Coded Computation for Multicore Setups

Kangwook Lee
KAIST
kw1jjang@kaist.ac.kr

Ramtin Pedarsani
UC Santa Barbara
ramtin@ece.ucsb.edu

Dimitris Papailiopoulos
UW-Madison
dimitris@papail.io

Kannan Ramchandran
UC Berkeley
kannanr@eecs.berkeley.edu

***Abstract*—Consider a distributed computing setup consisting of a mater node and n worker nodes, each equipped with p cores, and a function $f(\boldsymbol{x}) = g(f_1(\boldsymbol{x}), f_2(\boldsymbol{x}), \ldots, f_k(\boldsymbol{x}))$, where each f_i can be computed independently of the rest. Assuming that the worker computational times have exponential tails, what is the minimum possible time for computing f? Can we use coding theory principles to speed up this distributed computation?**

Unlike the case where the local functions are linear (recently studied in [1]), in the non-linear case, it is not clear if traditional codes can provide *any* gains due to the high density of the parities. To resolve this problem, we propose the use of codes with sparse generator matrices for assigning local functions to different cores, and provide design guidelines for optimal constructions that minimize the runtime. We show that our coding solution offers (up to a constant factor) optimal performance, and has a provable *unbounded* gap compared to any uncoded schemes.

I. Introduction

In recent years, deploying algorithms on distributed systems has become the de facto choice for large-scale machine learning and data analytics. However, the performance of these algorithms is critically affected by bottlenecks in communication and delays caused by *stragglers*, or nodes that are substantially slower compared to the average node in the same system.

There have been several approaches for straggler mitigation, ranging from replicating jobs across redundant nodes, to simply dropping stragglers (if the computation at hand is robust to some errors) [2]–[9]. In a recent work [1], the authors proposed the use of erasure codes to mitigate the effects of stragglers; the high level idea is that stragglers can be assumed as erased, hence a master can start 'decoding' a partial set of local results to obtain the full result. More specifically, in [1] the authors employ simple MDS codes and coded caching techniques to speed up distributed matrix multiplication and data shuffling. In [10], the authors present a novel trade-off between computation and communication in MapReduce style setups. More recently, the authors of [11] present *gradient coding*, a technique that is used to recover the sum of all functions from a subset of coded symbols, with applications to distributed gradient descent. Recently, [12] presents a trade-off between coding flexibility and sparsity of a code, focusing on the matrix multiplication application. In [13], the authors propose a algorithm for speeding up distributed matrix multiplication in heterogeneous clusters.

These studies have limited their focus to distributed setups where the coding technique is oblivious to the potentially multicore nature of the each individual node. However, in practice, several of the publicly available cloud infrastructures provide CPU instances that can have up to 128 cores (e.g., x1.32xlarge on amazon EC2), or GPU instances with 1000s of CUDA cores (e.g., g2.2xlarge on amazon EC2) [14]. One could argue that each of these cores can be considered as an individual compute node in the distributed network. However, due to the fact that core-to-RAM communication is orders of magnitude faster than node-to-node communication in the network, we can aim for a more powerful solution that fully exploits the multi-core processing architecture in modern distributed computing.

Accordingly, in this work, we focus on master-worker architectures, where each of the n workers has p equally computationally capable cores. Our goal is to compute a separable function $f(\boldsymbol{x}) = g(f_1(\boldsymbol{x}), f_2(\boldsymbol{x}), \ldots, f_k(\boldsymbol{x}))$ as fast as possible, by exploiting as much the computational resources of the system. Specifically, our goal is to find the best function assignment and worker-master communication scheme that minimizes the time to compute f, under the (reasonable) assumption that the individual worker computational times are identically distributed and have exponential tails.

The main contribution of our work is a scheme for job assignment and coded computation that achieves the expected optimal runtime (up to constant factors). We supplement our constructions with a converse establishing that *any* job assignment (irrespective of job replication) not employing coding during worker-to-master communication has an unbounded gap of runtimes compared to our coded solution.

