High-Dimensional Coded Matrix Multiplication

Kangwook Lee
School of EE
KAIST
kw1jjang@kaist.ac.kr

Changho Suh
School of EE
KAIST
chsuh@kaist.ac.kr

Kannan Ramchandran
Dept. of EECS
UC Berkeley
kannanr@eecs.berkeley.edu

***Abstract*—Coded computation is a framework for providing redundancy in distributed computing systems to make them robust to slower nodes, or stragglers. In [1], the authors propose a coded computation scheme based on maximum distance separable (MDS) codes for computing the product $\boldsymbol{A}^\intercal\boldsymbol{B}$, and this scheme is suitable for the case where one of the matrices is small enough to fit into a single compute node. In this work, we study coded computation involving large matrix multiplication where both matrices are large, and propose a new coded computation scheme, which we call *product-coded matrix multiplication*. Our analysis reveals interesting insights into which schemes perform best in which regimes. When the number of backup nodes scales sub-linearly in the size of the product, the product-coded scheme achieves the best run-time performance. On the other hand, when the number of backup nodes scales linearly in the size of the product, the MDS-coded scheme achieves the fundamental limit on the run-time performance. Further, we propose a novel application of low-density-parity-check (LDPC) codes to achieve linear-time decoding complexity, thus allowing our proposed solutions to scale gracefully.**

I. Introduction

Distributed systems featuring computing and storage capacity at unprecedented scale are driving today's big data era. These systems are characterized by various sources of individual node delays and latencies related to queueing, computing and communicating, that cannot be controlled or tracked at fine scale [2]. These unpredictable individual-component latencies result in a new kind of 'systems noise' that we need to deal with if we are to provide delay guarantees in large-scale distributed computing systems.

Coded computation is a principled framework for providing redundancy in distributed computing systems to make them robust to slower computing nodes, or *stragglers*. In [1], the authors propose a coded computation scheme called 'MDS-coded matrix multiplication', based on maximum distance separable (MDS) codes. The key idea is as follows. Imagine a distributed computing system with a single master node (master) and 3 worker nodes (workers). Assume that the latencies of the workers are unpredictable due to system noise. The goal is to compute a matrix multiplication $\boldsymbol{A}^\intercal\boldsymbol{B}$ as quick as possible in the presence of stragglers. The MDS-coded scheme first splits $\boldsymbol{A}$ into two submatrices, i.e., $\boldsymbol{A} = [\boldsymbol{A}_1\ \boldsymbol{A}_2]$, and precomputes the parity of the two, e.g., $\boldsymbol{A}_3 = \boldsymbol{A}_1 + \boldsymbol{A}_2$. Then, for each i, it assigns the task of computing $\boldsymbol{A}_i^\intercal\boldsymbol{B}$ to worker i. When worker i finishes its task, it sends the result back to the master node. One can observe that under this scheme the master node can compute $\boldsymbol{A}^\intercal\boldsymbol{B}$ as soon as *any* 2 out of the 3 task results are obtained. For instance, $\boldsymbol{A}_1^\intercal\boldsymbol{B}$ and $\boldsymbol{A}_3^\intercal\boldsymbol{B}$ are sufficient for the master node to reconstruct $\boldsymbol{A}_2^\intercal\boldsymbol{B} = \boldsymbol{A}_3^\intercal\boldsymbol{B} - \boldsymbol{A}_1^\intercal\boldsymbol{B}$, and hence $\boldsymbol{A}^\intercal\boldsymbol{B}$.

One assumption made in this approach is that the size of $\boldsymbol{B}$ is small enough since otherwise computing $\boldsymbol{A}_i^\intercal\boldsymbol{B}$'s with individual workers is infeasible. This assumption restricts its applicability of the proposed scheme to modern applications involving large-scale matrix multiplications.

This motivates us to study the problem of large matrix multiplication. We first generalize various distributed computation schemes to this new problem. We then design a new coded computation scheme based on *product codes*, called *product-coded matrix multiplication*. We analyze the expected run-time of various computation schemes, thus revealing interesting insights into which schemes perform best in which regimes. When the number of backup workers scales sub-linearly in the size of the product (the number of operations to compute the product), the product-coded scheme achieves the best average run-time performance. On the other hand, when the number of backup workers scales linearly in the size of the product, the MDS-coded scheme achieves the optimal average run-time.

