Download and Access Trade-offs in Lagrange Coded Computing

Netanel Raviv*, **Qian Yu**†, **Jehoshua Bruck***, and **Salman Avestimehr**†
*Department of Electrical Engineering, California Institute of Technology, Pasadena 91125, CA, USA
†Department of Electrical Engineering, University of Southern California, Los Angeles 90089, CA, USA

Abstract—**Lagrange Coded Computing (LCC) is a recently proposed technique for resilient, secure, and private computation of arbitrary polynomials in distributed environments. By mapping such computations to composition of polynomials, LCC allows the master node to complete the computation by accessing a minimal number of workers and downloading all of their content, thus providing resiliency to the remaining stragglers. However, in the most common case in which the number of stragglers is less than in the worst case scenario, much of the computational power of the system remains unexploited. To amend this issue, in this paper we expand LCC by studying a fundamental trade-off between download and access, and present two contributions. In the first contribution, it is shown that without any modification to the encoding process, the master can decode the computations by accessing a larger number of nodes, however downloading less information from each node in comparison with LCC (i.e., trading access for download). This scheme relies on decoding a particular polynomial in the ideal that is generated by the polynomials of interest, a technique we call *Ideal Decoding*. This new scheme also improves LCC in the sense that for systems with adversaries, the overall downloaded bandwidth is *smaller* than in LCC. In the second contribution we study a real-time model of this trade-off, in which the data from the workers is downloaded sequentially. By clustering nodes of similar delays and encoding the function with *Universally Decodable Matrices*, the master can decode once sufficient data is downloaded from every cluster, regardless of the internal delays within that cluster. This allows the master to utilize the partial work that is done by stragglers, rather than to ignore it, a feature that most past works in coded computing are lacking.**

I. Introduction

The immensity of contemporary datasets no longer allows computations to be done on a single machine, and distributed computations are inevitable. Since most users cannot afford to maintain a network of servers (or workers), burdensome computations are often outsourced to third party cloud services. However, this approach opens a Pandora's box of resiliency, security, and privacy issues. First, it was demonstrated in the past (e.g. [16]) that a fraction of the servers, referred to as *stragglers*, can be 5 to 8 times slower than the average, and hence computation tasks that rely on successful completion of all subtasks are destined to be delayed considerably. Second, many computations are highly susceptible to *adversaries*, or *Byzantine workers*, that might attempt to alter the result of the computation for their benefit [2]. Third, privacy infringement is major concern in the information age, and hence privacy-preserving computation protocols are essential.

The term *Coded Computing* broadly refers to a family of techniques that utilize coding to inject computation redundancy in order to alleviate the various issues that arise in distributed computations. Over the past few years, Coded Computing has seen a tremendous success in providing elegant solutions to the aforementioned issues in various tasks of interest, such as gradient coding (e.g., [5], [6], [10]), matrix multiplication (e.g., [3], [4], [17]), and bandwidth reduction in iterative algorithms (e.g., [7]). More recently, *Lagrange Coded Computing* (LCC) has been proposed in [18] as a *universal* data encoding technique that can simultaneously alleviate the issues of resiliency, security, and privacy for arbitrary multivariate polynomial computations, hence expanding coded computing to new domains.

In LCC, the dataset is encoded by evaluations of the well-known Lagrange polynomial, and each codeword symbol is stored on a different worker in the distributed system. Then, the workers apply the multivariate polynomial of interest on their encoded data, as if no coding is taking place, and return the computation results back to the master. By viewing the computation as a composition of a multivariate polynomial (the computation that is to be executed), and a univariate one (the encoding Lagrange polynomial), the task of finalizing the computation in the presence of stragglers and adversaries boils down to polynomial interpolation with errors and erasures. Then, the master finalizes the computation by evaluating the interpolated polynomial at appropriately chosen points. Being fundamental to our current contribution, the LCC scheme is described in greater detail in Subsection II-A.

However, LCC allows no flexibility in terms of download-access trade-off. That is, the master performs the computation by accessing a minimum number of workers, and downloading their data in its entirety. As a result, in every scenario with less than the maximum number of stragglers, some non-stragglers remain idle during the download process, and the communication bottleneck intensifies due to unexploited parallel links between these non-stragglers and the master. Moreover, LCC considers every worker as being either a straggler or a non-straggler, and the partial work that is done by stragglers is ignored. In this paper we improve LCC by addressing these aspects, the static and the dynamic, of the download-access trade-off.

In our first contribution, it is shown that with no further changes in the encoding phase, the decoding phase can be flexible in terms of the number of workers that are accessed, and the number of symbols that are downloaded from each of them. This is done by having the server perform extra linear computations; these computation turn multiple low-degree polynomial evaluations (the computation results) to a smaller number of high-degree polynomial evaluations. These high-degree polynomials lie in the ideal that is generated by the lower-degree ones, and hence we term this technique *Ideal Decoding*. More importantly, the surprising corollary of this part of the paper is that the overall download bandwidth can be *reduced* for systems with adversaries, when compared to ordinary LCC.

In our second contribution, we consider a dynamic model of the access-download trade-off, where the master has continuous access to all servers, and the data arrives sequentially. By encoding *the polynomial itself* with *Universally Decodable Matrices* (UDMs), a previously defined notion, we match the amount of download from each server to the naturally occurring delays in the system. Specifically, we cluster the workers in the system according to the expected computation times, and have workers in the same cluster operate on the same encoded data. By allowing the functions that are applied in each cluster to differ, it is shown that the decoding can be completed once sufficient information has arrived from each cluster, *regardless* of the internal delays within that cluster.

This paper is structured as follows. Preliminary background on LCC and UDMs is given in Section II, and our contributions are formally stated in Section III. The former extension of LCC is given in Section IV, and the latter in Section V.

II. Preliminaries

We use the standard notation $[N]$ for the set $\{1,\ldots,N\}$, denote the underlying field[1] by $\mathbb{F}$, and denote the composition operation between polynomials by $\circ$.