II. Distributed Model and Problem Formulation

In this work, we consider a distributed master-worker setup of n workers, each equipped with p cores. See Fig. 1 for high-level illustration of the system setup. The workers can compute locally assigned tasks and can send messages to the master. Our goal is to compute a function $f(\boldsymbol{x}) = g(f_1(\boldsymbol{x}), f_2(\boldsymbol{x}), \ldots, f_k(\boldsymbol{x}))$ in a distributed way, where $f_i : \mathbb{R}^d \to \mathbb{R}^w$, and any of the local functions f_i can be assigned to and computed locally by any of the n workers. For notational simplicity, we define a vector $\boldsymbol{f} = [f_1(\boldsymbol{x})^\intercal, f_1(\boldsymbol{x})^\intercal, \ldots, f_k(\boldsymbol{x})^\intercal]^\intercal$, and define $g(\boldsymbol{f}) = g(f_1(\boldsymbol{x}), f_2(\boldsymbol{x}), \ldots, f_k(\boldsymbol{x}))$.

A. Algorithm Protocol

We now formally describe a general protocol of distributed master-worker algorithms. In this work, we assume a single-round of communication without cooperation, i.e., worker tasks are fixed at the beginning of the algorithm, and cannot

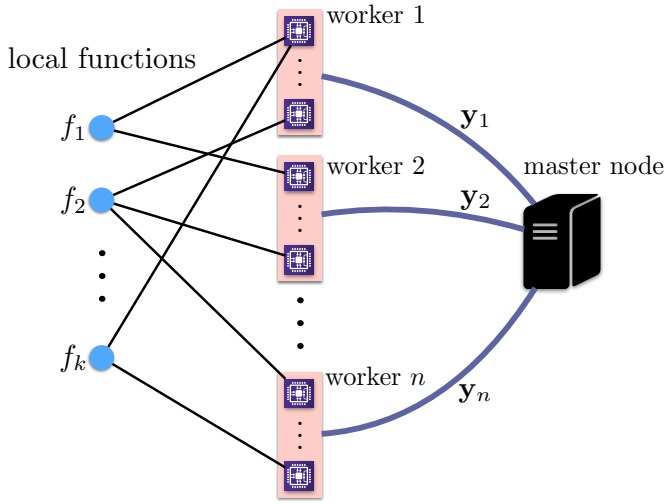


Fig. 1: A master-worker architecture of distributed computation with multiple cores per machines. Each of k local functions is assigned to a subset of n workers. Each worker computes the set of assigned functions, composes a message $\boldsymbol{y}_i$, and transmits the message to the master node. The master node collects $\boldsymbol{y}_i$'s until it can fully decode the k local functions.

be changed during runtime. Moreover, in the following we will assume that $w = 1$, i.e., the functions are 1-dimensional, however our results trivially generalize to arbitrary w. The *function assignment matrix* for worker i, denoted by $\boldsymbol{C}_i = [c_{i,j,m}] \in \mathbb{R}^{R_i \times k}$, is a coefficient matrix dictating which set of functions have to be computed by the worker as well as how a message of length R_i is constructed from locally computed function evaluations.

Given this function assignment matrix, the goal of worker i is to compute the message $\boldsymbol{y}_i := \boldsymbol{C}_i \boldsymbol{f}$ and transmit this message to the master node. Note that a distributed computation scheme can be fully specified by the function assignment matrices for all workers, i.e., $\boldsymbol{C} := [\boldsymbol{C}_1, \boldsymbol{C}_2, \ldots, \boldsymbol{C}_n]$.

A computation scheme $\boldsymbol{C}$ is either *uncoded* or *coded* depending on the structure of the function assignment matrices.