Further, we show how low-density-parity-check (LDPC) codes provide our schemes with linear-time decoding algorithms, thus allowing them to scale gracefully.

A. Computation Model and Notation

Throughout the paper, without loss of generality, we assume that a worker in the distributed computing system is capable of computing a dot product of vectors of size d. When worker i is assigned a task of computing a dot product, the task completion time, denoted by T_i, is randomly distributed, Note that in this work, we do not distinguish time taken for computing a dot product and time taken for transmitting the computation result to the master node. Further, we denote the k^{th} order statistics of $(T_1, \ldots, T_n)$ by $T_{k:n}$. That is, $T_{k:n}$ is the k^{th} smallest value among $(T_1, \ldots, T_n)$. The overall computational time, at which the master node can fully decode the overall computation result, is denoted by T.

B. Related Work

The idea of applying codes to speed up distributed algorithms has been applied to various setups with different goals. In [3], an anytime coding scheme for approximate matrix multiplication is proposed, and it is shown that the proposed scheme can improve the quality of approximation compared with the other existing coded schemes for exact computation.

This work was supported by Institute for Information & communications Technology Promotion (IITP) grant funded by the Korea government (MSIP) (2017-0-00694, Coding for High-Speed Distributed Networks).

978-1-5090-4096-4/17/$31.00 ©2017 IEEE

In [4], the authors propose a scheme called 'Short-Dot', which induces some sparsity to the encoded matrices at the cost of reduced decoding flexibility. The authors show that the overall computation time can be reduced by carefully trading off the decoding flexibility with the amount of computation per worker. The authors of [5] study the optimal code design for computing gradients in a distributed system. The authors observe that in many machine learning problems, the gradient of a loss function is the sum of simpler gradients, each of which is computed with each data point. Based on the observation that the only thing that matters is the total gradient, the authors propose a novel coded computation scheme tailored for computing a sum of functions.

Codes are also shown useful for reducing the communication overhead. In [6], the authors reduce the communication overhead required for the shuffling phase of MapReduce. In [1], the authors reduce the communication cost required for rearranging training data points across distributed workers. The trade-off between computational time and communication overhead in the MapReduce framework is studied in [7].

II. Overview of The Existing Schemes

In this section, we overview a few existing schemes, which can be applied to the large matrix multiplication $\boldsymbol{A}^\intercal\boldsymbol{B}$. We assume that both matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ scale together, i.e., $\boldsymbol{A} \in \mathbb{R}^{d\times k}$ and $\boldsymbol{B} \in \mathbb{R}^{d\times k}$. Let us denote the i^{th} column of the matrix $\boldsymbol{A}$ and $\boldsymbol{B}$ by $\boldsymbol{a}_i \in \mathbb{R}^d$ and $\boldsymbol{b}_i \in \mathbb{R}^d$, respectively, i.e., $\boldsymbol{A} = [\boldsymbol{a}_1\ \boldsymbol{a}_2\ \cdots\ \boldsymbol{a}_k]$, and $\boldsymbol{B} = [\boldsymbol{b}_1\ \boldsymbol{b}_2\ \cdots\ \boldsymbol{b}_k]$. Let N be the number of workers. Note that $N \geq k^2$, as computing $\boldsymbol{A}^\intercal\boldsymbol{B}$ requires k^2 dot products, i.e., $\boldsymbol{a}_i^\intercal\boldsymbol{b}_j$'s for $1 \leq i, j, \leq k$.

A. Replication-based Computation

When $N = zk^2$ for some positive integer z, the replication scheme redundantly assigns each computation task $\boldsymbol{a}_i^\intercal\boldsymbol{b}_j$ to z workers. Each of the k^2 task results can be obtained from the fastest worker among the z ones computing the same dot product, and the master node can recover $\boldsymbol{A}^\intercal\boldsymbol{B}$ when it collects all the $\boldsymbol{a}_i^\intercal\boldsymbol{b}_j$'s.