We consider a system with a master node and N workers, in which a dataset $\mathbf{X}=(\mathbf{x}_1,\ldots,\mathbf{x}_K)$, with $\mathbf{x}_k\in\mathbb{F}^{M\times 1}$ for every $k\in[K]$, is coded as $\tilde{\mathbf{x}}_1,\ldots,\tilde{\mathbf{x}}_N$, and each *codeword symbol* $\tilde{\mathbf{x}}_n$ is stored in one of the N workers. The master node is interested in the computation results $\{\mathbf{y}_k\triangleq f(\mathbf{x}_k)\}_{k\in[K]}$ for a polynomial $f=(f_1,\ldots,f_L)$, where $f_\ell:\mathbb{F}^M\to\mathbb{F}$ for all $\ell\in[L]$ and $G\triangleq\max\{\deg(f_\ell)\}_{\ell\in[L]}$. To achieve this, f is applied by the workers on their stored data, and the results of the computation $\{\tilde{\mathbf{y}}_n=f(\tilde{\mathbf{x}}_n)\}_{n\in[N]}$ on the codeword symbols are transmitted back to the master. Many tasks which are studied in coded computation fall under this framework, including matrix multiplication, and gradient coding whenever the loss function is a polynomial, or is approximated by one.

For integers A and S, a coding scheme is said to be S-resilient and A-secure if the master is capable of extracting $\{\mathbf{y}_k\}_{k\in[K]}$, even if up to S workers fail to respond in a timely manner, and up to A workers reply with purposefully erroneous data. In addition, for an integer T the scheme is called T-private if every set of T colluding workers remain information-theoretically oblivious to the content of $\mathbf{X}$, i.e., if $I(\{\tilde{\mathbf{x}}_t\}_{t\in\mathcal{T}};\mathbf{X})=0$ for every $\mathcal{T}\subseteq[N]$ of size at most T, where I denotes mutual information, and $\mathbf{X}$ is seen as chosen uniformly at random.

In Section V, it is further assumed that the results of the computation on the coded data $\tilde{\mathbf{y}}_n=f(\tilde{\mathbf{x}}_n)=(f_1(\tilde{\mathbf{x}}_n),\ldots,f_L(\tilde{\mathbf{x}}_n))$, from every worker $n\in[N]$, arrive at the master *sequentially*. That is, $f_1(\tilde{\mathbf{x}}_n)$ arrives, followed by $f_2(\tilde{\mathbf{x}}_n)$, and so on. In addition, we allow the polynomial f itself to be coded, and the encoding can potentially differ from one worker to another. That is, each worker $n\in[N]$ corresponds to L polynomials $h_{n,1},\ldots,h_{n,L}$, each of which is a linear combination of the polynomials $\{f_\ell\}_{\ell\in[L]}$.

A. Lagrange Coded Computing

Lagrange Coded Computing [18] follows the outline that is described above, and achieves resiliency, security, and privacy that is known to be optimal in many cases. LCC relies heavily on the Lagrange polynomial, as follows.

Given the data matrix $\mathbf{X}$, fix K distinct elements $\boldsymbol{\beta}=(\beta_1,\ldots,\beta_K)$ and additional N distinct elements $\boldsymbol{\alpha}=(\alpha_1,\ldots,\alpha_N)$ in $\mathbb{F}$. By using the well-known Lagrange interpolation formula, define $u=u_{\mathbf{X},\boldsymbol{\beta}}$ as the lowest degree polynomial such that $u(\beta_k)=\mathbf{x}_k$ for every $k\in[K]$. Notice that u is in fact a vector of polynomials, but we refer to it as a polynomial for simplicity. It is well known that the degree of u (or more precisely, the degree of every component of u) is at most $K-1$. Then, in the storage phase, the polynomial u is evaluated at α_n and the evaluation is sent to worker n, i.e., $\tilde{\mathbf{x}}_n=u(\alpha_n)$ for every $n\in[N]$.

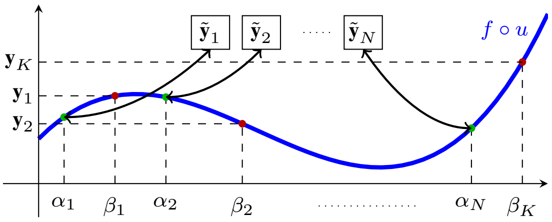

Fig. 1. Illustration of LCC. Following the computation, worker $n\in[N]$ holds $\tilde{\mathbf{y}}_n$, which is an evaluation of $f\circ u$ at α_n. The results of the computation $\{\mathbf{y}_k\}_{k\in[K]}$ are obtained by interpolating $f\circ u$ from $\{\tilde{\mathbf{y}}_n\}_{n\in[N]}$, and evaluating it at $\beta_1,\ldots,\beta_K$.

In the computation phase, every worker applies the polynomial f on its stored data, and sends the results back to the master. Since $\tilde{\mathbf{x}}_n=u(\alpha_n)$, it follows that $f(\tilde{\mathbf{x}}_n)$ is an evaluation at α_n of the *univariate* polynomial $f\circ u$, whose degree is at most $G(K-1)$. Hence, since $u(\beta_k)=\mathbf{x}_k$ for every $k\in[K]$, it follows that the results of the computation $\{\mathbf{y}_k\}_{k\in[K]}$ can be obtained by decoding the coefficients of $f\circ u$, and evaluating it at $\beta_1,\ldots,\beta_K$ (see Figure 1).

Moreover, whenever there exists a privacy requirement (i.e., when $T>0$ and $\mathbb{F}$ is finite), the data matrix $\mathbf{X}$ is padded with T random entries $\mathbf{T}=(\mathbf{t}_1,\ldots,\mathbf{t}_T)\in\mathbb{F}^{M\times T}$. The user fixes $\boldsymbol{\beta}=(\beta_1,\ldots,\beta_{K+T})$ and $\boldsymbol{\alpha}$ such that $\{\beta_k\}_{k\in[K]}\cap\{\alpha_n\}_{n\in[N]}=\varnothing$, and defines $u=u_{\mathbf{X}|\mathbf{T},\boldsymbol{\beta}}$ as the unique polynomial such that $u(\beta_k)=\mathbf{x}_k$ for every $k\in[K]$ and $u(\beta_{K+t})=\mathbf{t}_t$ for every $t\in[T]$. Then, the encoding is performed by evaluating u at the points of $\boldsymbol{\alpha}$, and we have the following theorem.

