2019 57th Annual Allerton Conference on Communication, Control, and Computing (Allerton)
Allerton Park and Retreat Center
Monticello, IL, USA, September 24-27, 2019

Lagrange Coded Computing with Sparsity Constraints

Mohammad Fahim and Viveck R. Cadambe
Department of Electrical Engineering, Pennsylvania State University.
Email: fahim@psu.edu, viveck@engr.psu.edu.

***Abstract*—In this paper, we propose a distributed coding scheme that allows for lower computation cost per computing node than the standard Lagrange Coded Computing scheme. The proposed coding scheme is useful for cases where the elements of the input data set are of large dimensions and the computing nodes have limited computation power. This coding scheme provides a trade-off between the computation cost per worker and the recovery threshold in a distributed coded computing framework. The proposed scheme is also extended to provide data privacy against at most t colluding worker nodes in the system.**

I. Introduction

With the substantial increase in the size of today's data sets and the huge computations required to be performed on them in different machine learning applications, distributing such data on separate machines/computing nodes and processing them in parallel has been a necessity. However, once a computation is split on different computing nodes, the whole computation time is now controlled by the slowest computing nodes "stragglers". [1] reports measurements from a Google service that distributes a request to a large number of servers, measurements shows that the time spent waiting for the last 5% of servers to respond is the same as the time spent waiting for the first 95% servers to respond. That is, ignoring the last 5% servers' response doubles the speed of the whole operation. Such experiments on the effect of slow/failed computing nodes on the overall delay of the distributed systems have inspired the work on mitigating straggler nodes in order to speed up distributed computations. One way to reduce the effect of stragglers in distributed computing frameworks is through coding techniques where redundant computations are issued in a specific manner so that the desired computation can be recovered upon the completion of any subset of the computing nodes, of a specific size; and hence the stragglers can be ignored leading to speeding up the distributed computation. Recently, various codes have been introduced for different distributed computations related to machine learning applications, e.g., for matrix multiplication [2]–[10], linear solvers [11], [12], and gradient methods [13]–[15].

In addition to stragglers, another challenge that arises due to performing large scale computations on separate machines is privacy. Since performing distributed computation on large scale data may require a data owner to outsource such data to external machines to perform this job, the data owner may wish to keep this data private, i.e., unknown to the external machines, while the computation is done. This problem has been known as *secure multi-party computation (MPC)*. One of the first secure MPC protocols is BGW [16] which provides privacy guarantees for any arithmetic computation that involves any number of parties $n \geq 2$. However, a drawback of BGW is that it requires multiple rounds of communication between the parties which may lead to undesired overall delay in the computation; an issue that has been resolved by *Lagrange Coded Computing (LCC)* [17].

While most of the coding techniques in the literature of coded distributed computing have been application specific (e.g., a coding strategy that is developed for gradient methods may not be applicable for matrix multiplication), as a generalization to Systematic MatDot codes [6], a coding strategy based on Lagrange polynomials, denoted by Lagrange Coded Computing (LCC), has been introduced in [17] which provides computation redundancy for a wide set of functions. However, for mitigating stragglers, LCC provides computation redundancy only for polynomial functions and its recovery threshold is relatively high and proportional to the total degree of the polynomial function of interest. It has been shown in [17] that not only LCC can be used to speed up distributed computation but it also provides privacy guarantees in secure multi-party computations.

The goal of LCC is to compute a function of interest f acting on a specific data set $\mathcal{X} := \{X_1, \cdots, X_m\}$, i.e., to compute $f(\mathcal{X}) := \{f(X_i) : i \in [m]\}$. The key idea in adding computation redundancy using LCC lies in encoding the data set into a polynomial g such that $g(\beta_i) = X_i, \forall i \in [m]$, for some scalars $\beta_1, \cdots, \beta_m$, and sending evaluations of this polynomial g to the computing nodes where the function of interest is evaluated at the received polynomially encoded data point. That is, for a distributed system with P computing nodes, distinct scalars $\alpha_1, \cdots, \alpha_P$ are defined and for every $i \in [P]$ node i receives $g(\alpha_i)$ and computes $f(g(\alpha_i))$. Once any subset of the nodes with a size of the degree of the polynomial $f(g(\cdot))$ plus one are done, $f(g(\cdot))$ can be interpolated and evaluating $f(g(\beta_i)), \forall i \in [m]$, reveals $f(\mathcal{X})$.

