"Short-Dot": Computing Large Linear Transforms Distributedly Using Coded Short Dot Products

Sanghamitra Dutta, *Student Member, IEEE*, Viveck Cadambe, *Member, IEEE*, and Pulkit Grover, *Senior Member, IEEE*

***Abstract*— We consider the problem of computing a matrix-vector product Ax using a set of P parallel or distributed processing nodes prone to "straggling," *i.e.*, unpredictable delays. Every processing node can access only a fraction (s/N) of the N-length vector x, and all processing nodes compute an equal number of dot products. We propose a novel error correcting code–that we call "Short-Dot"–that introduces redundant, shorter dot products such that only a subset of the nodes' outputs are sufficient to compute Ax. To address the problem of straggling in computing matrix-vector products, prior work uses replication or erasure coding to encode parts of the matrix A, but the length of the dot products computed at each processing node is still N. The key novelty in our work is that instead of computing the long dot products as required in the original matrix-vector product, we construct a larger number of redundant and short dot products that only require a fraction of x to be accessed during the computation. Short-Dot is thus useful in a communication-constrained scenario as it allows for only a fraction of x to be accessed by each processing node. Further, we show that in the particular regime where the number of available processing nodes is greater than the total number of dot products, Short-Dot has lower *expected computation time* under straggling under an exponential model compared to existing strategies, e.g. replication, in a scaling sense. We also derive fundamental limits on the trade-off between the length of the dot products and the recovery threshold, i.e., the required number of processing nodes, showing that Short-Dot is near-optimal.**

***Index Terms*— Algorithm-based fault tolerance, coded computing, matrix sparsification, stragglers.**

I. Introduction

THIS work proposes a novel coding-theory-inspired technique for speeding up computing linear transforms of high-dimensional data by distributing it across multiple processing nodes that compute shorter dot products. Our main focus is on addressing the "straggler effect," i.e., the problem of delays caused by a few slow processing nodes that bottleneck the entire computation. We provide a technique (building on [2]–[5]) that introduces redundancy in the computation by designing a new error correction mechanism that allows the size of the individual dot products computed at each processing node to be shorter than the length of the input. Shorter dot products offer advantages in computation, storage and communication in distributed linear transforms. In addition to straggler tolerance, our coding scheme can also be used to detect or correct erroneous computations, and is therefore potentially applicable to computing systems with erroneous hardware as well [6]–[8].

The problem of computing linear transforms of high-dimensional vectors is *the* critical step [9] in several machine learning and signal processing applications. Dimensionality reduction techniques such as Principal Component Analysis (PCA), Linear Discriminant Analysis (LDA), taking random projections require the computation of short and fat linear transforms on high-dimensional data. Linear transforms are the building blocks of various machine learning algorithms, e.g. regression and classification etc., and are also used in acquiring and preprocessing data through Fourier transforms, wavelet transforms, filtering, etc. Fast and reliable computation of linear transforms are thus a necessity for low-latency inference [9]. Due to the saturation of Moore's law, increasing the speed of computing in a single node is becoming difficult, forcing practitioners to adopt distributed processing to speed up computing for ever-increasing data dimensions and sizes.

A common problem in distributed systems is "straggling" [10] where a few slow or faulty processing nodes (stragglers) can delay the entire computation. Straggling is attributed to network latency, shared resources, maintenance activities, and power limitations [2], among other factors. Specific to computing matrix vector products, classical distributed techniques, e.g., Block-Striped Decomposition [11], Fox's method [11], [12], and Cannon's method [11], rely on dividing the computational task equally among all available processing nodes[1] without any redundant computation. The fusion node collects the outputs from each processing node to complete the computation and thus has to wait for all the processing nodes to finish. Popular techniques for straggler mitigation include different forms of checkpointing [15] on top of straggler detection (e.g. that used in Hadoop [16]), that periodically save the current state of the system, and return to

Manuscript received January 27, 2018; revised January 30, 2019; accepted May 5, 2019. Date of publication July 9, 2019; date of current version September 13, 2019. This work was supported in part by the Systems on Nanoscale Information fabriCs (SONIC) which is one of the six SRC STARnet Centers sponsored by MARCO and DARPA, and in part by NSF Awards under Grant 1350314, Grant 1464336, Grant 1553248, and Grant 1763657. This paper was presented in part at the 2016 Proceedings of Advances in Neural Information Processing Systems [1].

S. Dutta and P. Grover are with the Department of Electrical and Computer Engineering, Carnegie Mellon University, Pittsburgh, PA 15213 USA (e-mail: sanghamd@andrew.cmu.edu; pulkit@cmu.edu).

V. Cadambe is with the Department of Electrical Engineering, Pennsylvania State University, University Park, PA 16802 USA (e-mail: vxc12@engr.psu.edu).

Communicated by K. Narayanan, Associate Editor for Coding Techniques.

Color versions of one or more of the figures in this article are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TIT.2019.2927558

[1] Strassen's algorithm [13] and its generalizations offer a recursive approach to faster matrix multiplications over multiple processing nodes, but they are often not preferred because of their high communication cost [14].

the saved state in case of faults. However, roll-backward-error-correction mechanisms are expensive because they require rescheduling and repeating computations, and rebuilding states already reached.

An alternate approach is to use *roll-forward error correction* techniques. These techniques address straggling by introducing redundancy in the computational tasks across different processing nodes. The fusion node now requires outputs from only a subset of all the processing nodes to successfully finish the entire computation. The use of erasure coding for fault tolerant computing dates back to Algorithm-Based-Fault-Tolerance (ABFT) [15], [17]. Erasure codes are often found to offer advantages over replication (the latter is analyzed in [3]–[5]). Maximum Distance Separable (MDS) [18] codes were used in [2] to code matrices in one dimension for speeding up matrix operations. In [19], coding was used to speed up MapReduce [20], a commonly used framework for distributing and shuffling data while executing data-intensive tasks on distributed clusters. "Coded Computing" is now an emerging area of research ([1], [2], [19], [21]–[54]) which advances on coding approaches in classical works on Algorithm-Based Fault Tolerance (ABFT), and also provides novel analyses of required computation time ([2], [23], [44], [47]), including that in our preliminary version of this work in [1]. Following the conference publication [1], there have been several interesting works in the coded computing community (see [21] for survey) on coded linear algebra [22]–[37], coded convolutions [47], [48], gradient coding [38]–[42], coded Fast Fourier Transforms [55], [56], performance analysis [45], [46], coded neural networks [30], [57], coded iterative computing [58]–[61] etc.

A. Problem Formulation

We consider the problem of computing a matrix-vector product ($\boldsymbol{A}_{M\times N}\boldsymbol{x}_{N\times 1}$) using a set of P parallel or distributed processing nodes, under two constraints:

(i) The outputs from *any* K out of P nodes are sufficient to compute the original matrix-vector product.
(ii) Every processing node can only access a fraction $\frac{s}{N}$ of the long vector $\boldsymbol{x}$ during computation due to limitations of communication.[2]

We also assume that each of these nodes computes an equal number of dot products, i.e., the task allocated to each node is equal. Our *goal* is to minimize the fraction $\frac{s}{N}$ for a given K. Here K is the straggler *recovery threshold*, i.e. the worst-case number of processing nodes that need to finish to compute successfully. Thus, *any* $(P-K)$ stragglers can be tolerated.

As is the case in [2], $\boldsymbol{A}$ is encoded into a matrix $\boldsymbol{F}$, and parts of $\boldsymbol{F}$ are stored beforehand at the individual processing nodes. When $M < P$,[3] our problem reduces to the following: Find a matrix $\boldsymbol{F}$ for a given $\boldsymbol{A}$ such that:

(i) *Any* K rows of $\boldsymbol{F}$ can linearly span the row space of $\boldsymbol{A}$.
(ii) The number of non-zero entries in each row of $\boldsymbol{F}$ is constrained to be at most s, which we need to minimize.

When $M < P$, if there are no stragglers, we can choose $K = P$, i.e., wait for all the nodes to finish. One example is the uncoded strategy (see Section II and Fig. 2a) that simply distributes the task equally among the P nodes, and hence $s = \frac{MN}{P}$. However, this technique is bottlenecked by stragglers. The MDS-coding-based strategy in [2] has $K = M$. There, the M rows of $\boldsymbol{A}$ are encoded using a (P, M) MDS code into P rows such that *any* M rows can linearly span the row space of $\boldsymbol{A}$ (elaborated in Section II and Fig. 2c). This, however, requires $s = N$, i.e., the entire $\boldsymbol{x}$ is accessed by each node.

In this paper, we aim to derive fundamental limits on the trade-off between K and s, given the matrix dimensions M and N as well as the total number of nodes P. The key difference between our work and [2] is that we constrain that every processing node receive only a fraction $\frac{s}{N}$ of the long vector $\boldsymbol{x}$ during computation. We thus allow the size of individual dot products computed at each processing node to be smaller than N (the full length of the input) as each processing node only accesses a fraction of the vector $\boldsymbol{x}$.