Theorem 1. *[18] Lagrange Coded Computing provides an S-resilient, A-secure, and T-private scheme for computing $\{\mathbf{y}_k\}_{k\in[K]}$ for any polynomial f, as long as $N\geq(K+T-1)G+S+2A+1$.*

Remark 1. *LCC has additional applications in obtaining another aspect of information-theoretic privacy. In the so-called* function-privacy, *the identity of the polynomial f should be kept private from sets of colluding servers. This problem, that is also known as* Private Computation *[13], is a generalization of the well-studied* Private Information Retrieval *problem, and is studied in [9].*

B. Universally Decodable Matrices

Universally Decodable Matrices (UDMs) have been studied in the past for various applications, such as slow-fading channels [14], and decoding of scalar codes in the presence of stragglers [8]. They are tightly connected to various previously defined notions, such as m-codes [11], and their corresponding metric was thoroughly studied in [1].

Definition 1. *For integers L and P, matrices $\boldsymbol{B}_1,\ldots,\boldsymbol{B}_P\in\mathbb{F}^{L\times L}$ are called UDMs if for every nonnegative integers $n_1,\ldots,n_P$ that sum to L, the following matrix is invertible–*

$$(\boldsymbol{b}_{1,1},\ldots,\boldsymbol{b}_{1,n_1},\boldsymbol{b}_{2,1},\ldots,\boldsymbol{b}_{2,n_2},\ldots,\boldsymbol{b}_{P,1},\ldots,\boldsymbol{b}_{P,n_P}),\quad(1)$$

where $\boldsymbol{b}_{i,j}$ is the j'th column of $\boldsymbol{B}_i$, indexed from left to right.

[1] Our techniques operate over any large enough finite field or any infinite one.

For example, the following matrices are UDMs for $L = 4$, $P = 3$ and $\mathbb{F} = GF(2)$.

$$\mathbf{B}_1 = \mathbf{I}, \quad \mathbf{B}_2 = \mathbf{J}, \quad \mathbf{B}_3 = \begin{pmatrix} 1 & & & \\ 1 & 1 & & \\ 1 & & 1 & \\ 1 & 1 & 1 & 1 \end{pmatrix},$$

where $\mathbf{I}$ is the identity matrix, and $\mathbf{J}$ is a matrix whose anti-diagonal is all 1's, and the remaining entries are 0. We focus our attention on the following construction of UDMs, that requires $|\mathbb{F}| \geq P - 1$.

Theorem 2. *[15, Prop. 14] For positive integers P and L, and a primitive[2] element $\alpha \in \mathbb{F}$, the matrices $\boldsymbol{B}_1 = \boldsymbol{I}$, $\boldsymbol{B}_2 = \boldsymbol{J}$, and $\boldsymbol{B}_3, \ldots, \boldsymbol{B}_P$ are UDMs, where $(\boldsymbol{B}_{p+2})_{k,n} = \binom{k}{n}\alpha^{p(k-n)}$ for $(p, n, k) \in [P-2] \times [L] \times [L]$.*

A crucial ingredient of the proof of Theorem 2 is the following proposition, which utilizes the notion of *Hasse derivative*. For a nonnegative integer n, the n'th Hasse derivative of a polynomial $\zeta(x) \triangleq \sum_{i=0}^{L-1} z_{i+1}x^i$ is $\zeta^{(n)}(x) = \sum_{i=0}^{L-1} \binom{i}{n} z_{i+1}x^{i-n}$ (for $i < n$ we have $\binom{i}{n} = 0$). Note that $\zeta(x)$ has a zero of multiplicity m at a point $\gamma \in \mathbb{F}$ if and only if $\zeta^{(n)}(\gamma) = 0$ for every $0 \leq n \leq m$, and $\zeta^{(m)}(\gamma) \neq 0$ [15, Lemma 13]. In addition, we define $\zeta^{(n)}(\infty) = z_{L-n}$.

Proposition 1. *Let $\gamma_1 = 0, \gamma_2 = \infty$, and $\gamma_p = \alpha^p$ for $p \in \{3, \ldots, P\}$. For a polynomial $\zeta(x) = \sum_{i=0}^{L-1} z_{i+1}x^i$ and integers $p \in [P]$ and $\ell \in [L]$, we have that $(z_1, \ldots, z_L) \cdot \boldsymbol{b}_{p,\ell} = \zeta^{(\ell-1)}(\gamma_p)$.*

Intuitively, to prove Theorem 2, one shows that if $(z_1, \ldots, z_L) \cdot \mathbf{B}' = 0$, where $\mathbf{B}'$ is the matrix in (1), then the respective polynomial $\zeta(x) = \sum_{i=0}^{L-1} z_{i+1}x^i$ must have at least L zeros (including multiplicities). This follows easily from Proposition 1 and the properties of the Hasse derivative.

III. Our Contribution

In order to successfully interpolate the polynomials $\{f_\ell \circ u\}_{\ell \in [L]}$ in LCC (Subsection II-A), the user must download the results $f_1(\tilde{\mathbf{x}}_n), \ldots, f_L(\tilde{\mathbf{x}}_n)$ from at least $(K+T-1)G+2A+1$ workers $n \in [N]$. In Section IV, we provide a scheme which enables to complete the computation on Lagrange encoded data in many other points of the download-access trade-off. In what follows, we let $H \triangleq G(K+T-1)+1$.