B. (One-dimensional) MDS-coded Computation

The MDS-coded algorithm, proposed in [1], assumes that one of the two matrices, say $\boldsymbol{B}$, is small enough that computation of $\boldsymbol{a}_i^\intercal\boldsymbol{B}$ is computable with distributed workers. Hence, by viewing the large matrix multiplication problem $\boldsymbol{AB}$ as k instances of small matrix multiplication problem, i.e., $\boldsymbol{A}^\intercal\boldsymbol{B} = [\boldsymbol{A}^\intercal\boldsymbol{b}_1\ \boldsymbol{A}^\intercal\boldsymbol{b}_2\ \cdots\ \boldsymbol{A}^\intercal\boldsymbol{b}_k]$, one can apply the MDS-coded computation scheme. Assuming that $N = nk$, workers are divided into k groups of size n, each of which is dedicated to compute $\boldsymbol{A}^\intercal\boldsymbol{b}_j$ for some j. Consider the first group, which is for computing $\boldsymbol{A}^\intercal\boldsymbol{b}_1$. The MDS-coded scheme first encodes the k columns of $\boldsymbol{A}$ using an (n, k) MDS code to obtain n *coded* columns, say $\boldsymbol{a}_1$ to $\boldsymbol{a}_n$. It then assigns computation of $\boldsymbol{a}_i^\intercal\boldsymbol{b}_1$ to the i^{th} worker of this group. Similarly, the j^{th} group of n workers jointly computes $\boldsymbol{A}^\intercal b_j$.

The computational time of the MDS-coded computation is determined by the maximum of the computational times among the k groups, and the computational time of each group is determined by the k^{th} fastest worker among the n workers in the group.

C. Fully MDS-coded Computation

Treating $\boldsymbol{a}_i^\intercal\boldsymbol{b}_j$'s as k^2 systematic symbols, one could apply an (N, k^2) MDS code to obtain a set of N coded computation tasks. While this approach guarantees the optimal decoding flexibility, i.e., any k^2 task results are sufficient to recover $\boldsymbol{A}^\intercal\boldsymbol{B}$, this approach is inefficient since it significantly increases the amount of computation assigned to backup workers. To see this, consider the case of $N = k^2+1$. One can apply a systematic (k^2+1, k^2) MDS code to obtain the following k^2+1 computation tasks: $\boldsymbol{a}_1^\intercal\boldsymbol{b}_1, \boldsymbol{a}_1^\intercal\boldsymbol{b}_2, \ldots, \boldsymbol{a}_k^\intercal\boldsymbol{b}_{k-1}, \boldsymbol{a}_k^\intercal\boldsymbol{b}_k$, and $\sum_{i=1}^{k}\sum_{j=1}^{k} w_{ij}\boldsymbol{a}_i^\intercal\boldsymbol{b}_j$, where w_{ij}'s are encoding coefficients. Note that while each of the first k^2 computation tasks involves a single dot product, the last worker has to compute k^2 dot products (and scalar multiplications) and $k^2 - 1$ additions. Thus, this backup worker effectively becomes wasted. In general, the generator matrices of MDS codes are dense, implying heavy workloads of backup workers.

III. Product-coded Matrix Multiplication

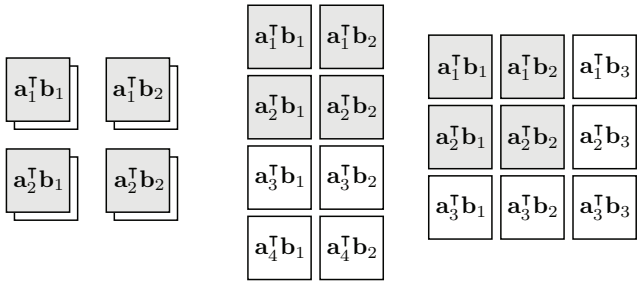
In this section, we propose a new coded computation scheme based on *product codes*, called the product-coded matrix multiplication. While the MDS-coded computation encodes computation along one dimension only, the product-coded scheme encodes computation along both dimensions, achieving an improved coding gain compared to the (one-dimensional) MDS-coded scheme. Product code is one way of constructing a larger code with small codes as building blocks [8]. For instance, when one is given an (n, k) MDS code, a product code can be constructed as follows. We first arrange k^2 symbols in a k-by-k array. Every row of the array is encoded with the (n, k) MDS code, resulting in a k-by-n array. Finally, each column of the array is encoded with the (n, k) MDS code, giving us an n-by-n array, including the n^2 encoded symbols. Let us call this product code an $(n, k)^2$ product code. By viewing $\boldsymbol{a}_i^\intercal\boldsymbol{b}_j$'s as k^2 input symbols, the product-coded computation applies an $(n, k)^2$ product code, made of an (n, k) MDS code, to obtain n^2 coded computation tasks. Assuming a linear MDS code, it can be shown that (i, j) entry of the coded computation task is simply $\boldsymbol{a}_i^\intercal\boldsymbol{b}_j$ where $\boldsymbol{a}_i$ is the i^{th} encoded column if the matrix $\boldsymbol{A}$ is encoded via an (n, k) MDS code, and $\boldsymbol{b}_i$ is the j^{th} encoded column of $\boldsymbol{B}$ under the same MDS code. Note that unlike the fully MDS-coded scheme, every computation tasks of the product-coded scheme involves only a single dot product, not requiring any additional computational overhead or unbalanced workloads.