Definition 1 (Uncoded and Coded Schemes). A computation scheme is called uncoded if every row of $\boldsymbol{C}_i$ has at most one non-zero entry for all $1 \leq i \leq n$, and called coded otherwise.

In other words, if any of the workers composes a message by linearly combining the computed function values under a certain computation scheme, the scheme is a coded scheme; if a computation scheme does not add function values while composing messages, the scheme is an uncoded scheme.

For simplicity, we assume that a worker is assigned at most p local functions. We model this as a constraint on the function assignment matrix by imposing that its number of non-zero columns cannot be greater than p: by defining the set of local functions assigned to worker i by $\mathcal{F}_i$, we have

$$|\mathcal{F}_i| := |\{k : (\exists j)\, [c_{i,j,k} \neq 0]\}| \leq p.$$

With these definitions, the computation protocol can be fully described as follows. Given the function assignment $\boldsymbol{C}_i$, worker i first computes the allocated functions in $\mathcal{F}_i$ in parallel with its p cores, composes the message $\boldsymbol{C}_i \boldsymbol{f}$, and transmits it to the master.

The master node continues collecting messages from the workers until it can *fully* recover $\boldsymbol{f}$ and hence $f(\boldsymbol{x}) = g(\boldsymbol{f})$. Observe that this is feasible once the received equations with respect to $f_1, \ldots, f_k$ have a unique solution, or the corresponding coefficient matrix is full rank. These protocols of workers and master are described in Alg. 1 and Alg. 2

Algorithm 1 Master node's protocol

on Receiving an input argument $\boldsymbol{x}$
  Multicast $\boldsymbol{x}$ to all the workers
  $\boldsymbol{C}_{\text{rec}} = [\,]$, $\boldsymbol{y}_{\text{list}} = [\,]$
  while rank$(\boldsymbol{C}_{\text{rec}}) < k$ **do**
    on Receiving a message $\boldsymbol{y}_j$ from worker j
      $\boldsymbol{C}_{\text{rec}} \leftarrow [\boldsymbol{C}_{\text{rec}}\ \boldsymbol{C}_j]$, $\boldsymbol{y}_{\text{list}} \leftarrow \langle \boldsymbol{y}_{\text{list}}, \boldsymbol{y}_j \rangle$
  end while
  $\boldsymbol{f} \leftarrow \texttt{dec}(\boldsymbol{C}_{\text{rec}}, \boldsymbol{y}_{\text{list}})$
  Return $f(\boldsymbol{x}) = g(\boldsymbol{f})$

Algorithm 2 Worker node i's protocol

on Receiving an input argument $\boldsymbol{x}$
  Compute $[f_j(\boldsymbol{x})]_{j \in \mathcal{F}_i}$ and form $\boldsymbol{y}_i = \boldsymbol{C}_i \boldsymbol{f}$
  Send $\boldsymbol{y}_i$ to the master node

B. Communication and Computation Time

Denote the completion time of worker whose function assignment matrix is $\boldsymbol{C}'$ by $T(\boldsymbol{C}')$. We index the workers in the order of completion by $i_1, i_2, \ldots, i_n$. We also denote by ℓ the number of messages received after which full decoding can take place. That is, ℓ is the minimum ℓ' such that

$$\text{rank}[\boldsymbol{C}_{i_1}^{\intercal} \ \cdots \ \boldsymbol{C}_{i_{\ell'}}^{\intercal}] = k. \tag{1}$$

Thus, the overall runtime of the algorithm of a computation scheme $\boldsymbol{C}$, denoted by $T(\boldsymbol{C})$, is the completion time of worker i_ℓ, which is $T(\boldsymbol{C}_{i_\ell})$.

We assume that $T(\boldsymbol{C}_i)$ is composed of communication time and computation time:

$$T(\boldsymbol{C}_i) = T_{\text{comm}}(\boldsymbol{C}_i) + T_{\text{comp}}(\boldsymbol{C}_i).$$

For the communication time, we assume that the uplink communication bandwidth from each machine to the master node is fixed, and hence the time it takes for worker i to send a message of length R_i is assumed to be R_i, i.e., $T_{\text{comm}}(\boldsymbol{C}_i) = R_i$.