Theorem 3. *In Lagrange Coded Computing, it is possible to complete the computation by downloading L/R symbols from any set of $RH + 2A$ workers, for every rational $R = \frac{R_e}{R_d} > 1$ such that $R_e|L$, $R_d|H$, and $N \geq RH + 2A$.*

It will be clear in the sequel that the requirements $R_e|L$ and $R_d|H$ are mere convenience, and can be alleviated at the price of rounding operations. Theorem 3 is proved by using a technique we term *Ideal Decoding*. In this technique, every server $n \in [N]$ linearly combines the results $\{\tilde{y}_{n,\ell}\}_{\ell \in [L]}$, together with powers of α_n, to produce evaluations of certain polynomials $\{g_i\}_{i \in [L/R]}$, which lie in the ideal which is generated by $\{f_\ell \circ u\}_{\ell \in [L]}$ in the ring of univariate polynomials over $\mathbb{F}$. These g_i's are interpolated by the master from their evaluations, and the original $\{f_\ell \circ u\}_{\ell \in [L]}$ are obtained by computing some polynomial combinations of the g_i's. We emphasize that this final polynomial computation can be done by a combination of shifts, additions, and negations of field elements, and does not require polynomial multiplications. Having obtained $\{f_\ell \circ u\}_{\ell \in [L]}$, the master finalizes the computation as in ordinary LCC.

A surprising corollary of Theorem 3 is that for systems with adversaries, the overall download of our suggested scheme outperforms that of ordinary LCC (see Remark 2).

While the scheme of Theorem 3 enables the user to download fewer symbols from every worker than in ordinary LCC, computing these symbols requires the computation of all functions f_i on the coded data. Furthermore, the reduction factor R must be known a priori, and hence, this scheme is not suitable to handle run-time delays in the system.

To amend these issues, in the second part of this paper (Section V), we consider systems in which the workers are arranged in *clusters*. Then, the data is encoded by an LCC scheme whose code length is the number of clusters (rather than the number of workers), and all servers in a cluster store the same codeword symbol (we refer to such systems as *clustered LCC*). By encoding f with UDMs, it is shown that the computation can be completed by downloading L elements from each cluster, *regardless* of their exact source within the cluster. This scheme enables stronger stragglers tolerance, in the sense that it exploits the partial work that is done by the stragglers, in a way that can accommodate any possible combination of delays within each cluster.

Theorem 4. *In clustered LCC, it is possible to complete the computation by downloading L sequentially arriving symbols from each one of $H + 2A$ clusters.*

Further, in cases where the number of adversaries *per cluster* is known, we have the following.

Theorem 5. *In clustered LCC with at most A_i adversaries in each cluster i, it is possible to complete the computation by downloading $(2A_i + 1)L$ sequentially arriving symbols from cluster i for each one of H clusters.*

IV. Achieving a download-access trade-off

In this section we prove Theorem 3 by introducing extra linear computations at the workers. Let $R > 1$ be the required reduction factor, which is known to all workers, and for now assume that it is an integer (fractional reduction factors will be treated in the sequel). In addition, for every $\ell \in [L]$ denote $f_\ell \circ u$ by r_ℓ. Following the storage phase and the computation phase, every server $n \in [N]$ contains $\{\tilde{y}_{n,\ell}\}_{\ell \in [L]}$, and computes

$$\begin{pmatrix} \sum_{i=1}^{R} \alpha_n^{(i-1)H} \tilde{y}_{n,i} \\ \sum_{i=1}^{R} \alpha_n^{(i-1)H} \tilde{y}_{n,R+i} \\ \vdots \\ \sum_{i=1}^{R} \alpha_n^{(i-1)H} \tilde{y}_{n,L-R+i} \end{pmatrix} \triangleq \begin{pmatrix} g_1(\alpha_n) \\ g_2(\alpha_n) \\ \vdots \\ g_{L/R}(\alpha_n) \end{pmatrix}.$$

Since $\tilde{y}_{n,\ell} = r_\ell(\alpha_n)$ and $\deg(r_\ell) \leq H-1$ for every $\ell \in [L]$, it follows that each server $n \in [N]$ now holds L/R evaluations at α_n of the polynomials $\{g_i\}_{i=1}^{L/R}$, each of which is of degree at most $RH - 1$. Hence, having received the responses from any set of at least $RH + 2A$ servers, the user is able to obtain the coefficients of all g_i's by Reed-Solomon decoding. Now, it is readily verified that for every $i \in [L/R]$ the first H coefficients of g_i coincide with those of $r_{(i-1)R+1}$, the next H with those of $r_{(i-1)R+2}$, and so on. Hence, all the polynomials $\{r_\ell\}_{\ell \in [L]}$ can be found, and the scheme is finalized by evaluating them at $\beta_1, \ldots, \beta_K$.

[2] If $\mathbb{F}$ is infinite, α should be any element whose order is at least $P - 1$. If $\mathbb{F}$ is finite, the binomial coefficient should be computed over the integers, and then naturally embedded in $\mathbb{F}$.

It is apparent from the simple case of an integer R that the gist of the extra linear computations by the workers is to obtain polynomial evaluations of some higher degree g_i's in the ideal which is generated by the r_i's. Then, after obtaining the coefficients of the g_i's, the coefficients of the r_i's can be trivially extracted.

Now, let R be fractional. In this case, one must choose different polynomials $\{g_i\}_{i\in[L/R]}$ judiciously, so that this extraction is still possible. In what follows, the polynomials g_i are defined anew so that some overlap exists between the coefficients of the r_i's in them. Then, after the interpolation by the master, this overlap is resolved by performing a polynomial combination of the g_i's. We begin with an illustrative example.

