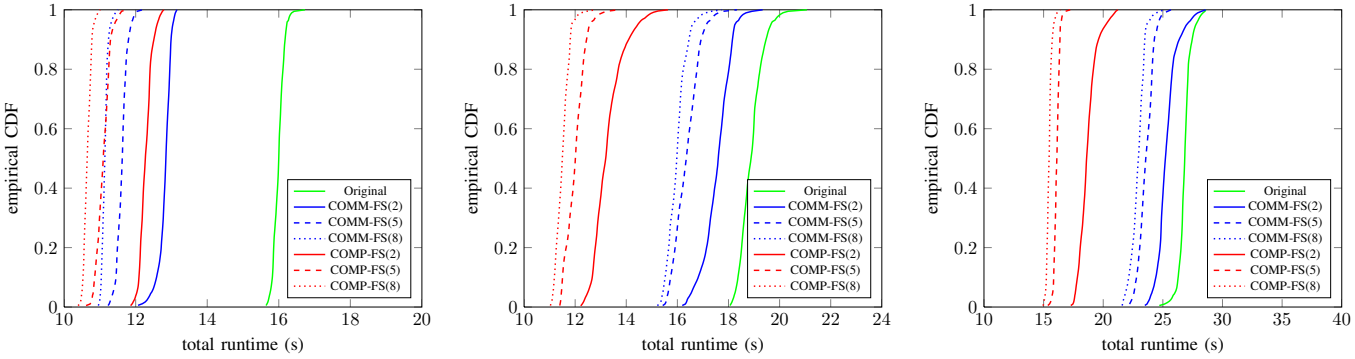
where m is the number of pieces of a local result.

A. Experiments on Alibaba Cloud

Experiment Setup: Based on Alibaba Cloud, we conduct experiments on 21 machines in a cluster. These 21 machines are homogeneous with 1-vCPU, 2GB memory and 1Mbps bandwidth, and the machine model is ecs.n4.small. Machines are connected through private Internet. Among the 21 machines, one acts as the master node, and the others act as worker nodes.

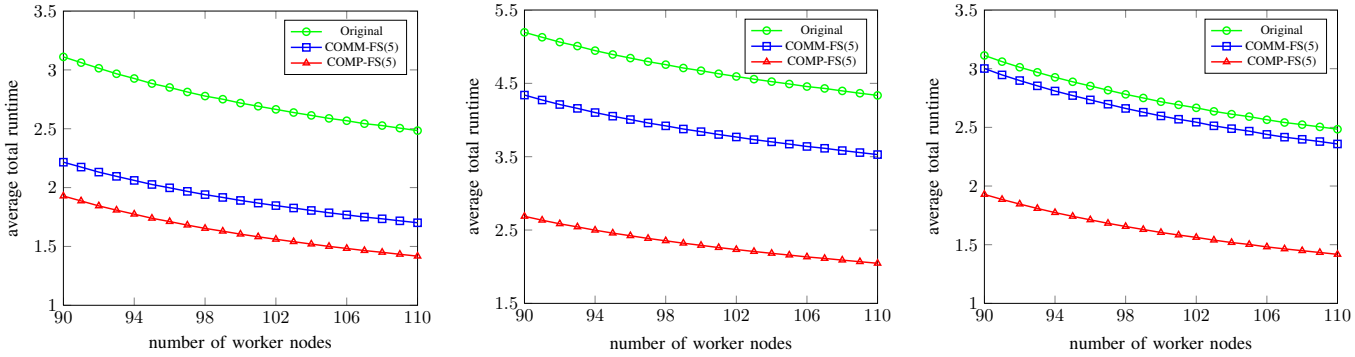
Workload: We consider the task of matrix-matrix multiplication $\mathbf{MN}$, where $\mathbf{M}\in\mathbb{R}^{32000\times b}$ and $\mathbf{N}\in\mathbb{R}^{b\times 3000}$. The setting of integer b will be explained soon. Since $\mathbf{N}$ is not very large, the original coding scheme is chosen to be a $(20,16)$ Reed-Solomon code: the matrix $\mathbf{M}$ is partitioned along the row side into 16 parts $\mathbf{M}_1,\ldots,\mathbf{M}_{16}$, which are then encoding into 20 parts $\tilde{\mathbf{M}}_1,\ldots,\tilde{\mathbf{M}}_{20}$, and each worker node computes $\tilde{\mathbf{M}}_i\mathbf{N}$ for a different i exactly, and then transmits it to the master node.

Apparently, the effect of our proposed approaches is closely related to the difference between computation time and communication time that each worker node takes to finish its task. Taking this into account, we set b to be three different values of 50, 300, and 800. In particular, when $b=50$, the average communication time that each worker node takes to transmit is about 5 times the average computation time. When $b=300$, the average communication time is very close to the average computation time. When $b=800$, the average communication time is about $1/3.5$ of the average computation time. For our



(a) the case $b = 50$ (b) the case $b = 300$ (c) the case $b = 800$

Fig. 3: The empirical cumulative distribution of the total runtime.