At this point, we abstract away decoding and encoding complexities from our model, but we will revisit the complexity of decoding in Section IV-B. Our goal is to find the optimal assignment and coding schemes, i.e.,

$$\boldsymbol{C}^\star = \arg\min_{\boldsymbol{C}} \mathbb{E}[T(\boldsymbol{C})]. \tag{2}$$

To design optimal function assignment matrices, we need to define a model for the local compute times. We assume

fully correlated computation times of p cores inside a single worker, and let $X_i,\ 1 \leq i \leq n$ be the random variable denoting the computation time of the i^{th} worker. We assume that each core is equally computationally strong and that all functions f_i require *on average* the same computation time; since $|\mathcal{F}_i| \leq p$, we will assume that all workers, on average, spend the same computational effort on the allocated jobs, i.e., $\mathbb{E}[X_i] = C$.

We model the runtime of the workers by a random variables X_i that are independent and identically distributed –and similar to [1]– they follow an exponential distribution with c.d.f. $\Pr(X_i \leq x) = 1 - e^{-\lambda x}$, with mean $\mathbb{E}[X_i] = \frac{1}{\lambda} = C$. With this exponential computational-time model, we present the following useful results on the order statistics: the expected value of the maximum of n exponential random variables of rate μ is H_n/μ where $H_n = \sum_{i=1}^{n} 1/i$; the expected value of the k^{th} order statistics of n exponential random variables of rate μ is $(H_n - H_{n-k})/\mu$; and the minimum of r exponential random variable of rate μ is exponentially distributed with rate $r\mu$. Further, $H_k = \log k + \gamma + o(1)$, where γ is a fixed constant.

Our goal is to characterize $\min_{\boldsymbol{C}} T(\boldsymbol{C})$, and obtain fundamental bounds for uncoded and coded approaches. In this paper, we focus on the practically relevant regime where the number of machines n scales linearly with the number of functions k, the number of cores per machine p scales at least logarithmically with the number of functions k, and the average computational-time of every machine C is a constant. This regime is formally stated in the following assumption.

Assumption 1. *We assume that the number of workers is a constant times the number of functions, i.e., $\frac{n}{k} = \gamma = \Theta(1)$ for some $\gamma \geq 1$. Moreover, the number of cores is at least logarithmic, i.e., $p = \Omega(\log k)$. Finally, the average time to compute one functions is assumed to be constant $C = \Theta(1)$.*

Remark 1. Apart from its simplicity, our model of fully correlated times inside a single worker is motivated by the fact that the delays of completion time of distributed workers are mainly caused by extrinsic system delays, such us communication bottlenecks, shared resources, failed machines, etc [15]. More often than not, locally all p cores are expected to perform approximately the same, without any stragglers.

Our main theorem is a "capacity" bound type of result on the optimal runtime for a coded distributed setup.

Theorem 1. *(informal) Under Assumption 1, any scheme for distributed computation of $f(\boldsymbol{x})$ requires runtime at least $\Omega(1)$. Any strictly uncoded distributed computation scheme requires at least expected time $\Omega(\sqrt{\log k})$. There exists coded distributed computation schemes that achieve the expected optimal runtime of $\Theta(1)$.*

In the following sections, we formalize the above result and present our converse and code construction that leads to the (up to constants) tight $\Theta(1)$ bound on the runtime of *coded computation for multicore setups*.

III. Uncoded Schemes

In this section, we study the performance of uncoded schemes, and show that the best uncoded scheme has an expected runtime of $\Omega(\sqrt{\log k})$. Further, this lower bound can be achieved by an optimal repetition scheme.