Note that when k or more symbols are collected from any row or column, the entire row or column can be reconstructed via the decoding algorithm of the underlying MDS code. Hence, one can iteratively reconstruct the missing entries in such rows and columns until every rows and columns is either fully reconstructed or has more than $n - k$ missing entries.

A. Example: $k = 2$

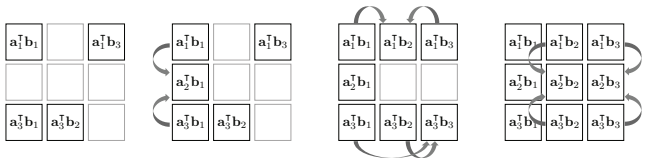
We now provide an illustration of the above schemes for the case of $k = 2$. That is, one wants to compute

$$\boldsymbol{A}^\intercal\boldsymbol{B} = \begin{bmatrix} \boldsymbol{a}_1^\intercal \\ \boldsymbol{a}_2^\intercal \end{bmatrix} \begin{bmatrix} \boldsymbol{b}_1 & \boldsymbol{b}_2 \end{bmatrix} = \begin{bmatrix} \boldsymbol{a}_1^\intercal\boldsymbol{b}_1 & \boldsymbol{a}_1^\intercal\boldsymbol{b}_2 \\ \boldsymbol{a}_2^\intercal\boldsymbol{b}_1 & \boldsymbol{a}_2^\intercal\boldsymbol{b}_2 \end{bmatrix}. \quad (1)$$



(a) 2x Replication (b) $(4,2)$ MDS code (c) $(3,2)^2$ Product code

Fig. 1: Illustration of different computation schemes.



(a) Iteration 0 (b) Iteration 1 (c) Iteration 2 (d) Iteration 3

Fig. 2: The decoding algorithm for the product-coded scheme.

Given $N = 8$ workers (or 4 backup workers), the 2x-replication scheme can be deployed: it assigns two workers for each of the 4 tasks, as shown in Fig. 1a.

Another choice is to use the $(4,2)$-MDS-coded matrix multiplication algorithm. That is, we encode the columns of $\boldsymbol{A}$ using a $(4,2)$ MDS code to obtain $\boldsymbol{a}_1$ to $\boldsymbol{a}_4$. We then apply the $(4,2)$-MDS coded matrix multiplication to $\boldsymbol{A}^\intercal\boldsymbol{b}_1$ and $\boldsymbol{A}^\intercal\boldsymbol{b}_2$ as in Fig. 1b.

With $N = 9$ workers (or 5 backup workers), one can use the $(3,2)^2$-product-coded matrix multiplication algorithm. We first apply a $(3,2)$ MDS code to both $\boldsymbol{A}$ and $\boldsymbol{B}$ to obtain $\boldsymbol{a}_1$ to $\boldsymbol{a}_3$ and $\boldsymbol{b}_1$ to $\boldsymbol{b}_3$. Then, the computation tasks $\boldsymbol{a}_i^\intercal\boldsymbol{b}_j$'s are assigned to the workers as illustrated in Fig. 1c.

Fig. 2 illustrates the peeling decoding process for the product-coded scheme. Assume that the master node has collected 4 computation results as shown in Fig. 2a. In the first iteration, the first column has 2 computation results, and hence can be decoded to obtain $\boldsymbol{a}_2^\intercal\boldsymbol{b}_1$. In the second iteration, the first and third rows are decodable, and the missing computation result in each row is recovered. Once the second iteration is done, the second and third column become decodable, thus allowing for the full recovery of $\boldsymbol{A}^\intercal\boldsymbol{B}$.

B. Decodability Condition of Product-coded Scheme

The computational times of the replication-based scheme and the MDS-coded scheme can be stated with order statistics. Unlike these schemes, the computational time of the product-coded scheme cannot be simply stated with order statistics because the decodability condition depends on the pattern formed by the completed tasks.