Although LCC distributes the computation of $f(\mathcal{X})$ equally over the computing nodes where, for every $i \in [P]$, node i computes a single evaluation of f at the point $g(\alpha_i)$, the system design in LCC assumes that each of the distributed system's computing nodes are always capable

This work is supported by NSF grant No. CCF 1763657.

$$g_1(\alpha) = \frac{a_1(\alpha-\alpha_1)(\alpha-\alpha_3)(\alpha-\alpha_5)(\alpha-\beta_2)}{(\beta_1-\alpha_1)(\beta_1-\alpha_3)(\beta_1-\alpha_5)(\beta_1-\beta_2)} + \frac{a_2(\alpha-\alpha_1)(\alpha-\alpha_3)(\alpha-\alpha_5)(\alpha-\beta_1)}{(\beta_2-\alpha_1)(\beta_2-\alpha_3)(\beta_2-\alpha_5)(\beta_2-\beta_1)}, \text{ and}$$

$$g_2(\alpha) = \frac{b_1(\alpha-\alpha_2)(\alpha-\alpha_4)(\alpha-\alpha_6)(\alpha-\beta_2)}{(\beta_1-\alpha_2)(\beta_1-\alpha_4)(\beta_1-\alpha_6)(\beta_1-\beta_2)} + \frac{b_2(\alpha-\alpha_2)(\alpha-\alpha_4)(\alpha-\alpha_6)(\alpha-\beta_1)}{(\beta_2-\alpha_2)(\beta_2-\alpha_4)(\beta_2-\alpha_6)(\beta_2-\beta_1)}. \quad (1)$$

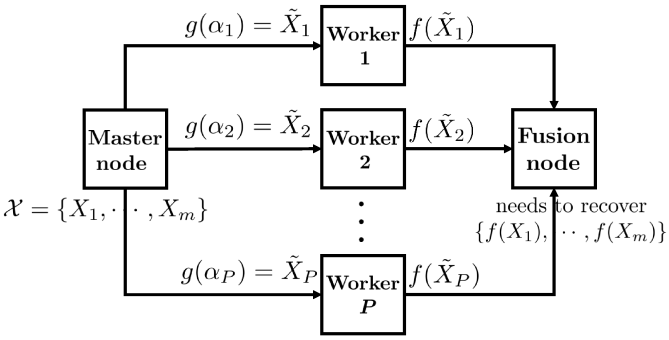

Fig. 1. The Lagrange coded computing system framework

of solely performing this computation regardless of how complex the function f is or how large the dimensions of the function f's inputs are. This poses a limitation on LCC for the cases where this assumption is not true, i.e., each single computing node is not powerful enough to compute an evaluation of f because the computation cost of f is high and/or the inputs of f are of high dimensions. One way to get around this problem is decomposing the function of interest f into multiple low complexity functions and then applying LCC to each function recursively. For example, let f be the multivariate polynomial $f(X_1, \cdots, X_m) = A\prod_{i=1}^{m} X_i$, for some constant matrix $A \in \mathbb{F}^{N\times N}$, where $X_1, \cdots, X_m$ takes values in $\mathbb{F}^{N\times N}$ for some field $\mathbb{F}$. The computational complexity of such computation is $O(mN^3)$. Assume, due to memory and processing limitations on each computing node, that a single computing node can only solve operations of complexity $O(N^3)$. To get around this limitation, we may decompose f into m functions: $f_1(X_1), f_2(X_2; X_1), \cdots, f_m(X_m; X_1, \cdots, X_{m-1})$ defined by the recursive relation: $f_i(X_i; X_1, \cdots, X_{i-1}) = f_{i-1}(X_{i-1}; X_1, \cdots, X_{i-2})X_i, \forall i > 1$, and $f_1(X_1) = AX_1$. Now, since each f_i has a computation complexity of $O(N^3)$, LCC can be applied sequentially to compute $f_1, \cdots, f_m$, in this order, to obtain the evaluation of f. However, aside from the fact that it is unclear if this solution is applicable for any function f, the solution involves multiple rounds of communication (one for each low complexity function f_i) which imposes excessive overall delay in the distributed system, a problem that most of secure MPC algorithms suffer, and where LCC originally has advantage over them since it involves a single communication round.

In this paper, we propose a single-communication-round coding scheme that enables tuning how much computation every node can perform beforehand, and thus, providing more compatibility with different distributed systems, depending on their nodes' computation power. The main idea of the proposed coding scheme is to reduce the dimension of the encoded data points that are being transmitted to each computing node, then, each of the computing nodes evaluates the exact function of interest at the received encoded data point with reduced dimension leading to a lower computation cost per node. We show the idea of our solution through a toy example in the following section.