We first consider a replication-based uncoded scheme where $R,\ 1 \leq R \leq p$, functions are grouped together, and each of k/R groups of functions are replicated γR times. More precisely, assume that k/R and nR/k are integer numbers. Then, the functions are divided to k/R groups as follows:

$$\mathcal{J}_i = \{f_{R(i-1)+1}, \ldots, f_{Ri}\}, \quad i \in [k/R]. \tag{3}$$

Each group is then repeatedly assigned to $nR/k = \gamma R$ machines. The following theorem states the optimal latency performance of the replication uncoded schemes.

Theorem 2. *Under Assumption 1, the expected runtime of the optimal replication scheme is $\Theta\left(\sqrt{\log k}\right)$.*

Proof: For notational simplicity, we denote the runtime of the repetition scheme, $T(\boldsymbol{C}^{\text{repetition}})$, simply by T. Since every machine computes R functions and the scheme is uncoded, the communication time is R for all workers. Further, the computational time is the first time that every function is computed in at least one of the machines. Let $\tilde{T}_i$ be the first time that $f_i(\boldsymbol{x})$ is computed in at least one of the γR machines that $f_i(\boldsymbol{x})$ is assigned to. Since the repetition scheme is uncoded, we have

$$T = R + \max_{i \in [k]} \tilde{T}_i. \tag{4}$$

Note that since the functions are grouped according to (3), the functions in each group are computed at the same time by the fastest of the γR machines that the group is assigned to, i.e.

$$\tilde{T}_{R(i-1)+1} = \tilde{T}_{R(i-1)+2} = \cdots = \tilde{T}_{Ri}$$

for $i \in [k/R]$. Further, since the computing times of the machines are i.i.d., one finds that $\tilde{T}_i$ is distributed as the minimum of γR i.i.d. random variables with exponential distribution for all $i \in [k]$. Thus, the computational time term in (4) can be written as the maximum of k/R i.i.d. random variables, each distributed as the minimum of γR i.i.d. exponential random variables:

$$T = R + \max_{i \in [k/R]} \tilde{T}_{1+R(i-1)}.$$

One can then compute that

$$\begin{aligned} \mathbb{E}[T] &= R + \frac{C}{\gamma R}\left(1 + \frac{1}{2} + \cdots + \frac{1}{k/R}\right) \\ &= R + \frac{C}{\gamma R}\log(k/R) + o(1). \end{aligned} \tag{5}$$

By optimizing the expression over R, one can find the optimal uncoded scheme. First, it is easy to show that $\mathbb{E}[T]$ is concave in R. Let $a = \frac{C}{\gamma}$. By equating $\frac{d\mathbb{E}[T]}{dR}$ with 0, we have $R^2 = a\log(k/R) + a$. The above equation can be solved in closed using the Lambert function $W(x)$ that is the solution

to $f(x) = xe^x$. Using the fact that $W(x) = \log x - \log\log x + o(1)$, we obtain

$$R^\star = \sqrt{\frac{a}{2} W\left(\frac{2}{a} k^2 e^2\right)} = \sqrt{\log k} + o(1). \tag{6}$$

Plugging this in (5) concludes the proof. ∎

Next, we show that under Assumption 1, no uncoded scheme is able to achieve an expected completion time of $o(\sqrt{\log k})$. This result is stated formally in the following theorem.

Theorem 3. *Under Assumption 1, the expected runtime of* any *uncoded scheme is* $\Omega(\sqrt{\log k})$.

Proof: We prove the theorem by contradiction. Suppose that there exists an uncoded scheme with expected completion time that is $o(\sqrt{\log k})$. This implies that the deterministic communication time of the uncoded scheme is also $o(\sqrt{\log k})$, i.e. the number of functions assigned to each machine is $o(\sqrt{\log k})$ since in the uncoded scheme, the computed functions at each machine are sent back to the master node one by one without coding. Let $R = o(\sqrt{\log k})$ be the maximum number of functions that is assigned to one machine.