Example 1. *Let $H = 4$, $R = 11/4$, $A = 0$, and $L = 22$. For every $n \in [N]$, the n'th server computes*

$$\begin{aligned} g_{1,n} &\triangleq \tilde{y}_{n,1} + \alpha_n^4 \tilde{y}_{n,2} + \alpha_n^7 \tilde{y}_{n,3} \\ g_{2,n} &\triangleq \tilde{y}_{n,3} + \alpha_n \tilde{y}_{n,4} + \alpha_n^5 \tilde{y}_{n,5} + \alpha_n^7 \tilde{y}_{n,6} \\ g_{3,n} &\triangleq \tilde{y}_{n,6} + \alpha_n^2 \tilde{y}_{n,7} + \alpha_n^6 \tilde{y}_{n,8} + \alpha_n^7 \tilde{y}_{n,9} \\ g_{4,n} &\triangleq \tilde{y}_{n,9} + \alpha_n^3 \tilde{y}_{n,10} + \alpha_n^7 \tilde{y}_{n,11}. \end{aligned}$$

It is readily verified that for every i and n, the value $g_{i,n}$ is an evaluation at α_n of a polynomial g_i, whose degree is at most 10. Hence, all g_i's can be extracted from $RH = 11$ workers. Then, the master computes

$$\sum_{j=0}^{3}(-1)^j x^{7j} g_{j+1} = r_1 + x^4 r_2 - x^8 r_4 - x^{12} r_5 + x^{16} r_7 \\ + x^{20} r_8 - x^{24} r_{10} - x^{28} r_{11}.$$

Since $\deg(r_i) \leq 3$ for all i, the coefficients of r_1, r_2, r_4, r_5, r_7, r_8, r_{10} and r_{11} in the above expression do not overlap, and can therefore be extracted. Then, r_3 can be found from g_1, r_1 and r_2; r_6 from g_2, r_3, r_4, and r_5; r_9 from g_3, r_6, r_7, and r_8; and finally, r_9 from g_4, r_{10}, and r_{11}. To obtain $r_{12}, \ldots, r_{22}$, we define $g_5, \ldots, g_8$ similarly by using $r_{12}, \ldots, r_{22}$, and conclude the scheme by evaluating all r_i's. Overall, we have accessed 11 workers and downloaded 8 elements from each, instead of accessing 4 workers and downloading 22 elements from each.

In general, for $\ell \in [L]$ and $h \in [H]$, define $g^{(\ell,h,n)}$ as

$$\tilde{y}_{n,\ell} + \alpha_n^h \sum_{j=0}^{j'-1} \alpha_n^{jH} \tilde{y}_{n,\ell+j+1} + \alpha_n^{H(R-1)} \tilde{y}_{n,\ell+j'+1}, \tag{2}$$

where $j' = j'(h) \triangleq \lceil R - h/H \rceil - 1$. Further, for $i \in [L/R]$, let

$$\begin{aligned} h_i &\triangleq H(\lfloor (i-1)R \rfloor - (i-1)R + 1), \text{ and} \\ j_i &\triangleq j'(h_i) = \lceil R - h_i/H \rceil - 1. \end{aligned}$$

Finally, let $\ell_1 \triangleq 1$, and for $i \in \{2, \ldots, L/R\}$ define

$$\ell_i \triangleq \begin{cases} \ell_{i-1} + j_{i-1} + 1 & \text{if } h_i \neq H \\ \ell_{i-1} + j_{i-1} + 2 & \text{if } h_i = H. \end{cases}$$

Following the computation of $\tilde{y}_{n,\ell}$ for every $\ell \in [L]$, every server n computes $\{g^{(\ell_i,h_i,n)}\}_{i=1}^{L/R}$ and sends the results to the master. It follows from (2) that for every $i \in [L/R]$, the expression $g^{(\ell_i,h_i,n)}$ is an evaluation at α_n of

$$g_i \triangleq r_{\ell_i} + x^{h_i} \sum_{j=0}^{j_i-1} x^{jH} r_{\ell_i+j+1} + x^{H(R-1)} r_{\ell_i+j_i+1},$$

and $\deg(g_i) \leq HR - 1$. Hence, all polynomials $g_i(x)$ can be interpolated from the responses of $HR + 2A$ workers. It remains to show how the coefficients of the r_i's can be found from the g_i's.

We show how certain r_i's are extracted from $\{g_i(x)\}_{i=1}^{i'}$, where i' is an integer such that $h_2, \ldots, h_{i'} \neq H$ and $h_{i'+1} = H$ (this i' clearly exists, and it is at most $R_d + 1$). The remaining r_i's are extracted similarly. Given $g_1, \ldots, g_{i'}$, the master computes the following sum, in which the last term $x^{H(R-1)} r_{\ell_i+j_i+1}$ of g_i cancels out the first term $r_{\ell_{i+1}}$ of g_{i+1}, for every $i \in [i'-1]$.

$$\begin{aligned} \sum_{j=0}^{i'-1} (-1)^j x^{jH(R-1)} g_{j+1} = r_{\ell_1} &+ x^{h_1} \sum_{j=0}^{j_1-1} x^{jH} r_{\ell_1+j+1} \\ &- x^{H(R-1)+h_2} \sum_{j=0}^{j_2-1} x^{jH} r_{\ell_2+j+1} \\ &+ x^{2H(R-1)+h_3} \sum_{j=0}^{j_3-1} x^{jH} r_{\ell_3+j+1} \\ &\vdots \\ (-1)^{i'-1} & x^{(i'-1)H(R-1)+h_{i'}} \sum_{j=0}^{j_{i'}-1} x^{jH} r_{\ell_{i'}+j+1} \quad (3) \\ &+ (-1)^{i'-1} x^{i'H(R-1)} r_{\ell_{i'}+j_{i'}+1}. \quad (4) \end{aligned}$$

Therefore, to show that all r_i's in the above expression can be extracted, it is shown that the monomial degrees in the above sums, as are the ones in the first and the last summand, do not overlap. Since $(\ell_1, h_1) = (1, H)$ and $\deg(r_i) \leq H - 1$ for all i, it follows that r_{ℓ_1} and $x^{h_1} r_{\ell_1+1}$ do not share a common monomial, and hence the first sum does not overlap r_{ℓ_1}. For $k \in [i'-1]$, in order to show that the k'th sum does not share a common monomial with the $(k+1)$'th sum, we must show that

$$(k-1)H(R-1) + h_k + (j_k - 1)H + H - 1 < \\ kH(R-1) + h_{k+1}, \tag{5}$$

which readily follows from the definitions (see Lemma 7 in the appendix). Finally, to show that the last sum (3) does not share a common monomial with the last summand (4), we ought to show that

$$(i'-1)H(R-1) + h_{i'} + (j_{i'} - 1)H + H - 1 < i'H(R-1),$$

and this inequality follows from the definitions as well (see Lemma 8 in the appendix).