(a) The case $\tau_1 = 1/6$, $\lambda_1 = 6$ (b) The case $\tau_1 = 1$, $\lambda_1 = 1$ (c) The case $\tau_1 = 6$, $\lambda_1 = 1/6$

Fig. 4: Effect of the number of worker nodes.

proposed approaches, the number of pieces in a local result m is set to be 2, 5, and 8.

Experiment Result: We repeat the above experiment for each parameter setting by 100 times, and plot the empirical cumulative distribution of the total runtime in Fig. 3. From this figure, we can observe the followings:

- For all the three different values of b, both COMM-FS and COMP-FS perform much better than Original for all the values of m. In particular, compared to Original, COMM-FS(8) and COMP-FS(8) can reduce the median total runtime by 30.4% and 33.5% when $b = 50$, by 15.4% and 39.1% when $b = 300$, and by 14.5% and 42.3% when $b = 800$, respectively.
- When m grows larger, both COMM-FS and COMP-FS perform better, leading to a higher performance improvement. This follows the intuition that the larger m is, the higher utilization of the partial results of worker can be achieved. However, the performance improvement of both COMM-FS and COMP-FS by using a larger m becomes smaller when m grows. In particular, we can see that in all three cases, when m is changed from 5 to 8, the performance improvement of both COMM-FS and COMP-FS is little. This hints that, in practice, it is good enough to use a small value of m, e.g., 5 or 8.
- When b becomes larger, i.e., the ratio between the computation time and the communication time of each worker node becomes larger, the improvement of COMM-FS over Original becomes smaller. This is because COMM-FS only improves the communication phase and has no effect on the computation phase.

B. Simulations

Due to practical reasons, it is hard for us to conduct experiments involving a large number of worker nodes on Alibaba Cloud. In order to evaluate the effect of the number of worker nodes, we conduct simulations based on the probabilistic runtime model as stated in Sec. IV and Sec. V.

We consider an (n, k) polynomial-based coding scheme, where the number of worker nodes n ranges from 90 to 110, and k is chosen to be a constant 75 such that the computation overhead of each worker node is kept the same. For our approaches, the number of pieces of the local result is chosen to be 5. Regarding the parameters in the probabilistic runtime model, we set $\tau_2 = \lambda_2 = 1$, and consider three cases: $\tau_1 = 1/6, \lambda_1 = 6$, $\tau_1 = 1, \lambda_1 = 1$, and $\tau_1 = 6, \lambda_1 = 1/6$, which represents the average computation time is much less than, comparable to, and much larger than the average communication time, respectively, which is similar to the setting of b in the experiments on Alibaba Cloud.

For each parameter setting, we run the simulation for 2000 times, and plot the average total runtime in Fig. 4. From this figure, we have the following findings:

- The total runtime of all the schemes becomes smaller when n grows, since the recovery threshold is the same while there are more worker nodes.
- Even when $n = 90$ so that the coding scheme can only tolerate a few stragglers, our approaches can still significantly reduce the average total runtime.
- The runtime reduction percentage of COMM-FS and COMP-FS over Original becomes slightly larger when n grows. The main reason is that, when n becomes larger, there are more opportunities to exploit the partial work done by stragglers.

VIII. Conclusion

In this paper, we introduced two concepts, communication at full speed and computation at full speed, that characterize the full utilization of work done entirely or partially by every worker node in distributed computing. Then for a general polynomial-based coding framework, we proposed a simple yet very effective randomized approach that can lead any polynomial-based coding scheme to achieve communication at full speed, while the extra computation cost incurred is usually negligible. We also showed that by improving the randomized approach, computation at full speed can be achieved for some typical task scenarios. Both experiments conducted on Alibaba Cloud and simulations demonstrate that our approaches can lead to significant performance improvement over the original polynomial-based coding scheme. In the future, we would like to develop schemes that can achieve computation at full speed for other task scenarios.

Acknowledgment

This work was supported by the National Key R&D Program of China (No. 2018YFB1004704), the Hong Kong RGC Collaborative Research Fund (No. CityU C1008-16G), the NSFC Grants (No. 61832005, 61872171, and 61801248), the Natural Science Foundation of Jiangsu Province (No. BK20190058), the Key R&D Program of Jiangsu Province (No. BE2017152), the Science and Technology Program of State Grid Corporation of China (No. SGSHXT00JFJS1900092), and the Collaborative Innovation Center of Novel Software Technology and Industrialization.

References

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

[2] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster computing with working sets." *HotCloud*, vol. 10, no. 10-10, p. 95, 2010.

[3] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, A. Senior, P. Tucker, K. Yang, Q. V. Le *et al.*, "Large scale distributed deep networks," in *Proc. NeurIPS*, 2012, pp. 1223–1231.

[4] N. J. Yadwadkar, B. Hariharan, J. E. Gonzalez, and R. Katz, "Multi-task learning for straggler avoiding predictive job scheduling," *J. Mach. Learn. Res.*, vol. 17, no. 1, pp. 3692–3728, 2016.

[5] 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 *Proc. USENIX OSDI*, 2010, pp. 265–278.

[6] J. Dean and L. A. Barroso, "The tail at scale," *Commun. ACM*, vol. 56, no. 2, pp. 74–80, 2013.