We next study the expected computation time $T_{\text{comp}}(\boldsymbol{C}^{\text{uncoded}})$ of an arbitrary uncoded scheme $\boldsymbol{C}^{\text{uncoded}}$. Since the total number of functions computed in all cores (including the repetitions) is $\mathcal{O}(Rn)$, the average repetition factor of the functions is upper bounded by $Rn/k = o(\sqrt{\log k})$. Thus, a fraction $1 - o(1)$ of the functions are repeated $o(\sqrt{\log k})$ times. We denote the set of such functions by S. Note that if a positive fraction of the functions are repeated $\Omega(\sqrt{\log k})$ times, then the average repetition factor would also be $\Omega(\sqrt{\log k})$. From now on, we find a lower bound on the average completion time of the uncoded scheme by assuming that the $o(1)$ fraction of functions that are repeated $\Omega(\sqrt{\log k})$ times are immediately available at the master node when the computations start, and focusing only on the functions that are repeated $o(\sqrt{\log k})$ times, i.e. the functions S. Let $\gamma = o(\sqrt{\log k})$ be the maximum number of times that a function in set S is computed across machines. Let $\mathcal{I}_i$ be the set of machines that compute function i. Thus, $|\mathcal{I}_i| \leq \gamma$ for $i \in S$. Let $\tilde{T}_i$ be the first time that f_i is computed by at least one of the machines in $\mathcal{I}_i$. Then,

$$T_{\text{comp}}(\boldsymbol{C}^{\text{uncoded}}) \geq \max_{i \in S} \tilde{T}_i. \tag{7}$$

Note that the random variables $\tilde{T}_i,\ i \in S$ are not independent, which complicates the exact calculation of the right hand side of (7). We next claim that there exists a set of $\Theta(k/R^2)$ functions in S that do not share any common machines among them, i.e. they are computed on mutually disjoint set of machines. To see this, without loss of generality, consider $f_1 \in S$ that is computed over $\mathcal{I}_1$. There are at most γR functions that are computed over all the cores of the machines in $\mathcal{I}_1$. Thus, f_1 does not share a machine with at least $|S| - \gamma R$ other functions. Without loss of generality, assume that f_2 is among those. Repeating the argument, f_2 does not share a machine with at least $|S| - \gamma R$ other functions. Thus, f_1 and f_2 do not share a machine with at least $|S| - 2\gamma R$ functions. Thus, by repeating the above argument, we find that there exists a set S_1 of $|S|/\gamma R = \Theta(k/R^2)$ functions that are computed over mutually disjoint machines. Thus, the random variables $\{\tilde{T}_i\}$ for $i \in S_1$ are mutually independent. Further,

$$T_{\text{comp}}(\boldsymbol{C}^{\text{uncoded}}) \geq \max_{i \in S} \tilde{T}_i \geq \max_{i \in S_1} \tilde{T}_i. \tag{8}$$

Using the property of exponential random variables and independence, we can easily compute that

$$\mathbb{E}\left[\max_{i \in S_1} \tilde{T}_i\right] \geq \Theta\left(\frac{\log |S_1|}{R}\right) = \Theta\left(\frac{\log k}{R}\right) = \Omega\left(\sqrt{\log k}\right),$$

since $R = o(\sqrt{\log k})$. Therefore, the average completion time is dominated by the average computation time that is $\Omega(\sqrt{\log k})$. This completes the proof of the theorem. ∎

IV. Coded Schemes

In this section we will show how codes with sufficiently sparse parities, can offer orderwise optimal results. First, observe that traditional MDS codes are unsuitable for this setup due to the fact that they would require dense parities. This is because a dense parity that involves all k functions $f_1, \ldots, f_k$, implies that each of the n workers has to compute *all* of the k functions locally. In this section, we present how to go beyond MDS codes by using simple random linear codes and sparse-graph codes, which are shown to offer order-wise optimal performance.