Thus, we have obtained the coefficients of all involved r_i's, except for $r_{\ell_2}, r_{\ell_3}, \ldots, r_{\ell_{i'}}$, that were canceled out in (4). However, r_{ℓ_2} can be extracted from g_1 and $r_{\ell_1}, \ldots, r_{\ell_2 - 1}$; r_{ℓ_3} can be extracted from g_2 and $r_{\ell_2}, \ldots, r_{\ell_3 - 1}$; and so on.

Remark 2. *In cases where $A > 0$, the download bandwidth of the suggested scheme strictly outperforms the one in ordinary LCC. In LCC, the user downloads L symbols from each one of $H+2A$ workers, $L(H+2A)$ symbols overall. In our scheme however, the user downloads L/R symbols from $RH + 2A$ workers, $L(H + 2A/R)$ symbols overall.*

For instance, if Example 1 is accompanied by $A = 1$ adversary, then the g_i's are interpolated by accessing 13 workers, and downloading 8 symbols from each, an overall of 104 symbols. In ordinary LCC, the 22 polynomials r_i are interpolated by accessing 6 workers, and downloading 22 symbols from each, an overall download of 132 symbols.

V. Utilizing Partial Work by UDMs

In this section we prove Theorem 4 and Theorem 5 by adding a layer of encoding. That is, we encode *the polynomial f itself* by using UDMs, and apply the encoded polynomials on a partially replicated Lagrange code. To exploit the full potential of the scheme in this section, one must possess some knowledge regarding the expected computation power of the workers in the system; this assumption aligns well with contemporary cloud services, in which performance guarantees of workers are given as a function of their cost. The approaches towards proving both theorems are similar. However, in one the error correction is performed between the clusters, and in the other, within each cluster. We begin by proving Theorem 5, and then proceed to show that Theorem 4 is an easy corollary of it.

We partition the N workers to C different clusters $\mathcal{C}_1,\ldots,\mathcal{C}_C$ of varying sizes $P_1,\ldots,P_C$, respectively. Broadly speaking, one should group together slow workers to large clusters, and fast workers to small ones. In addition, for every $i\in[C]$ assume that there are at most A_i adversaries in cluster $\mathcal{C}_i$, for some A_i such that $0\le A_i\le\lfloor\frac{P_i-1}{2}\rfloor$.

First, the data matrix $\mathbf{X}$ is encoded by using a Lagrange code of length C, producing codeword symbols $\tilde{\mathbf{x}}_1,\ldots,\tilde{\mathbf{x}}_C$. For every $i\in[C]$, the codeword symbol $\tilde{\mathbf{x}}_i$ is replicated P_i times and each copy is stored on a different server in $\mathcal{C}_i$. Second, in the computation phase, every server computes L functions on its stored codeword symbol. These L functions are linear combinations of the polynomials $\{f_i\}_{i=1}^L$, and are unique to each server. The precise functions $h_{i,1},\ldots,h_{i,L}$ of worker i can either be agreed upon in advance, be transmitted by the master to the worker incrementally or together, or be computed at the worker after receiving the polynomials f_i.

For $i\in[C]$ identify the workers in $\mathcal{C}_i$ by the integers $1,2,\ldots,P_i$, and let $\mathbf{B}_1,\ldots,\mathbf{B}_{P_i}$ be $L\times L$ UDMs over $\mathbb{F}$. The L functions of server j in $\mathcal{C}_i$ are

$$(h_{j,1},\ldots,h_{j,L})\triangleq(f_1,\ldots,f_L)\cdot\mathbf{B}_j.$$

Then, each server j computes $h_{j,1}(\tilde{\mathbf{x}}_i)$, transmits the result to the master, continues to compute and transmit $h_{j,2}(\tilde{\mathbf{x}}_i)$, and so on. In what follows, it is shown that once at least $(2A_i+1)L$ responses are received from cluster i for at least H clusters, *regardless* of their particular source within each cluster, the master is capable of finalizing the computation. The decoding process operates in two steps. In one, the true value of $\tilde{\mathbf{y}}_i=f(\tilde{\mathbf{x}}_i)$ is extracted from the partial responses of every server in $\mathcal{C}_i$. Then, these error free results are given to a decoding algorithm for LCC of length C, which finalizes the computation. Hence, we focus on the decoding process at the cluster level, which is identical in all clusters.

For a given cluster $\mathcal{C}_i$ and $j\in[P_i]$, let u_j be the number of responses that were obtained from worker j up to a given point in time, and notice that $0\le u_j\le L$ for every j. Thus, the response from $\mathcal{C}_i$ can be written as

$$\begin{aligned}(w_1,\ldots,w_{N'})\triangleq(\tilde{y}_{i,1},\ldots,\tilde{y}_{i,L})\cdot&\\ (\mathbf{b}_{1,1},\ldots,\mathbf{b}_{1,u_1},\ldots,\mathbf{b}_{P_i,1},\ldots,\mathbf{b}_{P_i,u_{P_i}}),&\end{aligned}\tag{6}$$

where $\mathbf{b}_{i,j}$ is the j'th column of $\mathbf{B}_i$, and $N'\triangleq\sum_{j=1}^{P_i}u_j$.

Lemma 6. *For every $i\in[C]$, if $N'\ge(2A_i+1)L$, then $f(\tilde{\mathbf{x}}_i)$ is decodable.*

The outline of the proof is as follows, and complete details can be found in the appendix.