B. Why Go for Shorter Dot Products?

Communication Cost Savings: In several applications, e.g., in machine learning inference for time-critical applications, the trained model matrix $\boldsymbol{A}$ is fixed and different test data vectors $\boldsymbol{x}$ keep arriving in real-time [9]. In systems where multicasting $\boldsymbol{x}$ is not possible or is costly, it may be faster to communicate a subset of the coordinates of $\boldsymbol{x}$ to each processing node.

Machine Learning Training and Gradient Coding: One typical scenario that arises in machine learning training, as also pointed out in [38], is when $\boldsymbol{A}$ is fixed and known in advance and is required to be multiplied with another vector or matrix that is generated in real-time. In fact, the problem of gradient coding [38]–[41] is deeply connected with our problem when $M = 1$, $P = N$, and $\boldsymbol{A}$ is simply the vector $[1, 1, \ldots, 1]_{1\times N}$. The dataset is divided into N partitions. At each iteration of training, the master node requires the vector $[1, 1, \ldots, 1]_{1\times N}$ to be multiplied with a matrix whose every row consists of gradients from a different partition of the dataset, i.e.,

$$\boldsymbol{G} = \begin{bmatrix} \boldsymbol{g}_1^T \\ \vdots \\ \boldsymbol{g}_N^T \end{bmatrix}.$$

The goal is to generate a set of $P(= N)$ sparse vectors (essentially rows of $\boldsymbol{F}$) from $[1, 1, \ldots, 1]_{1\times N}$ such that any subset of them of size K can linearly span $[1, 1, \ldots, 1]_{1\times N}$. Because these generated $P(= N)$ vectors are sparse, the i-th processor only stores and computes gradients on the data partitions indexed in the support set of the i-th row of $\boldsymbol{F}$, and sends the product of the i-th row of $\boldsymbol{F}$ with $\boldsymbol{G}$ to the master node. The master node only waits for any K nodes to finish. Thus, Short-Dot codes apply to the gradient coding problem.

[2] We do not allow for any encoding on $\boldsymbol{x}$ as it requires access to all the values of $\boldsymbol{x}$.

[3] For the regime where $M \geq P$, the matrix $\boldsymbol{A}$ could be split horizontally into $\frac{M}{m}$ smaller sub-matrices of size $m \times N$ such that $m \leq P$, and the problem can be considered for each of those matrices separately, as we discuss in Section I-C.

In this work, we consider a more general version of the gradient coding problem where $M \geq 1$. For $M > 1$, a straightforward extension of the gradient coding strategy proposed in [38], [39] would encode each row of $\boldsymbol{A}$ separately and require M dot products, each of length $\frac{N(P-K+1)}{P}$, at each processing node. Instead, Short-Dot uses a novel "joint" encoding across rows that only requires a single dot product of length $\frac{N(P-K+M)}{P}$ at each processing node, while still requiring the same number of processing nodes (any K out of P) to finish. In terms of computational complexity at each node, note that $\frac{N(P-K+M)}{P}$ is less than $M \times \frac{N(P-K+1)}{P}$. Further, we also provide a tighter converse for $M > 1$ that proves that Short-Dot is *near-optimal*. It is worth noting that [38], [39] also introduce the notion of partial stragglers, which is outside the scope of our paper.

Computation and Memory Limited Architectures: Another reason for shorter dot products is that the computation time depends on the length of the dot products being computed. Processing nodes are also inherently memory limited, which limits the size of dot products that can be computed. In Section V we show that even in the absence of sparsity/communication constraints, Short-Dot can reduce the expected computation time (under model assumptions inspired from [2]) over existing strategies in a straggler-prone environment due to computation time savings.

Distributed Sensing and Inference: Sparsity constraints on the encoded matrix $\boldsymbol{F}_{P\times N}$ could also arise from practical constraints, e.g., when sensing using sensors with a limited field of view [62]. This problem could arise in Cognitive IoT systems, that typically perform some inference task on the observations, e.g. pattern recognition, classification etc. using a distributed network of cognitive sensors with a limited field of view. The goal of the IoT sensors is to compute a dimension-reducing projection on the sensed data, which is a critical first step in many machine-learning algorithms. Under such scenarios, it might be required to impose a pre-specified sparsity pattern on the final encoded matrix $\boldsymbol{F}$ to obtain a sparse sensing matrix for the IoT devices. This could be used to architect systems using fault-prone IoT sensing and communication networks that nevertheless provide reliable inference.

Error Correction for Emerging Hardware: Another interesting example comes from recent work on designing computational devices that exclusively compute dot products using analog processing units [6], [7], also referred to as "dot product nanofunctions." These devices are prone to increased errors when designed for longer dot products. Short-Dot is directly applicable in this scenario, as it allows the size of each individual dot product to be shorter than the size of the input. A related work in this direction, that also proposes error correcting codes for fault-prone "dot product engines," is [63].

C. Our Contributions

We propose a new error correcting code – Short-Dot – for computing parallel or distributed matrix-vector products (i.e., $\boldsymbol{A}_{M\times N}\boldsymbol{x}_{N\times 1}$) under stragglers when each processing node only has access to a fraction $\frac{s}{N}$ of the input vector $\boldsymbol{x}$. The key novelty is that we allow for shorter dot products to be computed at each individual node, thus requiring only a fraction of $\boldsymbol{x}$ as compared to existing techniques in [2], [24] that require the full vector $\boldsymbol{x}$ to be communicated to each node.

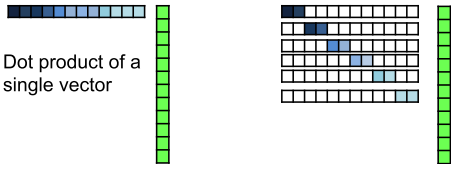
When the number of dot products is less than the number of nodes, i.e., $M < P$:

- For this case, Short-Dot encodes the given matrix $\boldsymbol{A}_{M\times N}$ into $\boldsymbol{F}_{P\times N}$ such that each N-length row of $\boldsymbol{F}$ has at most $s = \frac{N(P-K+M)}{P}$ non-zero elements. Each of the P processing nodes store a different row of $\boldsymbol{F}_{P\times N}$ and compute its dot product with $\boldsymbol{x}$. Because the locations of the s non-zero entries in a row of $\boldsymbol{F}$ are known by design, every processing node only requires those particular s entries of $\boldsymbol{x}$, and not the whole vector $\boldsymbol{x}$, and also has a reduced computational complexity of $\Theta(s)$ instead of $\Theta(N)$ at each node. Here $K = \kappa(s)$ parameterizes the worst-case recovery threshold under stragglers as a function of s. The dot products of any K out of the P rows of $\boldsymbol{F}$ with $\boldsymbol{x}$ are sufficient to recover $\boldsymbol{Ax}$. In other words, any K rows of $\boldsymbol{F}$ can be linearly combined to generate all the rows of $\boldsymbol{A}$.
- We show that the uncoded parallelization strategy and MDS coding strategy [2] are actually two special cases of Short-Dot with $K = \kappa\left(\frac{MN}{P}\right) = P$ and $K = \kappa(N) = M$ respectively. Short-Dot allows for K to vary over the range M to P, and accordingly allows the length of the dot product s to reduce over the range N to $\frac{N}{P}$.
- We provide fundamental limits (converse results) on the trade-off between the length of the dot products (s) and the recovery threshold (number of nodes to wait for, i.e., $K = \kappa(s)$) for *any* such strategy in Section IV. Our converse shows that Short-Dot is near-optimal in the regime ($M < P$). This is an improvement over our previous result [1].
- Even in the absence of any sparsity/communication constraints on the processing nodes, we show that Short-Dot can still be useful in speeding up linear computations for $M < P$. Assuming exponentially distributed computation times (used in [2]), Short-Dot outperforms competing strategies, namely, uncoded parallelization, replication strategy and MDS coding strategy (see Section II) not only in communication, but also in *computation time* under straggling. We derive the expected computation time required by our strategy and compare it to uncoded parallelization, replication and MDS coding (see Fig. 2) demonstrating that Short-Dot is universally faster over the entire range of $M < P$. We also explicitly show a regime ($M = \frac{P}{\log P}$) where Short-Dot outperforms all competing strategies in expected computation time by a factor of $\frac{\log(P)}{\log(\log P)}$, that can even diverge to infinity.

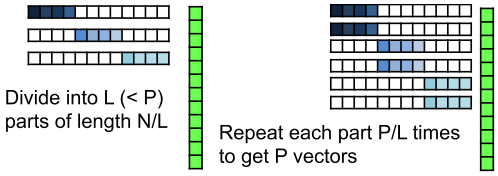
We note that in a follow-up work [23], a new converse has been obtained for this problem that improves upon the converses in this work and more importantly, proves the exact optimality of Short-Dot codes.