The peeling decoding algorithm for the product-coded scheme can be analyzed by viewing the iterative process as an edge-removal process in a bipartite graph [9]. Consider a bipartite graph with n left nodes (corresponding to the n rows) and n right nodes (corresponding to the n columns) with edges corresponding to the computation results that are *not* received yet. Observe that decoding a row (column) with less than or equal to $n-k$ missing entries can be viewed as removing the edges from a left (right) node if the degree is less than or equal to $n-k$. Therefore, the sufficient and necessary condition for successful decoding is non-existence of a subgraph with all nodes of degree at least $n-k+1$. In the graph theory terminology, the $(n-k+1)$-core of the bipartite graph, the largest subgraph with all nodes of degree at least $n-k+1$, must be empty.

In [10], the authors precisely characterize the sharp threshold of emergence of non-empty cores in a random graph. Later, the authors of [9] show that the same threshold holds for random bipartite graphs, stated as follows.

Theorem 1. *(From [9], [10]) Consider a random bipartite graph with n left nodes and n right nodes, where each edge presents with probability $\frac{\lambda}{n}$. Let $\ell \geq 3$ be fixed, $\pi_\ell(\lambda) = \sum_{i\geq\ell} \frac{e^{-\lambda}\lambda^i}{i!}$, and $\lambda_\ell = \min_{\lambda>0}\left[\frac{\lambda}{\pi_{\ell-1}(\lambda)}\right]$. Then, if $\lambda < \lambda_\ell$, the ℓ-core of the random bipartite graph is empty w.h.p. If $\lambda > \lambda_\ell$, the ℓ-core is non-empty w.h.p.*

IV. Computational time Analysis

In this section, we first review some useful results on order statistics given in [11], [12], and then present the asymptotic computational time analysis results. We focus on two different regimes. The first regime is where the number of backup workers scales sublinearly in the size of the product, i.e., $N = k^2 + \Theta(k)$, while the second regime is where the number of backup workers scales linearly in the size of the product, i.e., $N = k^2 + \Theta(k^2)$.

For simplicity, our computational time analysis assumes the exponential latency model: the response time of each worker is an exponential random variable with rate μ. We remark that the analysis can be easily extended to general response time distributions.

Due to the memoryless property, the expected value of the maximum of n exponential random variables of rate μ can be shown as H_n/μ where H_n is the n^{th} harmonic number, i.e., $H_n = \sum_{i=1}^{n} 1/i$. Similarly, the expected value of the k^{th} order statistics of n exponential random variables of rate μ is $(H_n - H_{n-k})/\mu$. Also, the minimum of z exponential random variable of rate μ is exponentially distributed with rate $z\mu$. For scaling k, $H_k = \log k + \gamma + o(1)$, where γ is a fixed constant.

A. Order Statistics

We first recap some known results on order statistics, which will be used for our computational time analysis. The probability density function and the cumulative distribution function of exponential random variables of rate μ are denoted by $f(t)$ and $F(t)$, respectively: $f(t) = \mu e^{-\mu t}$ and $F(t) = 1 - e^{-\mu t}$. Let T_i's be n i.i.d. exponential random variables of rate μ.

(1) (Central order statistics) For a fixed $0 < r < 1$,

$$T_{rn:n} \xrightarrow{p} N\left(t, \frac{r(1-r)}{nf^2(t)}\right), \tag{2}$$

where $t = F^{-1}(r)$.

(2) (Intermediate order statistics) For $k = k(n)$ such that $r_n := \frac{k}{n} \to 1$,

$$T_{k:n} \xrightarrow{d} N\left(t_n, \frac{n-k+1}{n^2 f^2(t_n)}\right), \tag{3}$$

where $t_n = F^{-1}(r_n)$.

(3) (Maximum of normal random variables) Consider n independent normal random variables $Y_i \sim N(\mu, \sigma^2)$ for $1 \le i \le n$. As n grows to infinity,

$$\mathbb{E}[Y_{n:n}] \approx \mu + \sigma\sqrt{2\log n}. \tag{4}$$

B. Regime I: Sublinear Number of Backup Workers

In this section, we focus on the case in which we have $\Theta(k)$ backup workers, which is sublinear in the size of the product, which is k^2. The following lemma first states the lower bound on the expected computational time, which holds for any computation scheme.