II. Motivating Example

Consider a distributed system that has 6 worker nodes, a polynomial function $f : \mathbb{F}^2 \to \mathbb{U}$ of degree one where $\mathbb{F}$ is a field, $\mathbb{U}$ is a vector space, and f is to be evaluated at two given points

$$X_1 = \begin{pmatrix} a_1 \\ b_1 \end{pmatrix} \text{ and } X_2 = \begin{pmatrix} a_2 \\ b_2 \end{pmatrix},$$

where $a_i, b_i \in \mathbb{F}$ for $i = 1, 2$. We aim to find a coding scheme that satisfies the following two conditions:

(a) Every transmitted vector from the master node has at most one non-zero entry.
(b) The set of evaluations $f(\mathcal{X}) = \{f(X_1), f(X_2)\}$ are recoverable once the fusion node receives the output of any 5 workers.

A solution to this problem can be as follows: For every worker node $i \in [6]$, define

$$\tilde{X}_i = \begin{pmatrix} \tilde{a}_i \\ \tilde{b}_i \end{pmatrix}$$

to be the encoded vector transmitted from the master node to this worker node where both $\tilde{a}_i, \tilde{b}_i \in \mathbb{F}$ and to be specified precisely later. Furthermore, we define two encoding functions $g_1, g_2 : \mathbb{F} \to \mathbb{F}$ as in (1), where $\alpha_1, \cdots, \alpha_6, \beta_1, \beta_2$ are unique elements in $\mathbb{F}$. Now, note the following regarding (1):

(i) $g_1(\alpha_i) = 0 \quad \forall i \in \{1, 3, 5\}$,
(ii) $g_2(\alpha_i) = 0 \quad \forall i \in \{2, 4, 6\}$,
(iii) $g_1(\beta_1) = a_1, g_1(\beta_2) = a_2$,
(iv) $g_2(\beta_1) = b_1, g_2(\beta_2) = b_2$.

We also define the mapping $g : \mathbb{F} \to \mathbb{F}^2$ as

$$g(\alpha) = \begin{pmatrix} g_1(\alpha) \\ g_2(\alpha) \end{pmatrix},$$

where $g_1(\alpha)$ and $g_2(\alpha)$ are as in (1). Now, for every $i \in [6]$, we let the encoded vector $\tilde{X}_i$ be $g(\alpha_i)$, i.e.,

$$\tilde{X}_i = g(\alpha_i) = \begin{pmatrix} g_1(\alpha_i) \\ g_2(\alpha_i) \end{pmatrix}. \quad (2)$$

We claim that the encoding in (2) satisfies both conditions (a) and (b). Indeed, recalling (i) and (ii), we conclude that, for any $i \in [6]$, either $g_1(\alpha_i) = 0$ or $g_2(\alpha_i) = 0$; hence, the first condition (sparsity condition) is satisfied. Moreover, for any $i \in [6]$, let $\{f(\tilde{X}_j)\}_{j\in[6]-\{i\}} = \{f(g(\alpha_j))\}_{j\in[6]-\{i\}}$ be the first five outputs received by the fusion node. Since f is of degree one in g, and g is of degree four in α, f is of degree four in α; hence, $f(g(\alpha))$ can be interpolated using the five outputs $\{f(g(\alpha_j))\}_{j\in[6]-\{i\}}$. Recalling (iii) and (iv), once $f(g(\alpha))$ is obtained, the fusion nodes evaluates

$$f(g(\beta_i)) = f\left(\begin{pmatrix} g_1(\beta_i) \\ g_2(\beta_i) \end{pmatrix}\right) = f\left(\begin{pmatrix} a_i \\ b_i \end{pmatrix}\right) = f(X_i), \quad i = 1, 2.$$

Thus, the second condition (decodability condition) is satisfied as well.

III. System Model and Problem Formulation

We consider a *computation system* that consists of three different types of nodes: (i) *a master node*; (ii) *worker nodes*; and (iii) *a fusion node* as depicted in Fig. 1.