When the number of dot products is greater than the number of nodes, i.e., $M \geq P$:

For the regime where $M \geq P$, the same encoding technique can be applied by first dividing the matrix $\boldsymbol{A}$ into smaller sub-matrices of size $m \times N$ such that $m \leq P$, and applying



(a) Uncoded parallelization: The vector $\boldsymbol{a}^T$ is divided into P equal parts, one for each node. Each node computes a short dot product of length N/P. The fusion node has to wait for all the nodes to finish.



(b) Replication with block partitioning: The vector $\boldsymbol{a}^T$ is divided into L equal parts where $L < P$, and each part is replicated P/L times to generate P vectors, one for each node. Each node computes a dot product of length N/L. The fusion node waits for one replica of each part to finish.

Fig. 1. Comparison between uncoded parallelization and replication with block partitioning: A dot product ($\langle \boldsymbol{a}, \boldsymbol{x} \rangle = \boldsymbol{a}^T \boldsymbol{x}$) of length $N = 12$ is being computed using the same of processing nodes ($P = 6$) for both the strategies.

Short-Dot on each of these sub-matrices separately. Short-Dot (with $s < N$) still has gains over MDS coding strategy (Short-Dot with $s = N$) for communication-constrained regimes because MDS coding strategy requires the entire vector $\boldsymbol{x}$ to be accessed by each node. However, in the absence of communication constraints, MDS coding strategy may be preferred because it has lower computational complexity per-node (see Section VIII-A for a detailed discussion).

We also show that if we move from an erasure model to an error model where some processing nodes are faulty and we do not know which of those are faulty, Short-Dot with straggler recovery threshold of K can correct $\lfloor \frac{P-K}{2} \rfloor$errors.

D. Some Notations and Definitions

Sets of indices are denoted by calligraphic fonts, e.g., $\mathcal{S}$, $\mathcal{U}$, $\mathcal{V}$, $\mathcal{W}$, $\mathcal{X}$ etc. Note that, for integers a and b where $b > a$, we also use the notation $a : b$ to denote the set of all the integers from a to b including them, i.e., $\{a, a+1, a+2, \ldots, b\}$.

We denote matrices and vectors in bold. A vector $\boldsymbol{x} \in \mathbb{R}^N$ refers to a column vector of dimensions $N \times 1$. For two column vectors $\boldsymbol{a}$ and $\boldsymbol{x}$ of the same dimensions, the dot product $\langle \boldsymbol{a}, \boldsymbol{x} \rangle$ is defined as the product $\boldsymbol{a}^T \boldsymbol{x}$. For ease of explanation, a matrix $\boldsymbol{A}_{(M \times N)}$ may be partitioned column-wise as $\boldsymbol{A}_{(M \times N)} = [\boldsymbol{A}_1 \boldsymbol{A}_2 \ldots \boldsymbol{A}_N]$ where $\boldsymbol{A}_j$ is the j-th column of $\boldsymbol{A}_{(M \times N)}$ or partitioned row-wise as

$$\boldsymbol{A}_{(M \times N)} = \begin{bmatrix} \boldsymbol{a}_1^T \\ \vdots \\ \boldsymbol{a}_M^T \end{bmatrix},$$

where $\boldsymbol{a}_i^T$ the i-th row vector of $\boldsymbol{A}_{(M \times N)}$.

$\boldsymbol{x}^{\mathcal{S}}$ (or $\boldsymbol{A}^{\mathcal{S}}$) denotes a sub-vector (or sub-matrix) consisting of the rows of the vector $\boldsymbol{x}$ (or matrix $\boldsymbol{A}$) indexed by the set $\mathcal{S}$. Similarly, we use the notation $\boldsymbol{A}_{col\ \mathcal{S}}$ to denote a sub-matrix of $\boldsymbol{A}$ consisting of only the columns of $\boldsymbol{A}$ indexed by set $\mathcal{S}$. We define the sparsity of a vector $\boldsymbol{x} \in \mathbb{R}^N$ as the number of non-zero elements in the vector, i.e., $\|\boldsymbol{x}\|_0 = \sum_{j=1}^{N} \mathbb{I}(x_j \neq 0)$ where $\mathbb{I}(\cdot)$ denotes the indicator function. Let $f(n)$ and $g(n)$ be two functions of n. The function $f(n) = \mathcal{O}(g(n))$ if there exists an n_0 and a constant c such that for all $n > n_0$, $f(n) \leq cg(n)$. Similarly, $f(n) = o(g(n))$ if for any chosen $\epsilon > 0$, one can find an n_0 such that for all $n > n_0$, $f(n) \leq \epsilon g(n)$. Lastly, $f(n) = \Theta(g(n))$ if $f(n) = \mathcal{O}(g(n))$ and $g(n) = \mathcal{O}(f(n))$.

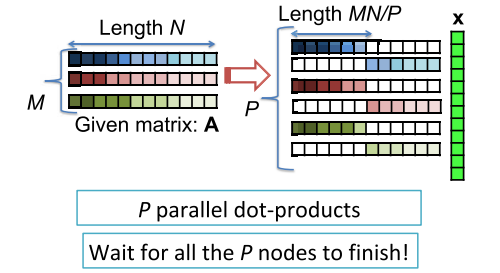
E. Paper Organization

The rest of the paper is organized as follows. Section II provides an overview of existing strategies for computing dot products, followed by Section III which formally introduces our proposed strategy and the main achievability results. Section IV provides our converse results. Section V analyzes the expected computation time of Short-Dot and compares it with existing strategies. Section VII discusses the extension of Short-Dot codes from correcting erasures to correcting errors, followed by a concluding discussion in Section VIII.

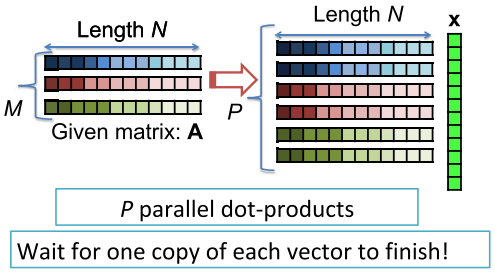
II. Overview of Existing Strategies for Computing Dot Products

In this section, we provide an overview of several existing strategies for computing dot products in a parallel or distributed scenario. To begin with, let us first consider the problem of computing a single dot product of an input vector $\boldsymbol{x} \in \mathbb{R}^N$ with a pre-specified vector $\boldsymbol{a} \in \mathbb{R}^N$. By an "uncoded" parallelization strategy (which includes Block Striped Decomposition [11]), we mean a strategy that does not use redundancy to overcome delays caused by stragglers. One uncoded strategy is to partition the dot product into P smaller dot products, where P is the number of available processing nodes. E.g. $\boldsymbol{a}$ can be divided into P parts, constructing P short vectors of sparsity $\frac{N}{P}$, with each vector stored in a different processing node as shown in Fig. 1a. Only the non-zero values of the vector need to be stored since the locations of the non-zero values is known apriori at every node. One might expect the computation time for each processing node to reduce by a factor of P. However, now the fusion node has to wait for all the P processing nodes to finish their computation, and the stragglers can now delay the entire computation. Can we construct P vectors such that dot products of a subset of them with $\boldsymbol{x}$ are sufficient to compute $\langle \boldsymbol{a}, \boldsymbol{x} \rangle$? A simple strategy of introducing redundancy is using replication with block partitioning i.e., constructing L vectors of sparsity N/L by partitioning the vector of length N into L parts ($L < P$), and replicating the L vectors P/L times so as to obtain P vectors of sparsity N/L as shown in Fig. 1b. For each of the L parts of the vector, the fusion node only needs the output of one processing node among all its replicas.

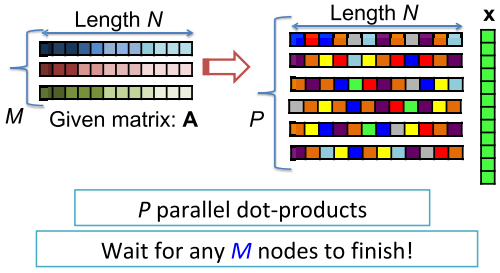
Instead of a single dot product, if one requires the dot product of $\boldsymbol{x}$ with M vectors $\{\boldsymbol{a}_1, \ldots, \boldsymbol{a}_M\}$, one can simply



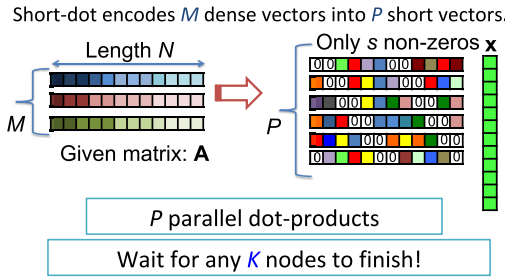
(a) Uncoded parallelization strategy



(b) Replication strategy



(c) MDS coding strategy



(d) Short-Dot coding strategy

Fig. 2. Comparison between different strategies of parallelization: Here $M = 3$ dot products of length $N = 12$ are being computed using $P = 6$ processing nodes.

repeat the aforementioned uncoded strategy M times as shown in Fig. 2a. For multiple dot products, an alternative replication-based strategy is to compute M dot products P/M times in parallel at different processing nodes. Now we only have to wait for at least one processing node corresponding to each of the M vectors to finish (see Fig. 2b). Improving upon replication, it is shown in [2] that a (P, M)-MDS code allows constructing P coded vectors such that any M of P dot products can be used to reconstruct all the M original vectors (see Fig. 2c).