Lemma 1. *For a fixed integer t, as k grows to infinity, the expected computational time of any scheme with $k^2 + tk$ workers is lower bounded as $\mathbb{E}[T] \ge \frac{1}{\mu}\log\left(\frac{k+t}{t}\right) + o(1)$.*

Proof: Since the master node has to collect at least k^2 task results, the lower bound on the computational time is the $(k^2)^{\text{th}}$ order statistics of $k^2 + tk$ computational times. ∎

The following theorems characterize the asymptotic computational time performances of the MDS-coded scheme and the product-coded scheme.

Theorem 2. *For a fixed positive integer t, as k grows to infinity, the expected computational time of the $(k+t, k)$-MDS-coded scheme with $k^2 + tk$ workers is*

$$\mathbb{E}[T_{\text{MDS-coded}}] \approx \frac{1}{\mu}\log\left(\frac{k+t}{t}\right) + \frac{1}{\mu t}\sqrt{2(t+1)\log k}. \tag{5}$$

Proof: Let $n = k + t$. Recall that the computational time of the MDS-coded scheme is determined by the maximum of the computational times among the k groups, and the computational time of each group is determined by the k^{th} fastest worker among n workers in the group.

We first consider the computational time of each group. It is easy to see that $t_n = \frac{1}{\mu}\log\left(\frac{n}{t}\right)$ satisfies the condition $F(t_n) = r_n$, and $f(t_n) = \mu e^{-\mu t_n} = \frac{\mu t}{n}$. Hence, using (3), we can see that the computational time of each group converges in distribution to $N\left(\frac{1}{\mu}\log\left(\frac{n}{t}\right), \frac{(t+1)}{t^2\mu^2}\right)$. Since the computational times of all groups converge to the same normal random variable in distribution, the joint distribution of them also converges to a jointly normal random variable. Applying (4) to these independent normal random variables concludes the proof. ∎

Theorem 3. *For a fixed even integer t, as k grows to infinity, the expected computational time of the $(k + t/2, k)^2$-product-coded scheme with $k^2 + tk + t^2/4$ workers is*

$$\mathbb{E}[T_{\text{product-coded}}] \approx \frac{1}{\mu}\log\left(\frac{k+t/2}{c_{t/2+1}}\right). \tag{6}$$

Proof: Let $n = k + t/2$. By Thm. 1, the $(k + t/2, k)^2$-product-coded scheme can recover the overall computation results as soon as the number of missing computation results goes below $\lceil c_{t/2+1} n \rceil$. Thus, the computational time of the product-coded scheme is the $(n^2 - \lceil c_{t/2+1} n \rceil)^{\text{th}}$ order statistics of the n^2 computational times. Since the ceiling can be safely ignored in the asymptotic regime, the claim is proved. ∎

Thus, with $k^2 + tk + O(1)$ workers, the product-coded computation achieves an order-wise improved computational time performance compared to the MDS-coded computation.

C. Regime II: Linear Number of Backup Workers

We now consider the regime where the number of backup workers is $\Theta(k^2)$, which is linear in the minimum number of workers k^2. The lower bound on the average runtime of any computation is as follows.

Lemma 2. *For a fixed constant δ, as k grows to infinity, the expected computational time of any scheme with $(1+\delta)k^2$ workers is lower bounded as $\mathbb{E}[T] \ge \frac{1}{\mu}\log\left(\frac{1+\delta}{\delta}\right) + o(1)$.*

Note that in this regime, the replication scheme is a feasible choice. The following proposition shows that its average computational time scales as $\log k$ in the limit.

Proposition 1. *For a fixed positive integer δ, as k grows to infinity, the expected computational time of $(1+\delta)$-replication scheme with $(1+\delta)k^2$ workers is $\mathbb{E}[T_{rep}] = \frac{2\log k}{(1+\delta)\mu} + o(1)$.*

Proof: Recall that the computational time of the $(1+\delta)$-replication scheme is the maximum of k^2 task completion times, and the completion time of each task is the minimum of $(1+\delta)$ response times. Since the minimum of $(1+\delta)$ exponential random variables of rate μ is exponentially distributed with rate $(1+\delta)\mu$, the overall runtime is the maximum of k^2 exponential random variables with rate $(1+\delta)\mu$. ∎

We now analyze the computational time performance of the MDS-coded scheme. The following theorem shows that by using standard concentration inequalities, the computational time of the MDS-coded scheme can be shown optimal.