Definition 3.1 (An (f, s, k, P, m) Computation System): An (f, s, k, P, m) computation system consists of the following:

(i) A *master node* that follows the following procedure: 1) receives m input vectors, i.e., the set of vectors $\mathcal{X} = \{X_1, \cdots, X_m\}$, where, for each $i \in [m]$, $X_i \in \mathbb{V}$, where $\mathbb{V}$ is a vector space with dimension d in $\mathbb{F}$ and $\mathbb{F}$ is some field and d is a positive integer, 2) computes $\tilde{X}_1, \cdots, \tilde{X}_P \in \mathbb{V}$, where $\tilde{X}_i = Enc_i(X_1, \cdots, X_m)$ such that each $\tilde{X}_i$ has at most s non-zero entries in $\mathbb{F}$, i.e., sparsity of each $\tilde{X}_i$ is at most s, and Enc_i is defined as a linear mapping $Enc_i : \mathbb{V} \times \cdots \times \mathbb{V} \to \mathbb{V}$, and 3) sends $\tilde{X}_i$ to worker node i, for every $i \in [P]$.

(ii) P *worker nodes* each of them owns the same polynomial function of interest $f : \mathbb{V} \to \mathbb{U}$, where $\mathbb{U}$ is a vector space, and performs the following procedure: 1) receives $\tilde{X}_i$ from the master node, 2) computes $f(\tilde{X}_i)$, and 3) sends the result $f(\tilde{X}_i)$ to the fusion node.

(iii) A *fusion node* that receives outputs from the successful worker nodes. If the number of successful workers is at least k, the fusion node performs decoding and produces the output $f(\mathcal{X}) = \{f(X_1), \cdots, f(X_m)\}$. That is, let $\mathcal{S} = \{s_1, \cdots, s_k\}$ be any set of k successful worker nodes, the fusion node computes $(f(X_1), \cdots, f(X_m)) = Dec_{\mathcal{S}}(\tilde{X}_{s_1}, \cdots, \tilde{X}_{s_k})$, where $Dec_{\mathcal{S}} : \mathbb{V} \times \cdots \times \mathbb{V} \to \mathbb{U}$ is a decoding function. Otherwise, i.e., if the number of successful workers is less than k, the fusion node declares a "computation failure."

An (f, s, k, P, m) computation system is one that consists of a master node, P worker nodes, and a fusion node that follow their above specified procedures, such that, when it receives $\mathcal{X} = \{X_1, \cdots, X_m\}$ as an input, it outputs $f(\mathcal{X}) = \{f(X_1), \cdots, f(X_m)\}$ if the number of successful workers is at least k.

We consider (f, s, k, P, m) computation systems where the computational complexities of the master node, each worker node, and the fusion node, when evaluated in terms of the parameters $\deg(f), d, s, P, m$, where $\deg(f)$ is the degree of f, are all less than the complexity of any sequential algorithm that takes $X_1, \cdots, X_m$ as input and computes $f(X_1), \cdots, f(X_m)$ as the output. Our goal in this paper is to characterize a class of (f, s, k, P, m) computation systems given f, s, P, m. That is, given f, s, P, m, we construct encoding and decoding functions $\{Enc_i\}_{i\in[P]}$, and $\{Dec_{\mathcal{S}}\}_{\mathcal{S}\subset[P],|\mathcal{S}|=k}$, respectively, and specify the computation system's $k(f, s, P, m) < P$ as a function of f, s, P, m.

IV. General Code Construction

In this section, we provide a general code construction solving the problem described in Section III. Our result can be formulated in the following theorem.

Theorem 4.1: For the coded computing problem under sparsity constraint described in Section III, a recovery threshold of

$$k = \left(\left\lceil \frac{P}{d} \right\rceil (d - s) + m - 1\right) \deg(f) + 1$$

is achievable.

Before proving the theorem, we first provide some useful definitions and then present the general code construction that achieves the recovery threshold in Theorem 4.1.

Let any data point $X_i, i \in [m]$ defined in Section III be written as $X_i = (x_{i1}, x_{i2}, \cdots, x_{id})$. Similarly, let any encoded vector $\tilde{X}_i, i \in [P]$ defined in Section III be written as $\tilde{X}_i = (\tilde{x}_{i1}, \tilde{x}_{i2}, \cdots, \tilde{x}_{id})$. Now, we define *the zero sets* $\mathcal{Z}_1, \cdots, \mathcal{Z}_P \subset [d]$. For, $i \in [P]$, *the zero set* $\mathcal{Z}_i$ is the set of indices of the encoded vector $\tilde{X}_i$ whose value is forced to zero by the encoding scheme to satisfy the sparsity constraint. Notice that, for any $i \in [P]$, $j \in \mathcal{Z}_i$ implies $\tilde{x}_{ij} = 0$; however, the converse is not necessarily true.