Proof sketch. It suffices to show that no two distinct codewords $(w_1,\ldots,w_{N'})$ and $(w'_1,\ldots,w'_{N'})$ can be made equal by an addition of an error vector which results from the presence of A_i adversaries in the cluster. Thanks to linearity, this is equivalent to obtaining the zero codeword by encoding $\{\tilde{y}_{i,\ell}\}_{\ell\in[L]}$ that are not all zero, and introducing $2A_i$ adversaries. If such a scenario is possible, one uses Proposition 1 to get that $\sum_{j=0}^{L-1}\tilde{y}_{i,j+1}x^j$ is the zero polynomial, a contradiction. □

Therefore, once at least $(2A_i+1)L$ responses has arrived at the master from each one of at least H clusters $\mathcal{C}_i$, the master obtains the respective $f(\tilde{\mathbf{x}}_i)=(f_1(\tilde{\mathbf{x}}_i),\ldots,f_L(\tilde{\mathbf{x}}_i))$, and the computation can be finalized by LCC decoding.

Now, to prove Theorem 4, observe that if there are no stragglers in a given cluster $\mathcal{C}_i$, then L responses from it suffice to obtain $\tilde{\mathbf{y}}_i$. Furthermore, if there are at most A adversaries overall in the system, in the worst case there will be at most one adversary in each cluster, and hence the master may potentially fail to produce $\tilde{\mathbf{y}}_i$ in at most A clusters. Therefore, having obtained $\tilde{\mathbf{y}}_i$ from at least $H+2A$ clusters i, at most A of which are potentially erroneous, the master can apply LCC decoding, and the theorem follows.

References

[1] A. Barg and W. Park, "On linear ordered codes," *IEEE Int. Symp. on Inf. Th.* (ISIT), vol. 33, p. 34, 2015.

[2] P. Blanchard, R. Guerraoui, and J. Stainer, "Machine learning with adversaries: Byzantine tolerant gradient descent," *Adv. in Neural Inf. Proc. Sys.* (NIPS), pp. 119-129, 2017.

[3] S. Dutta, V. Cadambe, and P. Grover, "Short-dot: Computing large linear transforms distributedly using coded short dot products," *Adv. in Neural Inf. Proc. Sys.* (NIPS) pp. 2100–2108, 2016.

[4] K. Lee, C. Suh, and K. Ramchandran, "High-dimensional coded matrix multiplication," *IEE Int. Symp. on Inf. Th.* (ISIT), pp. 2418–2422, 2017.

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

[6] S. Li, S. M. M. Kalan, A. S. Avestimehr, and M. Soltanolkotabi, "Near-optimal straggler mitigation for distributed gradient methods," *IEEE Int. Parallel and Distributed Processing Symposium Workshops* (IPDPSW), pp. 857–866, 2018.

[7] S. Li, M. A. Maddah-Ali, Q. Yu, A. S. Avestimehr, "A Fundamental Tradeoff Between Computation and Communication in Distributed Computing," *IEEE Trans. on Inf. Th.*, vol. 64, no. 1, pp. 109–128, 2018.

[8] N. Raviv, Y. Cassuto, R. Cohen and M. Schwartz, "Erasure correction of scalar codes in the presence of stragglers," *IEEE Int. Symp. on Inf. Th.* (ISIT), pp. 1983–1987, 2018.

[9] N. Raviv and D. A. Karpuk, "Private polynomial computation from Lagrange encoding," *arXiv:1812.04142*, 2018.

[10] N. Raviv, I. Tamo, R. Tandon, and A. G. Dimakis, "Gradient coding from cyclic MDS codes and expander graphs," *Int. Conf. on Machine Learning* (ICML), 2018.

[11] M. Y. Rosenbloom and M. A. Tsfasman, "Codes for the m-metric," *Problemy Peredachi Informatsii*, vol. 33, no. 1, pp. 55–63, 1997.

[12] A. Shamir, "How to share a secret," *Comm. of the ACM*, vol. 22, no. 11, pp. 612–613, 1979.

[13] H. Sun and S. A. Jafar, "The capacity of private computation," *arXiv:1710.11098*, 2017.

[14] S. Tavildar and P. Viswanath, "Approximately universal codes over slow-fading channels," *IEEE Trans. on Inf. Th.*, vol. 52, no. 7, pp. 3233–3258, 2006.

[15] P. O. Vontobel and A. Ganesan, "On universally decodable matrices for spacetime coding," *Designs, Codes and Cryptography*, vol. 41, no. 3, pp. 325–342, 2006.

[16] N. J. Yadwadkar, B. Hariharan, J. E. Gonzalez, and R. Katz, "Multi-task learning for straggler avoiding predictive job scheduling," *The Journal of Machine Learning Research*, vol. 17, no. 1, pp. 3692–3728, 2016.

[17] Q. Yu, M. A. Maddah-Ali, and A. S. Avestimehr, "Polynomial codes: an optimal design for high-dimensional coded matrix multiplication," *Adv. in Neural Inf. Proc. Systems* (NIPS), pp. 4403–4413, 2017.

[18] Q. Yu, S. Li, N. Raviv, S. M. M. Kalan, M. Soltanolkotabi, S. Avestimehr, "Lagrange Coded Computing: Optimal design for resiliency, security, and privacy", *arXiv:1806.00939*, to appear in *The International Conference on Artificial Intelligence and Statistics* (AISTATS), 2019.