A. Random Sparse Codes

We first propose a random-linear-coded computation scheme. Under this scheme, every worker computes a certain set of functions, linearly combines them, and sends to the master a message of length 1, i.e., $R_i = 1$ for all i. The function assignment matrix of user i, $\boldsymbol{C}_i \in \mathbb{R}^{1 \times k}$, is randomly drawn as a Gaussian random vector whose entries are i.i.d. zero-mean Gaussian random variables with unit variance 1 with probability p_k, and 0 with probability $1 - p_k$. The following theorem characterizes the expected runtime of this random scheme.

Theorem 4. *Under Assumption 1, with probability* $1 - e^{-\alpha \cdot p_k \cdot k}$*, the random-linear-coded computation* $\boldsymbol{C}^{random}$ *with* $p_k \geq \frac{\beta \log k}{k}$*, for some universal positive constants* α, β*, achieves*

$$\mathbb{E}[T(\boldsymbol{C}^{random})] = 1 + C \log\left(\frac{\gamma}{\gamma - 1}\right) + o(1) = \Theta(1). \tag{9}$$

Proof: First of all, assume that after the fastest k nodes we can recover our functions $f_1, \ldots, f_k$, then the completion time for such a scheme is $\Theta(1)$, due to the order statistics of the exponential distribution, i.e., the expected completion time is equal to the expected time that it takes the fastest k machines to return their results.

Then, the main question is what is the chance of successful decoding after receiving a random subset of k nodes. To do that, we employ a recent result in random matrix theory, i.e.,

Corollary 1.8 [16], which states that a random matrix of size k by k whose entries are standard Gaussian variables with probability $p_k \geq \frac{\beta \log k}{k}$ (and 0 otherwise) is non-singular with probability at least $1 - e^{-\alpha \cdot p_k \cdot k}$, where α and β are universal constants. ∎

B. LDGM codes

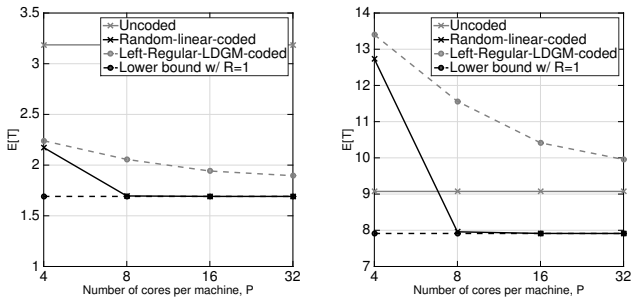
While the random-linear-coded computation achieves the constant runtime, the computational complexity of its decoding algorithm is $\mathcal{O}(k^3)$. Low-density generator matrix (LDGM) codes are linear codes whose generator matrices are sparse, i.e., the number of non-zero entries in each row is $\mathcal{O}(1)$. The sparse nature of their generator matrices allow for linear time decoding algorithm such as the peeling decoder [17]. Accordingly, we propose the LDGM-coded computation scheme, a computation scheme whose assignment matrix is the generator matrix of efficient LDGM codes.

However, compared to the random-linear-coded scheme, the LDGM-coded scheme is not able to collect all of the k task results by a fixed time for the following reason. Under the LDGM-coded scheme, each task is assigned a constant number of times on average, and hence at least a constant fraction of the tasks are assigned a constant number of times. The probability that each of these tasks is not completed by a fixed time is constant, and hence the average time to collect all task results is $\omega(1)$. Thus, the LDGM-coded computation scheme cannot achieve $\Theta(1)$ runtime in theory.

However, the probability that some of the task results are not completed by a fixed time can be made arbitrarily small by assigning a sufficiently large constant number of cores per worker. Further, while we focus on the setup where the target function needs to be exactly computed as well as all the function assignments are fixed prior to the actual computation process, some of these assumptions might be relaxed in practice. For instance, if a small number of tasks are missing after some fixed time, the master node may be able to reassign missing tasks to the idle workers at some additional cost. Also, some computation tasks – such as computing gradients of a loss function of machine learning problems – are inherently robust to missing computation tasks.