In this paper, we obtain the zero sets $\mathcal{Z}_i, i \in [P]$, in a similar way as introduced in ShortDot codes [3], i.e, based on the "circularly sliding train of zeros" of length $d - s$. Specifically, we let the zero set of $\tilde{X}_1$ be its first $d - s$ indices, i.e., $\mathcal{Z}_1 = \{1, 2, \cdots, d-s\}$, and then for the following zero sets of $\tilde{X}_2, \cdots, \tilde{X}_P$, we keep moving the zero set to right with step one and allowing circular shift. That is, for $i \in [P]$, $\mathcal{Z}_i = \{i \bmod d, i+1 \bmod d, \cdots, i+d-s-1 \bmod d\}$.

In addition to the zero sets, we also define the sets of zero-locators $\mathcal{R}_1, \cdots, \mathcal{R}_d \subset [P]$. For $j \in [d]$, *the zero-locator of index j set* (i.e., set $\mathcal{R}_j$) is the set of worker nodes such that the value of the j-th entry of the encoded vector received by any of these worker nodes is forced to zero due to the sparsity constraint, i.e., j belongs to the zero sets of all worker nodes in $\mathcal{R}_j$. Specifically, $\mathcal{R}_j = \{i : j \in \mathcal{Z}_i, i \in [P]\}$. Notice that if $\mathcal{Z}_1, \cdots, \mathcal{Z}_P$ are formed using the sliding train of zeros of length $d - s$, then we can conclude that $|\mathcal{R}_j| \leq \lceil \frac{P}{d} \rceil (d - s) \quad \forall j \in [d]$. Define unique $\beta_1, \cdots, \beta_m, \alpha_1, \cdots, \alpha_P \in \mathbb{F}$, and let $g(\alpha) = (g_1(\alpha), \cdots, g_d(\alpha))^T$, where

$$g_j(\alpha) = \sum_{k=1}^{m} x_{kj} \left(\prod_{r\in\mathcal{R}_j} \frac{\alpha - \alpha_r}{\beta_k - \alpha_r} \prod_{l\in[m]-k} \frac{\alpha - \beta_l}{\beta_k - \beta_l} \right), \quad (3)$$

$j \in [d]$. In the following, we describe the general coding procedure.

Construction 1 (Sparse LCC):

Master node (encoding): For all $i \in [P]$, the master node sends $\tilde{X}_i = (\tilde{x}_{i1}, \cdots, \tilde{x}_{id})$ to worker node i, where

$$\tilde{x}_{ij} = g_j(\alpha_i), \quad j \in [d]. \tag{4}$$

Worker nodes: For $i \in [P]$, worker node i computes $f(\tilde{X}_i)$ and sends the output to the fusion node.

Fusion node (decoding):

1) Waits till it collects the output of $\left(\lceil \frac{P}{d} \rceil (d-s) + m - 1\right) \deg(f) + 1$ worker nodes.
2) Interpolates $f(g(\alpha))$.
3) Evaluates $f(X_i) = f(g(\beta_i)) \quad \forall i \in [m]$.

Now, we prove Theorem 4.1.

Proof of Theorem 4.1 In order to prove the theorem, it suffices to show that Construction 1 is valid. That is, it suffices to show that Construction 1 satisfies both the sparsity and decodability conditions.

First, the sparsity condition is satisfied by recalling that $\tilde{x}_{ij} = g_j(\alpha_i)$, where $g_j(\alpha_i)$ is as defined in (3), and observing that $\{\alpha_i - \alpha_r\}_{r \in \mathcal{R}_j}$ is a subset of the factors of $g_j(\alpha_i)$ (i.e., $\tilde{x}_{ij}$). Therefore, $\tilde{x}_{ij}$ evaluates to zero whenever $i \in \mathcal{R}_j$.

For the decodability, notice that $f(g(\alpha))$ is of degree $\deg(f)$ in g and g is of degree at most $\lceil \frac{P}{d} \rceil (d-s) + m - 1$ in α; hence, $f(g(\alpha))$ is of degree at most $(\lceil \frac{P}{d} \rceil (d-s) + m - 1) \deg(f)$ in α. Since the output of the worker nodes can be seen as evaluations of the polynomial $f(g(\alpha))$ at $\alpha = \alpha_1, \cdots, \alpha_P$. Once the fusion node receives the output of $(\lceil \frac{P}{d} \rceil (d-s) + m - 1) \deg(f) + 1$ different worker nodes, it can interpolate the polynomial $f(g(\alpha))$. Moreover, notice from (3) that $g_j(\beta_i) = x_{ij} \quad \forall i \in [m], j \in [d]$. That is, $g(\beta_i) = X_i \quad \forall i \in [m]$ which directly implies that $f(X_i) = f(g(\beta_i)) \quad \forall i \in [m]$. □