Theorem 4. *For a fixed constant δ, as k grows to infinity, the expected computational time of the $((1+\delta)k, k)$-MDS-coded scheme with $(1+\delta)k^2$ workers is*

$$\mathbb{E}[T_{\text{MDS-coded}}] = \frac{1}{\mu}\log\left(\frac{1+\delta}{\delta}\right) + o(1). \tag{7}$$

Proof: Recall that the computational time of the MDS-coded computation is determined by the maximum of the computational times among the k groups, and the computational time of each group is determined by the k^{th} fastest worker among the n workers in the group. Let us denote the computational time of $\boldsymbol{a}_i^\intercal \boldsymbol{b}_j$ by $T_{i,j}$.

Define $t_u := \frac{1}{\mu}\log\left(\frac{1+\delta}{\delta}\right) + \alpha\frac{\sqrt{\log k}}{k}$ for some constant $\alpha > 0$. A worker is *not* completed by time t_u with probability

$$1 - F(t_u) = \frac{\delta}{1+\delta}e^{-\mu\alpha\frac{\sqrt{\log k}}{k}} \simeq \frac{\delta}{1+\delta}\left(1 - \mu\alpha\frac{\sqrt{\log k}}{k}\right). \tag{8}$$

Applying Hoeffding's inequality [13], we have

$$\Pr\left(\{1 \le i \le n : T_{i,j} > t_u\} \ge \delta k\right) \le e^{-2\left(\frac{\delta}{1+\delta}\right)^2\mu^2\alpha^2\log k} \tag{9}$$

$$= k^{-2\left(\frac{\delta}{1+\delta}\right)^2\mu^2\alpha^2}. \tag{10}$$

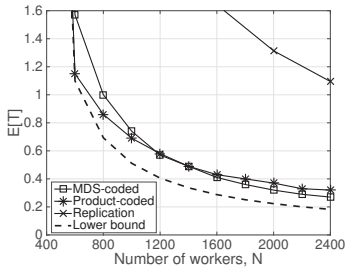


Fig. 3: The average computational times of various computation schemes. We run Monte Carlo simulations to estimate the average computational time.

Thus, applying the union bound, we have

$$\Pr(T_{\text{MDS-coded}} > t_u) \leq k^{1-2\left(\frac{\delta}{1+\delta}\right)^2\mu^2\alpha^2}. \quad (11)$$

By choosing α large enough, $\Pr(T_{\text{MDS-coded}} > t_u) = o(k^{-1})$. Since $T_{\text{MDS-coded}}$ is stochastically dominated by the maximum of $(1+\delta)k^2$ exponential random variables,

$$\mathbb{E}[T_{\text{MDS-coded}}] \leq (1-o(k^{-1}))t_u + o(k^{-1})H_{(1+\delta)k^2} \quad (12)$$

$$= \frac{1}{\mu}\log\left(\frac{1+\delta}{\delta}\right) + o(1). \quad (13)$$

Similarly, at time $t_l := \frac{1}{\mu}\log\left(\frac{1+\delta}{\delta}\right) - \alpha\frac{\log k}{k}$, one can also show that $\Pr(T_{\text{MDS-coded}} < t_l) = o(k^{-1})$. Thus, $\mathbb{E}[T_{\text{MDS-coded}}] \geq (1-o(k^{-1}))t_l = \frac{1}{\mu}\log\left(\frac{1+\delta}{\delta}\right) + o(1)$. ∎

Using a similar proof technique, one can also analyze the computational time of the product-coded scheme.

Proposition 2. *For a fixed constant δ, as k grows to infinity, the expected computational time of the $(\sqrt{1+\delta}k, k)^2$-product-coded scheme with $(1+\delta)k^2$ workers is*

$$\mathbb{E}[T_{\textit{product-coded}}] = \frac{1}{\mu}\log\left(\frac{1+\delta+\sqrt{1+\delta}}{\delta}\right) + o(1). \quad (14)$$

Thus, in the regime where the number of backup workers is $\Theta(k^2)$, the MDS-coded scheme asymptotically achieves the optimal computational time, while the product-coded computation achieves a strictly suboptimal computational time.