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

[8] S. Dutta, V. Cadambe, and P. Grover, "Short-dot: Computing large linear transforms distributedly using coded short dot products," in *Proc. NeurIPS*, 2016, pp. 2100–2108.

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

[10] A. B. Das, L. Tang, and A. Ramamoorthy, "C^{3}LES: Codes for coded computation that leverage stragglers," in *Proc. IEEE ITW*, 2018, pp. 1–5.

[11] S. Kiani, N. Ferdinand, and S. C. Draper, "Exploitation of stragglers in coded computation," in *Proc. IEEE ISIT*, 2018, pp. 1988–1992.

[12] A. Mallick, M. Chaudhari, and G. Joshi, "Fast and efficient distributed matrix-vector multiplication using rateless fountain codes," in *Proc. IEEE ICASSP*, 2019, pp. 8192–8196.

[13] S. Wang, J. Liu, N. Shroff, and P. Yang, "Computation efficient coded linear transform," in *Proc. AISTATS*, 2019, pp. 577–585.

[14] K. Lee, C. Suh, and K. Ramchandran, "High-dimensional coded matrix multiplication," in *Proc. IEEE ISIT*, 2017, pp. 2418–2422.

[15] Q. Yu, M. Maddah-Ali, and S. Avestimehr, "Polynomial codes: an optimal design for high-dimensional coded matrix multiplication," in *Proc. NeurIPS*, 2017, pp. 4403–4413.

[16] Q. Yu, M. A. Maddah-Ali, and A. S. Avestimehr, "Straggler mitigation in distributed matrix multiplication: Fundamental limits and optimal coding," in *Proc. IEEE ISIT*, 2018, pp. 2022–2026.

[17] S. Dutta, M. Fahim, F. Haddadpour, H. Jeong, V. Cadambe, and P. Grover, "On the optimal recovery threshold of coded matrix multiplication," *IEEE Trans. Inf. Theory*, vol. 66, no. 1, pp. 278–301, 2019.

[18] S. Wang, J. Liu, and N. Shroff, "Coded sparse matrix multiplication," in *Proc. ICML*, 2018, pp. 5139–5147.

[19] T. Baharav, K. Lee, O. Ocal, and K. Ramchandran, "Straggler-proofing massive-scale distributed matrix multiplication with d-dimensional product codes," in *Proc. IEEE ISIT*, 2018, pp. 1993–1997.

[20] R. Tandon, Q. Lei, A. G. Dimakis, and N. Karampatziakis, "Gradient coding: Avoiding stragglers in distributed learning," in *Proc. ICML*, 2017, pp. 3368–3376.

[21] M. Ye and E. A. Abbe, "Communication-computation efficient gradient coding," in *Proc. ICML*, 2018, pp. 5610–5619.

[22] Z. Charles, D. Papailiopoulos, and J. Ellenberg, "Approximate gradient coding via sparse random graphs," *arXiv preprint arXiv:1711.06771*, 2017.

[23] W. Halbawi, N. Azizan, F. Salehi, and B. Hassibi, "Improving distributed gradient descent using reed-solomon codes," in *Proc. IEEE ISIT*, 2018, pp. 2027–2031.

[24] S. Li, S. M. M. Kalan, A. S. Avestimehr, and M. Soltanolkotabi, "Near-optimal straggler mitigation for distributed gradient methods," in *Proc. IEEE IPDPSW*, 2018, pp. 857–866.

[25] N. Raviv, R. Tandon, A. Dimakis, and I. Tamo, "Gradient coding from cyclic mds codes and expander graphs," in *Proc. ICML*, 2018, pp. 4302–4310.

[26] H. Wang, S. Guo, B. Tang, R. Li, and C. Li, "Heterogeneity-aware gradient coding for straggler tolerance," in *Proc. IEEE ICDCS*, 2019, pp. 555–564.

[27] S. Dutta, V. Cadambe, and P. Grover, "Coded convolution for parallel and distributed computing within a deadline," in *Proc. IEEE ISIT*, 2017, pp. 2403–2407.

[28] Q. Yu, S. Li, N. Raviv, S. M. M. Kalan, M. Soltanolkotabi, and S. A. Avestimehr, "Lagrange coded computing: Optimal design for resiliency, security, and privacy," in *Proc. AISTATS*, 2019, pp. 1215–1225.

[29] N. Raviv, Q. Yu, J. Bruck, and S. Avestimehr, "Download and access trade-offs in lagrange coded computing," in *Proc. IEEE ISIT*, 2019.

[30] A. Shokrollahi, "Raptor codes," *IEEE Trans. Inf. Theory*, vol. 52, no. 6, pp. 2551–2567, 2006.

[31] "Alibaba Cloud." [Online]. Available: https://www.aliyun.com/product/ecs

[32] V. Strassen, "Gaussian elimination is not optimal," *Numerische mathematik*, vol. 13, no. 4, pp. 354–356, 1969.

[33] K. S. Kedlaya and C. Umans, "Fast polynomial factorization and modular composition," *SIAM J. Comput.*, vol. 40, no. 6, pp. 1767–1802, 2011.