A. Encoding and Decoding Complexities of Construction 1

1) Encoding Complexity: For each $i \in [P]$, computing $\tilde{X}_i$ requires computing a number of d evaluations $\tilde{x}_{i1}, \cdots, \tilde{x}_{id}$, where each of such evaluations is of complexity $O(\lceil \frac{P}{d} \rceil (d-s) + m - 1)$. Therefore, for any $i \in [P]$, the evaluation of each of $\tilde{X}_i$ has a computational complexity of $O(d(\lceil \frac{P}{d} \rceil (d-s) + m - 1))$. Therefore, the overall encoding complexity for P workers is $O(Pd(\lceil \frac{P}{d} \rceil (d-s) + m - 1))$.

2) Decoding Complexity: Noting that the co-domain of the polynomial $f(g(\alpha))$ is the vector space $\mathbb{U}$, let the dimension of $\mathbb{U}$ be $\dim(\mathbb{U})$. In the following, we compute the decoding complexity. First, the interpolation requires the inversion of a $k \times k$ Vandermonde matrix (considering the naive inverse approach for interpolation) with a complexity of $O(k^3)$. Then, the evaluation of each $f(g(\beta_i))$ requires a complexity of $O(k \dim(\mathbb{U}))$. Therefore, the evaluation of all $f(g(\beta_i)), i \in [m]$ requires $O(mk \dim(\mathbb{U}))$ operations. Thus, the whole decoding complexity is $O(k^3 + mk \dim(\mathbb{U}))$.

V. Private Lagrange Coded Computing under Sparsity Constraints

In this section, we aim to provide fault-tolerant codes for function computations in distributed systems under a sparsity constraint, and, in addition, we aim to also provide privacy against colluding computing nodes. Specifically, we consider a *semi-honest* model for the colluding computing nodes, where they correctly perform their designated computations; however, they may cooperate in order to gain some knowledge on the private input data. Before we formally state the problem tackled in this section, we provide the following definition borrowed from [16].

Definition 5.1 ($t-$private Coding Schemes): A coding scheme is $t-$private if any set of at most t computing nodes cannot eventually compute more than they could jointly compute solely from their set of inputs and outputs.

A. Problem Formulation

We consider the same problem stated in Section III with the restriction that, here, we assume that the field $\mathbb{F}$ is a finite field. In addition, we impose a privacy constraint that requires a $t-$private coding scheme solution. Specifically, we require that at most any t worker nodes cannot jointly compute the data set $\{X_1, \cdots, X_m\}$ given only their set of inputs and outputs.

B. A $t-$private Code Construction

We state our result in the following theorem.

Theorem 5.1: For the private coded computing problem under sparsity constraint described in Section V-A, a recovery threshold of

$$k = \left(\left\lceil \frac{P}{d} \right\rceil (d-s) + m + t - 1 \right) \deg(f) + 1$$

is achievable.

Before we provide a proof sketch to Theorem 5.1, in the following, we provide a t-private code construction denoted by *t-private Sparse LCC* that achieves the recovery threshold in Theorem 5.1. Let any data point $X_i, i \in [m]$ be written as $X_i = (x_{i1}, x_{i2}, \cdots, x_{id})$. Similarly, we let any encoded vector $\tilde{X}_i, i \in [P]$ be written as $\tilde{X}_i = (\tilde{x}_{i1}, \tilde{x}_{i2}, \cdots, \tilde{x}_{id})$, and define unique $\beta_1, \cdots, \beta_{m+t}, \alpha_1, \cdots, \alpha_P \in \mathbb{F}$, and let $\tilde{g}(\alpha) = (\tilde{g}_1(\alpha), \cdots, \tilde{g}_d(\alpha))$, where

$$\tilde{g}_j(\alpha) = \sum_{k=1}^{m} x_{kj} \left(\prod_{r \in \mathcal{R}_j} \frac{\alpha - \alpha_r}{\beta_k - \alpha_r} \prod_{l \in [m+t]-k} \frac{\alpha - \beta_l}{\beta_k - \beta_l} \right) + \sum_{k'=m+1}^{m+t} z_{(k'-m)j} \left(\prod_{r \in \mathcal{R}_j} \frac{\alpha - \alpha_r}{\beta_{k'} - \alpha_r} \prod_{l \in [m+t]-k'} \frac{\alpha - \beta_l}{\beta_{k'} - \beta_l} \right), \tag{5}$$

$j \in [d]$, where z_{ij} for $i \in [t]$ and $j \in [d]$ are independently and randomly selected from the finite field $\mathbb{F}$, and for $j \in [d]$, $\mathcal{R}_j$ is defined as in Construction 1, i.e., $\mathcal{R}_j = \{i : j \in \mathcal{Z}_i, i \in [P]\}$ where $\mathcal{Z}_i = \{i \bmod d, i+1 \bmod d, \cdots, i+d-s-1 \bmod d\}$. In the following, we describe the $t-$private Sparse LCC scheme.