Can We Go Beyond MDS Codes?

The MDS coding strategy [2] requires N-length dot products to be computed on each processing node. Short-Dot instead constructs P vectors of sparsity s ($\leq N$), such that the dot product of $\boldsymbol{x}$ with any K ($\geq M$) out of these P short vectors is sufficient to compute the dot product of $\boldsymbol{x}$ with all the M given vectors (see Fig. 2d). Compared to a straightforward application of MDS codes for this problem, Short-Dot codes are more flexible as they allow each processing node to compute a shorter dot product at the cost of waiting for some more processing nodes (since $K \geq M$), thus resulting in a trade-off. Short-Dot also potentially reduces the communication cost since only a shorter portion of the input vector is required to be communicated to each processing node. We also propose Short-MDS, an extension of the MDS coding strategy in [2], to create short dot products of length s, through block partitioning the matrix $\boldsymbol{A}$ and subsequent encoding of each block separately. We show that Short-MDS is a special case of Short-Dot in Theorem 2 (Section III-D) in the sense that Short-Dot codes with the same final sparsity pattern as Short-MDS codes also have the same recoverability properties that we elaborate on further in Section III-D. Furthermore, when $\frac{N}{s}$ is not an integer, one can also design Short-Dot codes with other sparsity patterns that require the decoder to wait for fewer processing nodes in worst case than Short-MDS codes as discussed in Section III-D.

III. Our Coded Parallelization Strategy: Short-Dot

In this section, we introduce our strategy of computing the matrix-vector product $\boldsymbol{Ax}$ where $\boldsymbol{x} \in \mathbb{R}^N$ is the input column vector and $\boldsymbol{A}_{(M\times N)} = [\boldsymbol{a}_1, \boldsymbol{a}_2, \ldots, \boldsymbol{a}_M]^T$ is a given matrix. First we informally introduce our coding technique in Section III-A. The main achievability result is provided in Section III-B, followed by the algorithm in Section III-C. We also propose an alternate coding strategy called Short-MDS codes in Section III-D.

A. Basic Construction Strategy

Short-Dot constructs a $P \times N$ matrix $\boldsymbol{F} = [\boldsymbol{f}_1, \boldsymbol{f}_2, \ldots, \boldsymbol{f}_P]^T$ such that the following two conditions are satisfied:

(i) Given *any* K (where $M \leq K \leq P$) rows of $\boldsymbol{F}$, there always exists M predetermined linear combinations of these rows that can generate each of the M row vectors $\{\boldsymbol{a}_1^T, \ldots, \boldsymbol{a}_M^T\}$. This essentially means *any* K rows of $\boldsymbol{F}$ should linearly span the row space of $\boldsymbol{A}$.

(ii) Any row of $\boldsymbol{F}$ has sparsity (number of non-zeros) at most $s = \frac{N}{P}(P - K + M)$.

Once such a matrix $\boldsymbol{F}$ is constructed, each sparse row of $\boldsymbol{F}$ (say $\boldsymbol{f}_i^T$) is sent to the i-th processing node ($i = 1, \ldots, P$) and dot products of $\boldsymbol{x}$ with all sparse rows are computed in parallel. Let $\mathcal{S}_i$ denote the support (set of non-zero indices) of $\boldsymbol{f}_i$. Thus, for any unknown vector $\boldsymbol{x}$, short dot products of length $|\mathcal{S}_i| \leq s = \frac{N}{P}(P - K + M)$ are computed on each processing node. Since the linear combination of any K rows of $\boldsymbol{F}$ can generate the rows of $\boldsymbol{A}$, i.e., $\{\boldsymbol{a}_1^T, \boldsymbol{a}_2^T, \ldots, \boldsymbol{a}_M^T\}$, the dot product from the earliest K out of P processing nodes can be linearly combined to obtain the linear transform $\boldsymbol{Ax}$. The strategy is pictorially illustrated in Fig. 3.

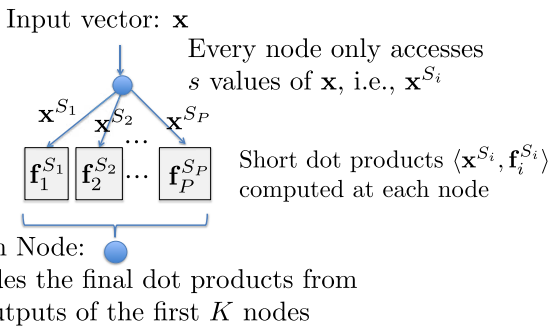
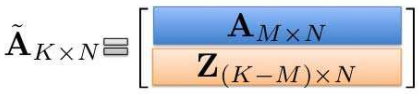
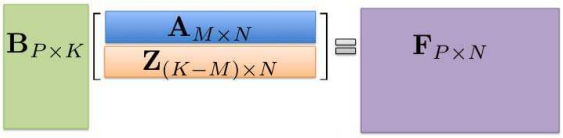


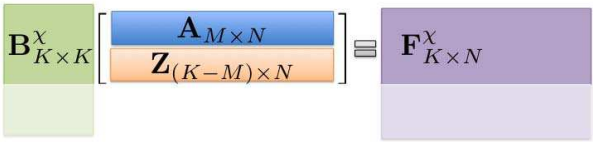
Fig. 3. Task flow for Short-Dot coding strategy: Each of the P nodes can only access any s values of the vector $\boldsymbol{x}$. Using Short-Dot codes, node i only accesses $\boldsymbol{x}^{\mathcal{S}_i}$, and computes short dot products $\langle \boldsymbol{x}^{\mathcal{S}_i}, \boldsymbol{f}_i^{\mathcal{S}_i} \rangle$, such that outputs from any K out of P processing nodes are sufficient to compute successfully.



(a) Create $\tilde{\boldsymbol{A}}$ by appending $(K-M)$ extra rows below $\boldsymbol{A}$. The choice of these extra rows will be discussed later in enforcing sparsity but for now assume that $\boldsymbol{Z}$ can be any matrix.



(b) Perform the following encoding: $\boldsymbol{F} = \boldsymbol{B}\tilde{\boldsymbol{A}}$. Here $\boldsymbol{B}$ is a chosen $P \times K$ matrix such that all $K \times K$ square sub-matrices are invertible. This ensures that *any* K rows of $\boldsymbol{F}$ can span the rows of $\boldsymbol{A}$.

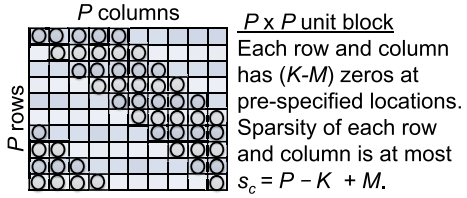


(c) Observation: Pick any K rows of $\boldsymbol{F}$, i.e. a sub-matrix $\boldsymbol{F}^{\chi}$ as shown here. All the rows of $\boldsymbol{A}$ can be generated from the rows of $\boldsymbol{F}^{\chi}$ as $\boldsymbol{B}^{\chi}$ is invertible.

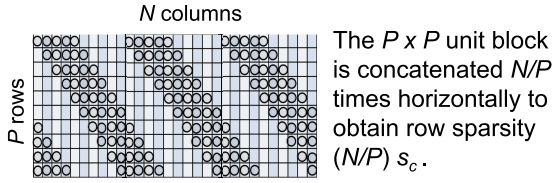
Fig. 4. Illustration of encoding scheme to satisfy the condition that *any* K rows of $\boldsymbol{F}$ can span the rows of $\boldsymbol{A}$

Before formally stating our algorithm and main results, we first provide an insight into why such a matrix $\boldsymbol{F}$ exists and develop an intuition into the construction strategy. Recall that the first condition that needs to be satisfied by $\boldsymbol{F}$ is as follows: *any* K rows of $\boldsymbol{F}$ should linearly span the row space of $\boldsymbol{A}$. We satisfy this condition using the encoding strategy, as shown in Fig. 4.