Appendix
Omitted proofs

Lemma 7. *For every $k \in [i'-1]$, we have that $(k-1)H(R-1)+h_k+(j_k-1)H+H-1 < kH(R-1)+h_{k+1}$.*

Proof. The claim is equivalent to

$$j_kH < H(R-1)+h_{k+1}-h_k+1. \tag{7}$$

On the one hand, we have

$$\begin{aligned} H(R-1)+h_{k+1}-h_k+1 &= \\ &= H(R-1)+H(\lfloor kR\rfloor - kR+1)- \\ &\quad H(\lfloor (k-1)R\rfloor-(k-1)R+1)+1\\ &= H(\lfloor kR\rfloor - \lfloor (k-1)R\rfloor -1)+1. \end{aligned}$$

On the other hand, we have

$$\begin{aligned} j_kH &= (\lceil R-\frac{h_k}{H}\rceil -1)H\\ &= H(\lceil R-\lfloor (k-1)R\rfloor+(k-1)R-1\rceil-1)\\ &= H(\lceil kR-\lfloor(k-1)R\rfloor-1\rceil-1)\\ &= H(\lceil kR\rceil-\lfloor(k-1)R\rfloor-2) \end{aligned}$$

where the last equality follows since $-\lfloor(k-1)R\rfloor-1$ is an integer. Therefore, it is readily verified that (7) is equivalent to

$$H(\lceil kR\rceil-2) < H(\lfloor kR\rfloor -1)+1. \tag{8}$$

Let $z\in\mathbb{Z}$ and $f\in[0,1)$ such that $kR=z+f$. Then, (8) is equivalent to $H(z-1)<H(z-1)+1$, which is clearly true. □

Lemma 8. *Following the definitions of Section IV, we have that $(i'-1)H(R-1)+h_{i'}+(j_{i'}-1)H+H-1 < i'H(R-1)$.*

Proof. First, observe that since $h_{i'+1}=H$, it follows from the definition of $h_{i'+1}$ that $i'R$ is an integer. Therefore, we have

$$\begin{aligned} h_{i'} &= H(\lfloor (i'-1)R\rfloor-(i'-1)R+1)\\ &= H(i'R+\lfloor -R\rfloor - i'R+R+1)\\ &= H(R+\lfloor -R\rfloor+1). \end{aligned} \tag{9}$$

Hence, it follows that

$$\begin{aligned} j_{i'}H &= (\lceil R-\frac{h_{i'}}{H}\rceil-1)H\\ &= (\lceil R-(R+\lfloor -R\rfloor+1)\rceil-1)H\\ &= (\lceil-\lfloor -R\rfloor-1\rceil-1)H\\ &= (-\lfloor-R\rfloor-2)H. \end{aligned} \tag{10}$$

Now, it is readily verified that the claim to be proved is equivalent to

$$h_{i'}+j_{i'}H-1<H(R-1),$$

that together with (9) and (10), is

$$\begin{aligned} H(R+\lfloor-R\rfloor+1)+(-\lfloor-R\rfloor-2)H-1 &< H(R-1)\\ -H-1 &< -H, \end{aligned}$$

which is clearly true. □

To prove Lemma 6, we identify $[N']$ with $\mathcal{N}\triangleq\{(s,t)|s\in[P_i],1\le t\le u_s\}$ in a way that follows from (6) naturally. That is, we denote $w_1=w_{1,1},\ldots,w_{u_1}=w_{1,u_1},w_{u_1+1}=w_{2,1},\ldots,w_{u_1+u_2}=w_{2,u_2}$, and so on. Hence, we have that $w_{s,t}=\zeta^{(t-1)}(\gamma_s)$ according to Proposition 1, where $\zeta(x)\triangleq\sum_{j=0}^{L-1}\tilde{y}_{i,j+1}x^j$. In addition, we cite the following lemma from [15], and provide its proof for completeness.

Lemma 9. *[15] If there exists $\mathcal{J}\subseteq[P_i]$ such that $\zeta^{(\ell-1)}(\gamma_j)=0$ for every $j\in\mathcal{J}$ and every $1\le\ell\le u_j$, and $\sum_{j\in\mathcal{J}}u_j\ge L$, then ζ is the zero polynomial.*

Proof. If $2\notin\mathcal{J}$, it follows that γ_j is a root of ζ of multiplicity u_j for every $j\in\mathcal{J}$, hence ζ has more roots than its degree, and the claim follows. Otherwise, we have that ζ has at least $L-u_2$ roots, and $\zeta^{(k-1)}(\infty)=\tilde{y}_{i,L-k+1}=0$ for all $1\le k\le u_2$, which implies that $\deg(\zeta)\le L-u_2-1$, and the claim follows once again. □

Proof of Lemma 6. It suffices to show that no two distinct codewords $(w_1,\ldots,w_{N'})$ and $(w'_1,\ldots,w'_{N'})$ can be made equal by an addition of an error vector which results from the presence of A_i adversaries in the cluster. To define the structure of such error vector, for an integer W and a vector $\mathbf{e}=(e_{s,t})_{(s,t)\in\mathcal{N}}$ in $\mathbb{F}^{N'}$, we say that $\mathbf{e}$ has u-weight at most W if there exists a subset $\mathcal{I}\subseteq[P_i]$ of size W such that

$$e_{s,t}\neq 0 \text{ only if } s\in\mathcal{I}.$$

Put differently, if there are at most A_i adversaries in $\mathcal{C}_i$, then the data received at the master from $\mathcal{C}_i$ is $(w_1,\ldots,w_{N'})+\mathbf{e}$, where $\mathbf{e}$ is of u-weight at most A_i. Hence, thanks to linearity, it suffices to show that the encoding in (6) cannot produce a nonzero codeword of u-weight at most $2A_i$.

Assuming the contrary, we have that there exists $\mathcal{I}\subseteq[P_i]$ of size $2A_i$ such that if $w_{s,t}\neq 0$ then $s\in\mathcal{I}$. Moreover, we have that

$$(2A_i+1)L\le\sum_{j=1}^{P_i}u_j=\sum_{j\in\mathcal{I}}u_j+\sum_{j\notin\mathcal{I}}u_j\le 2A_iL+\sum_{j\notin\mathcal{I}}u_j,$$

which implies that $\sum_{j\notin\mathcal{I}}u_j\ge L$. Hence, by defining $\mathcal{J}\triangleq[P_i]\setminus\mathcal{I}$ and applying Lemma 9, it follows that ζ is the zero polynomial, a contradiction. □