Construction 2 ($t-$private Sparse LCC):

Master node (encoding): For all $i \in [P]$, the master node sends $\tilde{X}_i = (\tilde{x}_{i1}, \cdots, \tilde{x}_{id})$ to worker node i, where

$$\tilde{x}_{ij} = \tilde{g}_j(\alpha_i), \quad j \in [d].$$

Worker nodes: For $i \in [P]$, worker node i computes $f(\tilde{X}_i)$ and sends the output to the fusion node.

Fusion node (decoding):

1) Waits till it collects the output of $\big(\lceil \frac{P}{d} \rceil (d-s) + m + t - 1\big) \deg(f) + 1$ worker nodes.
2) Interpolates $f(\tilde{g}(\alpha))$.
3) Evaluates $f(X_i) = f(\tilde{g}(\beta_i)) \quad \forall i \in [m]$.

Proof Sketch of Theorem 5.1 In order to prove the theorem, it suffices to show that Construction 2 is valid. That is, it suffices to show that it satisfies the sparsity, decodability, and privacy conditions.

First, the sparsity condition is satisfied by recalling that, in Construction 2, $\tilde{x}_{ij} = \tilde{g}_j(\alpha_i)$, where $\tilde{g}_j(\alpha_i)$ is as defined in (5), and observing that $\{\alpha_i - \alpha_r\}_{r \in \mathcal{R}_j}$ is a subset of the factors of $\tilde{g}_j(\alpha_i)$ (i.e., $\tilde{x}_{ij}$). Therefore, $\tilde{x}_{ij}$ evaluates to zero whenever $i \in \mathcal{R}_j$.

For the decodability, notice that $f(\tilde{g}(\alpha))$ is of degree $\deg(f)$ in $\tilde{g}$ and $\tilde{g}$ is of degree at most $\lceil \frac{P}{d} \rceil (d-s) + m + t - 1$ in α; hence, $f(\tilde{g}(\alpha))$ is of degree at most $(\lceil \frac{P}{d} \rceil (d-s) + m + t - 1) \deg(f)$ in α. Since the output of the worker nodes can be seen as evaluations of the polynomial $f(\tilde{g}(\alpha))$ at distinct $\alpha_1, \cdots, \alpha_P$. Once the fusion node receives the output of any $(\lceil \frac{P}{d} \rceil (d-s) + m + t - 1) \deg(f) + 1$ different worker nodes, it can interpolate the polynomial $f(\tilde{g}(\alpha))$. Moreover, notice from (5) that $\tilde{g}_j(\beta_i) = x_{ij} \quad \forall i \in [m], j \in [d]$. That is, $\tilde{g}(\beta_i) = X_i \quad \forall i \in [m]$ which directly implies that $f(X_i) = f(\tilde{g}(\beta_i)) \quad \forall i \in [m]$.

Lastly, the privacy condition in Construction 2 is implied by the fact that each of the worker nodes receives an evaluation of $\tilde{g}(\alpha)$ in which the linearly combined private inputs are mixed with a linear combination of t independent and randomly selected values from $\mathbb{F}$ such that no t evaluations of $\tilde{g}(\alpha)$ can recover the linearly combined encoded inputs. □

VI. Discussion: The t-private Sparse LCC as a Unifying Coding Scheme

The t-private Sparse LCC scheme introduced in this paper can be considered as a unifying coding scheme that includes LCC codes [17] and ShortDot codes [3] as special cases. Specifically, the Sparse LCC scheme reduces to the LCC scheme if $s = d$, and reduces to ShortDot codes if the function of interest f is defined as $f : \mathbb{F}^{N \times 1} \to \mathbb{F}$ such that $f(X) = C^T X$, for some constant $C \in \mathbb{F}^{N \times 1}$, and the data set on which f to be evaluated is $\mathcal{X} = \{X_1, \cdots, X_M\}$, where $N \gg M$.