Now that the first condition is satisfied, we will discuss how to choose $\boldsymbol{Z}$ so as to satisfy the second condition: Any row of $\boldsymbol{F}$ has sparsity (number of non-zeros) at most $s = \frac{N(P-K+M)}{P}$. Let us choose one such sparsity pattern for $\boldsymbol{F}$ that has at most $s = \frac{N(P-K+M)}{P}$ (also shown in Fig. 5).



(a) A $P \times P$ unit block with row sparsity $P - K + M$.



(b) The unit block is concatenated $\frac{N}{P}$ times to form an $N \times P$ matrix $\boldsymbol{F}$ with row sparsity $s = \frac{N}{P}(P - K + M)$.

Fig. 5. Example of a sparsity pattern of $\boldsymbol{F}$ achievable by the Short-Dot coding strategy.

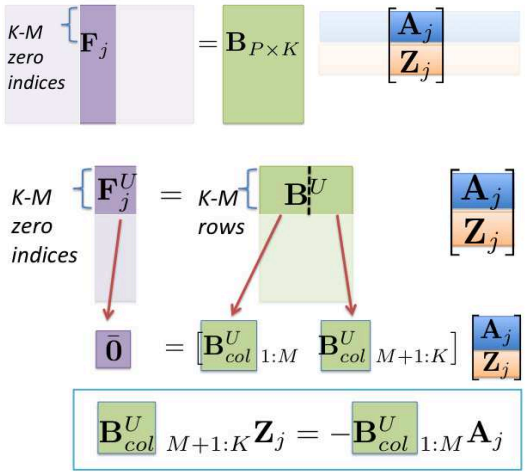


Fig. 6. A pictorial illustration showing that choosing $\boldsymbol{Z}_j = -[\boldsymbol{B}^{\mathcal{U}}_{col\ M+1:K}]^{-1}\ \boldsymbol{B}^{\mathcal{U}}_{col\ 1:M}\boldsymbol{A}_j$ for every column j ensures the desired sparsity pattern on $\boldsymbol{F}$.

Observe that in this pattern, every column of $\boldsymbol{F}$ has at least $(K-M)$ zeros. For any column $\boldsymbol{F}_j$, let $\mathcal{U}$ denote the set of indices that are zero. Having fixed $\boldsymbol{B}$, the column $\boldsymbol{F}_j$ is an affine linear combination of the entries of $\boldsymbol{Z}_j$. Thus choosing $\boldsymbol{Z}_j = -[\boldsymbol{B}^{\mathcal{U}}_{col\ M+1:K}]^{-1}\ \boldsymbol{B}^{\mathcal{U}}_{col\ 1:M}\boldsymbol{A}_j$ for every column j ensures that the entries of $\boldsymbol{F}_j$ indexed by $\mathcal{U}$ are set to zero (see Fig. 6) as long as the matrix $\boldsymbol{B}^{\mathcal{U}}_{col\ M+1:K}$ is invertible.

Now we formally state our main technical result.

B. Achievability Result

Theorem 1 (Short-Dot Achievability)**.** *Given an $M \times N$ matrix $\boldsymbol{A} = [\boldsymbol{a}_1, \boldsymbol{a}_2, \ldots, \boldsymbol{a}_M]^T$, there always exists a $P \times N$ matrix $\boldsymbol{F}$ such that the following two conditions hold:*

(i) Any K $(M \leq K \leq P)$ rows of the matrix $\boldsymbol{F}$ can be linearly combined to generate the row vectors $\{\boldsymbol{a}_1^T, \boldsymbol{a}_2^T, \ldots, \boldsymbol{a}_M^T\}$.

(ii) Each row of $\boldsymbol{F}$ *has sparsity at most* $s = \frac{N(P-K+M)}{P}$, *provided* P *divides* N.

To prove this theorem, we first prove the following lemma.

Lemma 1. *Let* $\boldsymbol{F} = \boldsymbol{B}\widetilde{\boldsymbol{A}}$ *where* $\widetilde{\boldsymbol{A}}$ *is a* $K \times N$ *matrix and* $\boldsymbol{B}$ *is any* $(P \times K)$ *matrix such that every square sub-matrix is invertible.*[4] *Then, any* K *rows of* $\boldsymbol{F}$ *can be linearly combined to generate any row of* $\widetilde{\boldsymbol{A}}$.

Proof of Lemma 1. Choose an arbitrary index set $\mathcal{X} \subset \{1, 2, \ldots, P\}$ such that $|\mathcal{X}| = K$. Let $\boldsymbol{F}^{\mathcal{X}}$ be the sub-matrix formed by the K rows of $\boldsymbol{F}$ indexed by $\mathcal{X}$. Then, $\boldsymbol{F}^{\mathcal{X}} = \boldsymbol{B}^{\mathcal{X}}\widetilde{\boldsymbol{A}}$. Now, $\boldsymbol{B}^{\mathcal{X}}$ is a $K \times K$ sub-matrix of $\boldsymbol{B}$, and is thus invertible. Thus, $\widetilde{\boldsymbol{A}} = (\boldsymbol{B}^{\mathcal{X}})^{-1}\boldsymbol{F}^{\mathcal{X}}$. The i-th row of $\widetilde{\boldsymbol{A}}$ is generated as $[i\text{-th Row of } (\boldsymbol{B}^{\mathcal{X}})^{-1}]\boldsymbol{F}^{\mathcal{X}}$ for $i = 1, 2, \ldots, K$. Thus, each row of $\widetilde{\boldsymbol{A}}$ is generated by the chosen K rows of $\boldsymbol{F}$. □

Proof of Theorem 1. We may append $(K - M)$ rows below $\boldsymbol{A} = [\boldsymbol{a}_1, \boldsymbol{a}_2, \ldots, \boldsymbol{a}_M]^T$, to form a $K \times N$ matrix given by: $\widetilde{\boldsymbol{A}} = \begin{bmatrix} \boldsymbol{A} \\ \boldsymbol{Z} \end{bmatrix}$ where $\boldsymbol{Z}$ is of dimension $(K-M)\times N$. The precise choice of these additional row vectors will be made explicit later. Next, we choose $\boldsymbol{B}$, a $P \times K$ matrix such that *any square sub-matrix of* $\boldsymbol{B}$ *is invertible*. Using Lemma 1, we thus show that *any* K rows of the matrix $\boldsymbol{B}\widetilde{\boldsymbol{A}}$ are sufficient to linearly generate any row of $\widetilde{\boldsymbol{A}}$, which happens to include the rows of $\boldsymbol{A}$, i.e., $\{\boldsymbol{a}_1^T, \boldsymbol{a}_2^T, \ldots, \boldsymbol{a}_M^T\}$.

Now, we show how the row sparsity of $\boldsymbol{F}$ can be constrained to be at most $\frac{N(P-K+M)}{P}$ by appropriately choosing the appended matrix $\boldsymbol{Z}$. We select a sparsity pattern that we want to enforce on $\boldsymbol{F}$ and then show that there exists a choice of the appended matrix $\boldsymbol{Z}$ such that the pattern can be enforced.

Example of a sparsity pattern enforced on $\boldsymbol{F}$: This is illustrated in Fig. 5. First, we construct a $P \times P$ "unit block" with a cyclic structure of non-zero entries, where $(K - M)$ zeros in each row and column are arranged as shown in Fig. 5. Each row and column have $K - M$ pre-specified zeros, and hence $s_c = P - K + M$ non-zero entries. This unit block is replicated along the column dimension $\frac{N}{P}$ times to form an $P \times N$ matrix with s_c non-zero entries in each column, and and $s = Ns_c/P$ non-zero entries in each row. We now show how choice of $\boldsymbol{Z}$ can enforce this pattern on $\boldsymbol{F}$.

From $\boldsymbol{F} = \boldsymbol{B}\widetilde{\boldsymbol{A}}$, the j-th column of $\boldsymbol{F}$ can be written as,

$$\boldsymbol{F}_j = \boldsymbol{B}\widetilde{\boldsymbol{A}}_j. \tag{1}$$

Each column of $\boldsymbol{F}$ has $K - M$ pre-specified zeros at locations indexed by $\mathcal{U} \subset \{1, 2, \ldots, P\}$. Let $\boldsymbol{B}^{\mathcal{U}}$ denote a $((K-M)\times K)$ sub-matrix of $\boldsymbol{B}$ consisting of the rows of $\boldsymbol{B}$ indexed by $\mathcal{U}$. Thus,

$$\boldsymbol{B}^{\mathcal{U}}\widetilde{\boldsymbol{A}}_j = [\boldsymbol{0}]_{(K-M)\times 1}. \tag{2}$$

Divide $\widetilde{\boldsymbol{A}}_j$ into two portions of lengths M and $K - M$ as follows: $\widetilde{\boldsymbol{A}}_j = \begin{bmatrix} \boldsymbol{A}_j \\ \boldsymbol{Z}_j \end{bmatrix}$. Thus,

$$\begin{aligned} \boldsymbol{B}^{\mathcal{U}}\widetilde{\boldsymbol{A}}_j &= \begin{bmatrix} \boldsymbol{B}^{\mathcal{U}}_{col\ 1:M} & \boldsymbol{B}^{\mathcal{U}}_{col\ M+1:K} \end{bmatrix} \begin{bmatrix} \boldsymbol{A}_j \\ \boldsymbol{Z}_j \end{bmatrix} \\ &= \boldsymbol{B}^{\mathcal{U}}_{col\ 1:M}\boldsymbol{A}_j + \boldsymbol{B}^{\mathcal{U}}_{col\ M+1:K}\ \boldsymbol{Z}_j \\ &= [\boldsymbol{0}]_{(K-M)\times 1}. \end{aligned} \tag{3}$$

[4]This condition is relaxed in Remark 1.