We now explain why the product-coded scheme outperforms the MDS-coded scheme only in the sublinear regime. When about k^2 out of $k^2 + tk$ workers are completed, it is likely that tk stragglers are not evenly spread out across different columns. Hence, recovering missing computation results by decoding both columns and rows significantly increases the decodability. On the other hand, when about k^2 out of $(1+\delta)k^2$ workers are completed, the δk^2 stragglers are evenly spread out across different columns with high probability. Thus, the one-way decoding of the MDS-coded scheme becomes sufficient, and the two-way redundancy of the product-coded scheme becomes wasteful.

V. Discussion

A. Simulation Results

We now provide a numerical comparison of the average computational times of the studied computation schemes. We set $k = 20$, and vary $N \in \{600, 800, \ldots, 2400\}$. Plotted in Fig. 3 is the average computational time as a function of N. The product-coded scheme closely matches the lower bound when N is small but starts performing worse than the MDS-coded scheme as N increases, as predicted in theory.

B. Decoding Complexity and LDPC Codes

The schemes studied in this work are built upon MDS codes. Since we deal with real numbers, a random matrix of size $k \times n$ can be used as the generator matrix of an (n, k) MDS code. The decoding complexity of such random linear codes is $O(k^3)$, which is negligible compared with the overall computational time if the matrix size d is much larger than k.

If d is comparable with k, one needs to make use of an MDS code that allows for an efficient decoding algorithm. However, all the known decoding algorithms, e.g., the best-known decoding algorithms for Reed-Solomon codes [14], have computational complexity of $\omega(k \log k)$. For this case, one can leverage efficient modern codes such as LDPC codes [15]. By replacing '*any* k out of n' of the (n, k) MDS condition with 'almost *any* $k(1+\epsilon)$ out of n' for an arbitrarily small $\epsilon > 0$, one may use the optimal LDPC code for erasure probability of $\frac{n-k}{n}$ to enjoy linear-time decoding complexity.

C. Open Problems

A few interesting questions remain open. Another natural question is whether a similar technique can be extended to the case of higher-dimensional linear operations such as tensor operations. Further, a complete characterization of run-time distributions is also an interesting future work.

References

[1] K. Lee, M. Lam, R. Pedarsani, D. Papailiopoulos, and K. Ramchandran, "Speeding up distributed machine learning using codes," *arXiv preprint arXiv:1512.02673*, 2015.

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

[3] N. Ferdinand and S. Draper, "Anytime coding for distributed computation," Presented at the 54th Annual Allerton conference on Communication, Control, and Computing, Monticello, IL, 2016.

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

[5] R. Tandon, Q. Lei, A. G. Dimakis, and N. Karampatziakis, "Gradient coding," *arXiv preprint arXiv:1612.03301*, 2016.

[6] S. Li, M. A. Maddah-Ali, and A. S. Avestimehr, "Coded MapReduce," in *Communication, Control, and Computing (Allerton), 2015 53rd Annual Allerton Conference on*, 2015.

[7] S. Li, M. A. Maddah-Ali, and A. S. Avestimehr, "A unified coding framework for distributed computing with straggling servers," *CoRR*, vol. abs/1609.01690, 2016.

[8] D. Costello and S. Lin, *Error control coding*. New Jersey, 2004.

[9] J. Justesen and T. Hoholdt, "Analysis of iterated hard decision decoding of product codes with Reed-Solomon component codes," in *Information Theory Workshop, 2007. ITW'07. IEEE*, 2007.

[10] B. Pittel, J. Spencer, and N. Wormald, "Sudden emergence of a giant k-core in a random graph," *Journal of Combinatorial Theory, Series B*, vol. 67, no. 1, pp. 111–151, 1996.

[11] H. A. David and H. N. Nagaraja, *Asymptotic Theory*. John Wiley & Sons, Inc., 2005, pp. 283–321.

[12] M. Petzold, "A note on the first moment of extreme order statistics from the normal distribution," in *rapport nr.: Seminar Papers*, no. 2000, 2000.

[13] W. Hoeffding, "Probability inequalities for sums of bounded random variables," *Journal of the American statistical association*, vol. 58, no. 301, pp. 13–30, 1963.

[14] F. Didier, "Efficient erasure decoding of Reed-Solomon codes," *arXiv preprint arXiv:0901.1886*, 2009.

[15] M. G. Luby, M. Mitzenmacher, M. A. Shokrollahi, and D. A. Spielman, "Efficient erasure correcting codes," *IEEE Transactions on Information Theory*, vol. 47, no. 2, pp. 569–584, 2001.