V. Simulation Results

In this section, we simulate the runtime performance of the computation schemes. We assume that $n = 256$, $k = 128$, and $p \in \{4, 8, 16, 32\}$. For the uncoded computation scheme, we vary $R \in \{1, 2, 4, 8, 16, 32\}$, choose the optimal value of $R^\star \leq p$ for each p, and plot the average runtimes of the respective optimal schemes. For both coded computation schemes, we run Monte Carlo simulation to estimate the runtime of them. The LDGM-coded schemes are designed based on LDGM codes with regular-left degree distribution. Shown in Fig. 2 are the average runtimes achieved by different schemes along with the lower bound on the runtime. We can observe that the random-linear-coded computation scheme achieve near-optimal runtimes for most cases.



(a) $C = 1$ (b) $C = 10$

Fig. 2: **Average runtime of various computation schemes.** The average runtime of different computation schemes are plotted. Plotted on left is the comparison of the runtimes of different computation schemes when $C = 1$, and plotted on right when $C = 10$. Note that we use the optimal value of R for each p for the uncoded scheme. One can observe that the random-linear-coded scheme achieves the best performance in most cases.

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] M. Zaharia, A. Konwinski, A. D. Joseph, R. H. Katz, and I. Stoica, "Improving MapReduce performance in heterogeneous environments," in *Proc. of the 8th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2008.

[3] G. Ananthanarayanan, S. Kandula, A. G. Greenberg, I. Stoica, Y. Lu, B. Saha, and E. Harris, "Reining in the outliers in Map-Reduce clusters using Mantri," in *Proc. of the 9th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2010.

[4] G. Ananthanarayanan, A. Ghodsi, S. Shenker, and I. Stoica, "Effective straggler mitigation: Attack of the clones," in *Proc. of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2013.

[5] N. B. Shah, K. Lee, and K. Ramchandran, "When do redundant requests reduce latency?" in *Proc. of the 51st Annual Allerton Conference on Communication, Control, and Computing*, 2013.

[6] D. Wang, G. Joshi, and G. W. Wornell, "Efficient task replication for fast response times in parallel computation," in *Proc. of ACM SIGMETRICS*, 2014.

[7] K. Gardner, S. Zbarsky, S. Doroudi, M. Harchol-Balter, and E. Hyytiä, "Reducing latency via redundant requests: Exact analysis," in *Proc. of ACM SIGMETRICS*, 2015.

[8] M. Chaubey and E. Saule, "Replicated data placement for uncertain scheduling," in *Proc. of IEEE International Parallel and Distributed Processing Symposium Workshop (IPDPS)*, 2015.

[9] J. Chen, X. Pan, R. Monga, S. Bengio, and R. Jozefowicz, "Revisiting distributed synchronous sgd," *preprint*, 2016.

[10] S. Li, M. A. Maddah-ali, and S. Avestimehr, "Coded MapReduce," Presented at the 53rd Annual Allerton conference on Communication, Control, and Computing, Monticello, IL, 2015.

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

[12] 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.

[13] A. Reisizadehmobarakeh, S. Prakash, R. Pedarsani, and S. Avestimehr, "Coded computation over heterogeneous clusters," *arXiv preprint arXiv:1701.05973*, 2017.

[14] "EC2 Instance Types - Amazon Web Services (AWS)," https://aws.amazon.com/ec2/instance-types/, accessed: 2017-01-23.

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

[16] A. Basak and M. Rudelson, "Invertibility of sparse non-hermitian matrices," *arXiv preprint arXiv:1507.03525*, 2015.

[17] T. Richardson and R. Urbanke, *Modern coding theory*. Cambridge University Press, 2008.