This implies,

$$\boldsymbol{B}^{\mathcal{U}}_{col\ M+1:K}\ \boldsymbol{Z}_j = -\boldsymbol{B}^{\mathcal{U}}_{col\ 1:M}[\boldsymbol{A}_j] \tag{4}$$

$$\Rightarrow \boldsymbol{Z}_j = -(\boldsymbol{B}^{\mathcal{U}}_{col\ M+1:K})^{-1}\ \boldsymbol{B}^{\mathcal{U}}_{col\ 1:M}[\boldsymbol{A}_j] \tag{5}$$

where the last step uses the fact that $[\boldsymbol{B}^{\mathcal{U}}_{col\ M+1:K}]$ is invertible because it is a $(K - M) \times (K - M)$ square sub-matrix of $\boldsymbol{B}$ and all square sub-matrices of $\boldsymbol{B}$ are invertible. This explicitly provides the vector $\boldsymbol{Z}_j$ which completes the j-th column of $\widetilde{\boldsymbol{A}}$ and ensures the sparsity of the j-th column of $\boldsymbol{F}$. The other columns of $\widetilde{\boldsymbol{A}}$ can be completed similarly. □

Corollary 1 (Achievable Sparsity Patterns)**.** *Given an* $M \times N$ *matrix* $\boldsymbol{A} = [\boldsymbol{a}_1, \boldsymbol{a}_2, \ldots, \boldsymbol{a}_M]^T$, *Short-Dot codes with recovery threshold* K *can be used to generate a* $P \times N$ *matrix* $\boldsymbol{F}$ *with any pre-specified sparsity pattern where every column of* $\boldsymbol{F}$ *has* $K - M$ *zeros.*

Proof of Corollary 1. The proof follows trivially from Theorem 1, as it is shown that any arbitrary $(K - M)$ indices of each column can be set as 0 irrespective of the other columns. □

Remark 1 (Relaxed conditions on matrix $\boldsymbol{B}$)**.** *It has been stated in Lemma 1 that all square sub-matrices of* $\boldsymbol{B}$ *need to be invertible. A matrix with i.i.d. Gaussian entries can be shown to satisfy this property with probability* 1. *In fact the condition on* $\boldsymbol{B}$ *in Lemma 1 can be relaxed, as evident from the proof. For matrix* $\boldsymbol{B}_{P\times K}$ *we only need two conditions.*

- *All* $K \times K$ *square sub-matrices are invertible.*
- *All* $(K - M) \times (K - M)$ *square sub-matrices in the last* $K - M$ *columns of* $\boldsymbol{B}$ *are invertible.*

A Vandermonde Matrix satisfies both these properties and thus can be used for encoding in Short-Dot.

With this insight in mind, we now formally state our algorithm.

C. Algorithm

Algorithm 1 Short-Dot (for a Specific Sparsity Pattern)

[A] Pre-Processing Step: Encode $\boldsymbol{F}$ (Performed Offline)
Given: Recovery threshold K, Encoding matrix $\boldsymbol{B}_{P\times K}$, and $\boldsymbol{A}_{M\times N} = [\boldsymbol{a}_1, \ldots, \boldsymbol{a}_M]^T = [\boldsymbol{A}_1, \boldsymbol{A}_2, \ldots, \boldsymbol{A}_N]$.

For $j = 1$ *to* N **do:**
  Set $\mathcal{U} \leftarrow (\{(j-1), \ldots, (j+K-M-1)\} \bmod P) + 1$
    ▷ The indices that are 0 for the j-th column of $\boldsymbol{F}$.
  Set $\boldsymbol{B}^{\mathcal{U}} \leftarrow$ Rows of $\boldsymbol{B}$ indexed by $\mathcal{U}$
  Set $\boldsymbol{Z}_j = -(\boldsymbol{B}^{\mathcal{U}}_{col\ M+1:K})^{-1}\ \boldsymbol{B}^{\mathcal{U}}_{col\ 1:M}[\boldsymbol{A}_j]$
    ▷ $\boldsymbol{Z}_j$ is a $(K - M) \times 1$ column vector.
  Set $\boldsymbol{F}_j = \boldsymbol{B}\begin{bmatrix} \boldsymbol{A}_j \\ \boldsymbol{Z}_j \end{bmatrix}$
    ▷ $\boldsymbol{F}_j$ is a $P \times 1$ column vector (j-th col of $\boldsymbol{F}$)
Encoded Output: $\boldsymbol{F}_{P\times N} = [\boldsymbol{f}_1 \boldsymbol{f}_2 \cdots \boldsymbol{f}_P]^T$
    ▷ Row representation of matrix $\boldsymbol{F}$.
For $i = 1$ *to* P **do:**
  Store $\mathcal{S}_i \leftarrow \text{Support}(\boldsymbol{f}_i)$
    ▷ Indices of non-zero entries in the i-th row of $\boldsymbol{F}$
  Send $\boldsymbol{f}_i^{\mathcal{S}_i}$ to i-th processing node

[B] Online computations
External Input : $\boldsymbol{x}$
Resources: P parallel processing nodes ($P > M$)
[B1] Parallelization Strategy: Divide task among parallel processing nodes:
1: **For** $i = 1$ *to* P **do:**
2: Access $\boldsymbol{x}^{\mathcal{S}_i}$ at the i-th processing node
3: Compute at i-th processing node: $\langle \boldsymbol{f}_i^{\mathcal{S}_i}, \boldsymbol{x}^{\mathcal{S}_i} \rangle$
Output: $\langle \boldsymbol{f}_i^{\mathcal{S}_i}, \boldsymbol{x}^{\mathcal{S}_i} \rangle$ from K nodes that finish first.
[B2] Fusion Node: Decode the dot products from the processing node outputs:
1: **Set** $\mathcal{X} \leftarrow$ Indices of the K nodes that finished first
2: **Set** $\boldsymbol{B}^{\mathcal{X}} \leftarrow$ Rows of $\boldsymbol{B}$ indexed by $\mathcal{X}$
3: **Set** $\boldsymbol{v}_{K\times 1} \leftarrow \boldsymbol{F}^{\mathcal{X}}\boldsymbol{x}$
$\triangleright$ $K \times 1$ column vector of outputs $\langle \boldsymbol{f}_i^{\mathcal{S}_i}, \boldsymbol{x}^{\mathcal{S}_i} \rangle \ \forall \ i \ \in \mathcal{X}$
4: **Set** $\boldsymbol{Ax} = [\langle \boldsymbol{a}_1, \boldsymbol{x}\rangle, \ldots, \langle \boldsymbol{a}_M, \boldsymbol{x}\rangle]^T \leftarrow [(\boldsymbol{B}^{\mathcal{X}})^{-1}]^{1:M}\boldsymbol{v}$
5: **Output:** $\langle \boldsymbol{a}_1, \boldsymbol{x}\rangle, \ldots, \langle \boldsymbol{a}_M, \boldsymbol{x}\rangle$

Remark 2 (Best-case recovery)**.** *Note that, the recovery threshold (K) is formally defined as the number of nodes that the decoder has to wait for in the worst-case, i.e., the minimum K for which* ***all*** *subsets of K nodes become sufficient to reconstruct the results, analogous to standard coding theory. In the best-case, there maybe (see Theorem 2 in Section III-D) some specific subsets of less than K nodes that are sufficient to reconstruct the entire result if those nodes finish first, even though not all subsets of that size are sufficient.*[5] *Interestingly though, the recovery threshold*[6] *of Short-Dot codes, i.e., $K = P - \frac{Ps}{N} + M$ can be shown to be near-optimal when compared with the fundamental limits for this problem (see Theorem 3, Theorem 4 and also follow up work in [23]).*

D. Short-MDS: A Special Case of Short-Dot

Here, we first propose an alternative coding strategy, that we call Short-MDS codes, to achieve a pre-specified row-sparsity of at most s. For this code construction, the number of nodes required to finish in the best-case, average-case and worst-case are different. Next, we demonstrate (in Theorem 2) that, while the construction of Short-MDS codes might appear different from Short-Dot codes, interestingly the recoverability properties of Short-MDS codes can also be obtained as a special case of the Short-Dot code construction with a chosen pre-specified sparsity pattern on $\boldsymbol{F}_{P\times N}$. In other words, there exists a Short-Dot code construction that requires the same number of nodes to finish in the best-case, average-case and worst-case as Short-MDS codes while imposing the same sparsity pattern on $\boldsymbol{F}_{P\times N}$ and allowing for a single dot product computation of length at most s at each node. Furthermore, Short-Dot codes being a more general scheme, also allow for other sparsity patterns on $\boldsymbol{F}_{P\times N}$ that can result in a lower recovery threshold than Short-MDS codes (or Short-Dot with the same sparsity pattern) when s does not divide N. Now we move on to the description of the Short-MDS coding strategy.