In addition, it is worth mentioning how $t-$private Sparse LCC differs from the work in [18]. In [18], a sharing scheme denoted by *Polynomial Sharing* is proposed to privately compute an arbitrary polynomial function of interest under a communication constraint from the source nodes to the worker nodes. The main difference between the Polynomial Sharing scheme and the t-private Sparse LCC is that the Polynomial Sharing is constructed based on defining private schemes over basic matrix operations (e.g., addition and multiplication of two matrices), and then extending to general polynomial functions (with arbitrary degree) by recursively applying the private scheme for the multiplication (or addition) of two matrices. This requires applying multiple communication round to privately compute the polynomial function of interest. However, in the t-private Sparse LCC, only one communication round is needed to compute any polynomial function of interest. However, the t-private Sparse LCC scheme requires more number of worker nodes P compared to the Polynomial Sharing scheme for the same t privacy guarantee.

References

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

[2] 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, 2018.

[3] 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 (NIPS)*, 2016, pp. 2092–2100.

[4] Q. Yu, M. A. Maddah-Ali, and A. S. Avestimehr, "Polynomial Codes: an Optimal Design for High-Dimensional Coded Matrix Multiplication," in *Advances In Neural Information Processing Systems (NIPS)*, 2017, pp. 4403–4413.

[5] M. Fahim, H. Jeong, F. Haddadpour, S. Dutta, V. Cadambe, and P. Grover, "On the optimal recovery threshold of coded matrix multiplication," in *Communication, Control, and Computing (Allerton)*, Oct 2017, pp. 1264–1270, extended version at http://arxiv.org/abs/1801.10292.

[6] S. Dutta, M. Fahim, F. Haddadpour, H. Jeong, V. Cadambe, and P. Grover, "On the optimal recovery threshold of coded matrix multiplication," *IEEE Transactions on Information Theory*, pp. 1–1, 2019.

[7] M. Fahim and V. R. Cadambe, "Numerically stable polynomially coded computing," in *2019 IEEE International Symposium on Information Theory (ISIT)*, July 2019, pp. 3017–3021, extended version at https://arxiv.org/abs/1903.08326.

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

[9] Q. Yu, M. A. Maddah-Ali, and A. S. Avestimehr, "Straggler mitigation in distributed matrix multiplication: Fundamental limits and optimal coding," *arXiv preprint arXiv:1801.07487*, 2018.

[10] S. Dutta, Z. Bai, H. Jeong, T. M. Low, and P. Grover, "A unified coded deep neural network training strategy based on generalized polydot codes," in *2018 IEEE International Symposium on Information Theory (ISIT)*, June 2018, pp. 1585–1589, http://arxiv.org/abs/1811.10 751.

[11] F. Haddadpour, Y. Yang, M. Chaudhari, V. R. Cadambe, and P. Grover, "Straggler-resilient and communication-efficient distributed iterative linear solver," *CoRR*, vol. abs/1806.06140, 2018. [Online]. Available: http://arxiv.org/abs/1806.06140

[12] S. Li, S. M. M. Kalan, Q. Yu, M. Soltanolkotabi, and A. S. Avestimehr, "Polynomially coded regression: Optimal straggler mitigation via data encoding," *preprint arXiv:1805.09934*, 2018.

[13] R. Tandon, Q. Lei, A. G. Dimakis, and N. Karampatziakis, "Gradient Coding: Avoiding Stragglers in Distributed Learning," in *International Conference on Machine Learning (ICML)*, 2017, pp. 3368–3376.

[14] M. Ye and E. Abbe, "Communication-computation efficient gradient coding," in *Proceedings of the 35th International Conference on Machine Learning, ICML, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, 2018, pp. 5606–5615. [Online]. Available: http://proceedings.mlr.press/v80/ye18a.html

[15] L. Chen, H. Wang, Z. Charles, and D. Papailiopoulos, "Draco: Byzantine-resilient distributed training via redundant gradients," in *International Conference on Machine Learning*, 2018, pp. 903–912.
[16] M. Ben-Or, S. Goldwasser, and A. Wigderson, "Completeness theorems for non-cryptographic fault-tolerant distributed computation," in *Proceedings of the twentieth annual ACM symposium on Theory of computing*. ACM, 1988, pp. 1–10.
[17] Q. Yu, N. Raviv, J. So, and A. S. Avestimehr, "Lagrange coded computing: Optimal design for resiliency, security and privacy," *arXiv preprint arXiv:1806.00939*, 2018.
[18] H. A. Nodehi and M. A. Maddah-Ali, "Secure coded multi-party computation for massive matrix operations," *arXiv preprint arXiv:1908.04255*, 2019.