Strategy Description: Our proposed Short-MDS codes are an extension of the MDS coding strategy proposed in [2]. First we block-partition the matrix of N columns vertically into $\lceil \frac{N}{s} \rceil$ sub-matrices of size $M \times s$ (except possibly the last block which is of size $M \times s'$ where $s' < s$ when s does not exactly divide N). We also divide the total processing nodes P equally into $\lceil \frac{N}{s} \rceil$ parts. Now, each sub-matrix can be encoded using a $(\frac{P}{\lceil \frac{N}{s} \rceil}, M)$ MDS code, assuming P is large enough to be divisible by $\lceil \frac{N}{s} \rceil$. We refer the reader to Fig. 7 for a pictorial illustration of this strategy.

Essentially, the Short-MDS code construction divides the P nodes into $\lceil \frac{N}{s} \rceil$ subsets of size $\frac{P}{\lceil \frac{N}{s} \rceil}$ each, and requires any M out of $\frac{P}{\lceil \frac{N}{s} \rceil}$ nodes in each subset to finish to be able to reconstruct the entire result. Therefore, in the best case, Short-MDS codes would only require $M\lceil \frac{N}{s} \rceil$ nodes to finish. However, in the worst case including the integer effect, this strategy requires $K = P - \frac{P}{\lceil \frac{N}{s} \rceil} + M$ processing nodes to finish, which is its recovery threshold.[7]

While the encoding techniques of Short-Dot and Short-MDS codes apparently seem to be different, the proposed Short-MDS code construction can also be obtained as a special case of the Short-Dot code construction with $K = P - \frac{P}{\lceil \frac{N}{s} \rceil} + M$ when imposing the same sparsity pattern on $\boldsymbol{F}$ as shown in Fig. 7c or Fig. 7f. In other words, we will show in Theorem 2 that the particular Short-Dot code construction with final sparsity pattern same as Short-MDS also has similar coding-theoretic recoverability properties as Short-MDS codes, in that, it also divides the P nodes into $\lceil \frac{N}{s} \rceil$ subsets of size $\frac{P}{\lceil \frac{N}{s} \rceil}$ each and requires any M out of the $\frac{P}{\lceil \frac{N}{s} \rceil}$ nodes in every subset to finish.

More importantly, our proposed Short-Dot codes provide a generalized framework achieving a wider variety of sparsity patterns, that can achieve a recovery threshold of $K = P - \frac{P}{\lceil \frac{N}{s} \rceil} + M$ (same final sparsity pattern as Short-MDS) or a recovery threshold of $K = P - \frac{Ps}{N} + M$, which is lower when s does not divide N, while imposing other sparsity patterns on $\boldsymbol{F}$, *e.g.*, the pattern of Fig. 5. Next, we discuss Theorem 2.

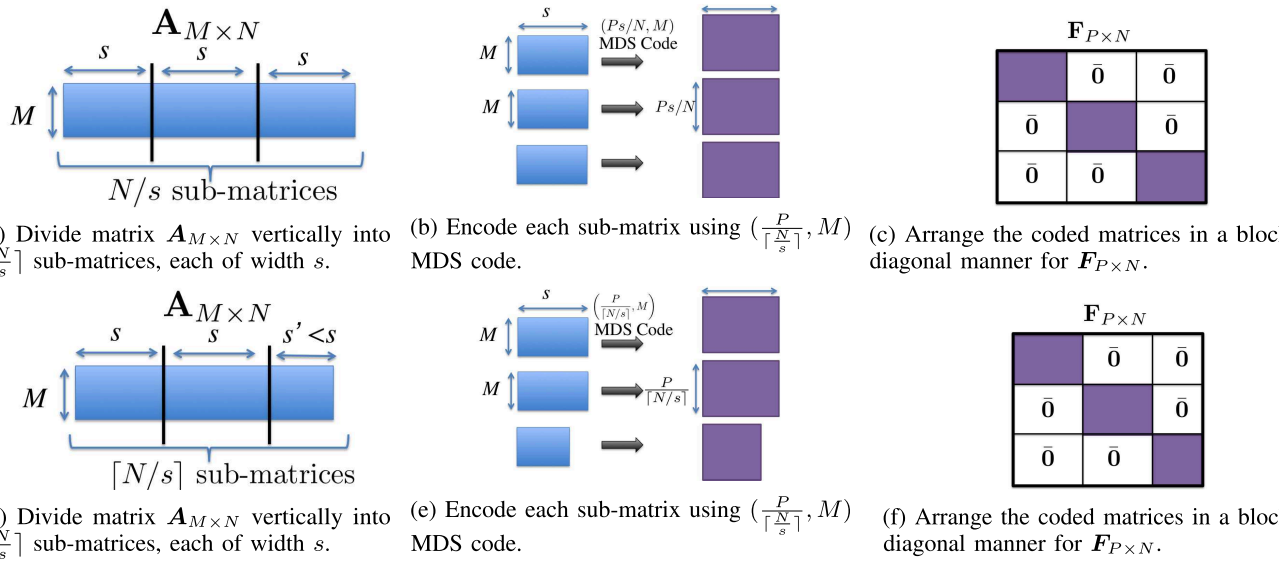
Theorem 2. *For the distributed coded matrix-vector product $\boldsymbol{A}_{M\times N}\boldsymbol{x}_{N\times 1}$, there exists a Short-Dot code construction with worst-case recovery threshold $K = P - \frac{P}{\lceil \frac{N}{s} \rceil} + M$, that divides all the P nodes into $\lceil \frac{N}{s} \rceil$ subsets of size $\frac{P}{\lceil \frac{N}{s} \rceil}$ each and can reconstruct all the dot products while requiring one to wait for only M nodes out of the $\frac{P}{\lceil \frac{N}{s} \rceil}$ nodes in every subset.*

Proof Sketch: The proof is provided in Appendix A. From the Short-Dot code construction, for any column of $\boldsymbol{F}$

[5] There are other coded computing strategies like Product Codes [22] where the number of nodes to wait for in the best-case, average-case and worst-case differ.

[6] This is derived assuming Ps/N is an integer. Otherwise, the expression becomes $K = P - \lfloor \frac{Ps}{N} \rfloor + M$ as also mentioned in Table I.

[7] More rigorously, the recovery threshold is $K = P - \left\lfloor \frac{P}{\lceil \frac{N}{s} \rceil} \right\rfloor + M$ when $\lceil \frac{N}{s} \rceil$ also does not divide P as mentioned in Table I.



(a) Divide matrix $\boldsymbol{A}_{M\times N}$ vertically into $\lceil \frac{N}{s} \rceil$ sub-matrices, each of width s.

(b) Encode each sub-matrix using $(\frac{P}{\lceil \frac{N}{s} \rceil}, M)$ MDS code.

(c) Arrange the coded matrices in a block-diagonal manner for $\boldsymbol{F}_{P\times N}$.

(d) Divide matrix $\boldsymbol{A}_{M\times N}$ vertically into $\lceil \frac{N}{s} \rceil$ sub-matrices, each of width s.

(e) Encode each sub-matrix using $(\frac{P}{\lceil \frac{N}{s} \rceil}, M)$ MDS code.

(f) Arrange the coded matrices in a block-diagonal manner for $\boldsymbol{F}_{P\times N}$.

Fig. 7. Illustration of our proposed Short-MDS coding strategy: (a), (b) and (c) correspond to the case where when s exactly divides N, while (d), (e) and (f) correspond to the case where s does not exactly divide N.

(say column j) observe that:

$$\boldsymbol{B}\begin{bmatrix}\boldsymbol{A}_j\\ \boldsymbol{Z}_j\end{bmatrix} = \begin{bmatrix}\boldsymbol{F}_j\end{bmatrix}. \tag{6}$$

Let $\mathcal{U}$ denote the indices of the $(K-M)$ indices that are pre-specified to be 0 in $\boldsymbol{F}_j$ and $\mathcal{U}^c$ denote the remaining $P-K+M=\frac{P}{\lceil \frac{N}{s} \rceil}$ indices. Observe that,

$$\begin{aligned}\boldsymbol{B}\begin{bmatrix}\boldsymbol{A}_j\\ \boldsymbol{Z}_j\end{bmatrix} &= \begin{bmatrix}\boldsymbol{B}^{\mathcal{U}^c}_{col\ 1:M} & \boldsymbol{B}^{\mathcal{U}^c}_{col\ M+1:K}\\ \boldsymbol{B}^{\mathcal{U}}_{col\ 1:M} & \boldsymbol{B}^{\mathcal{U}}_{col\ M+1:K}\end{bmatrix}\begin{bmatrix}\boldsymbol{A}_j\\ \boldsymbol{Z}_j\end{bmatrix}\\ &= \begin{bmatrix}\boldsymbol{F}^{\mathcal{U}^c}_j\\ \boldsymbol{F}^{\mathcal{U}}_j\end{bmatrix} = \begin{bmatrix}\boldsymbol{F}^{\mathcal{U}^c}_j\\ \boldsymbol{0}\end{bmatrix}.\end{aligned} \tag{7}$$

This leads to,

$$\boldsymbol{B}^{\mathcal{U}^c}_{col\ 1:M}\boldsymbol{A}_j + \boldsymbol{B}^{\mathcal{U}^c}_{col\ M+1:K}\boldsymbol{Z}_j = \boldsymbol{F}^{\mathcal{U}^c}_j, \text{ and} \tag{8}$$

$$\boldsymbol{B}^{\mathcal{U}}_{col\ 1:M}\boldsymbol{A}_j + \boldsymbol{B}^{\mathcal{U}}_{col\ M+1:K}\boldsymbol{Z}_j = \boldsymbol{0}. \tag{9}$$

The most important observation here is that the non-zero entries of $\boldsymbol{F}_j$, i.e., $\boldsymbol{F}^{\mathcal{U}^c}_j$ is not just an affine transform as one would think but in fact a linear transform of $\boldsymbol{A}_j$. More specifically,

$$\begin{aligned}(\boldsymbol{B}^{\mathcal{U}^c}_{col\ 1:M} - \boldsymbol{B}^{\mathcal{U}^c}_{col\ M+1:K}[\boldsymbol{B}^{\mathcal{U}}_{col\ M+1:K}]^{-1}\boldsymbol{B}^{\mathcal{U}}_{col\ 1:M})\boldsymbol{A}_j\\ = \boldsymbol{F}^{\mathcal{U}^c}_j\end{aligned}$$

Next, we show that this $\frac{P}{\lceil \frac{N}{s} \rceil} \times M$ matrix $\left(\boldsymbol{B}^{\mathcal{U}^c}_{col\ 1:M} - \boldsymbol{B}^{\mathcal{U}^c}_{col\ M+1:K}[\boldsymbol{B}^{\mathcal{U}}_{col\ M+1:K}]^{-1}\boldsymbol{B}^{\mathcal{U}}_{col\ 1:M}\right)$ also satisfies the MDS property that *any* M rows are linearly independent. Thus, the $P-K+M=\frac{P}{\lceil \frac{N}{s} \rceil}$ elements of $\boldsymbol{F}_j$ that are not pre-specified to be 0 are in fact coded elements of $\boldsymbol{A}_j$, encoded using a $\left(\frac{P}{\lceil \frac{N}{s} \rceil}, M\right)$ MDS code.

Remark 3 (Flexibility of Generator Matrices)**.** *Theorem 2 essentially demonstrates that Short-Dot codes when applied to achieve the sparsity pattern of Short-MDS codes is equivalent to encoding the M rows of each block of $\boldsymbol{A}$ (after initial block-partitioning) with a $(\frac{P}{\lceil \frac{N}{s} \rceil}, M)$ MDS code whose transpose of the generator matrix is given by $\left(\boldsymbol{B}^{\mathcal{U}^c}_{col\ 1:M} - \boldsymbol{B}^{\mathcal{U}^c}_{col\ M+1:K}[\boldsymbol{B}^{\mathcal{U}}_{col\ M+1:K}]^{-1}\boldsymbol{B}^{\mathcal{U}}_{col\ 1:M}\right)$ where $\mathcal{U}^c$ denotes the index of the nodes that will store this coded block. This is similar to the encoding of Short-MDS codes. Short-MDS codes might however allow for arbitrary $(\frac{P}{\lceil \frac{N}{s} \rceil}, M)$ MDS generator matrices for each block, while the generator matrices of Short-Dot codes are derived from the matrix $\boldsymbol{B}$. However, they have the same coding-theoretic recoverability properties in that they both divide the total nodes P into $\lceil \frac{N}{s} \rceil$ subsets such that one needs to wait for any M out of $\frac{P}{\lceil \frac{N}{s} \rceil}$ nodes in every subset.*

Remark 4 (Trade-offs between different sparsity patterns of Short-Dot)**.** *Theorem 2 also demonstrates that while the recovery threshold (worst-case) of Short-Dot is $K=P-\frac{Ps}{N}+M$ (or $K=P-\frac{P}{\lceil \frac{N}{s} \rceil}+M$ for Short-MDS type sparsity pattern under integer effects), under special scenarios fewer nodes might suffice as discussed in Remark 2 depending on the choice of the final sparsity pattern. This is suggestive of a trade-off between the different sparsity patterns in terms of worst-case recovery threshold (under integer effects), best-case and average-case recovery, expected computation time, deadline exponents [47], adversarial error-tolerance or other parameters depending on the application from where the sparsity constraint is arising from. In Section V-A, we discuss the differences in expected computation time due to the differences in the best-case and average-case recovery, although interestingly, the deadline exponent, i.e., the rate of decay of the probability of failure to finish within a deadline t (for large t) is determined by*

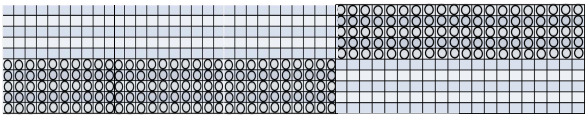
the worst-case recovery threshold (see Section V-B). Similarly, the adversarial error tolerance (standard in coding theory) is also determined by the worst-case recovery threshold. It will be an interesting future work to explore and compare the trade-offs between these different achievable sparsity patterns further.

Remark 5 (An interesting perspective on Short-Dot codes)**.** *Delving deeper into the proof of Theorem 2 provides an interesting perspective on the general Short-Dot code construction that is not specific to any sparsity pattern. Consider the j-th column of $\boldsymbol{A}$, i.e., $\boldsymbol{A}_j$ which consists of M elements. Short-Dot essentially encodes these M elements using a $(P-K+M, M)$ MDS code whose transpose of the generator matrix is given by $\left(\boldsymbol{B}^{\mathcal{U}^c}_{col\ 1:M} - \boldsymbol{B}^{\mathcal{U}^c}_{col\ M+1:K}[\boldsymbol{B}^{\mathcal{U}}_{col\ M+1:K}]^{-1}\boldsymbol{B}^{\mathcal{U}}_{col\ 1:M}\right)$ where $\mathcal{U}$ denotes the $(K-M)$ indices of the desired zero-indices of the column $\boldsymbol{F}_j$. These encoded $P-K+M$ elements are essentially the elements in the non-zero indices ($\mathcal{U}^c$) of the column $\boldsymbol{F}_j$.*

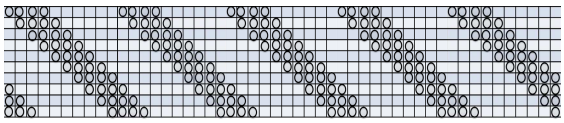
Discussion on Short-MDS and Short-Dot codes. We highlight some major points here:

- Short-Dot codes are a generalized framework that can achieve a wider variety of pre-specified sparsity patterns depending on the application as compared to Short-MDS. Essentially, any sparsity pattern on $\boldsymbol{F}_{P\times N}$ which has $K-M$ zeros in each column is achievable using Short-Dot codes (see Theorem 1 and Corollary 1), including the sparsity pattern imposed by Short-MDS codes on $\boldsymbol{F}_{P\times N}$. Interestingly, even though the two encoding techniques might appear to be somewhat different, Short-Dot codes imposing the same sparsity pattern on $\boldsymbol{F}_{P\times N}$ also ends up dividing the set of P nodes into $\frac{N}{s}$ subsets of size $\frac{P}{\frac{N}{s}}$ each such that any M out of $\frac{P}{\frac{N}{s}}$ nodes in every subset is sufficient as proved in Theorem 2. Therefore, Short-Dot codes also have the same best-case and average-case recoverability properties as Short-MDS codes, when the final sparsity pattern imposed on $\boldsymbol{F}$ is chosen as in Fig. 7c or Fig. 7f.
- In the regime where s does not exactly divide N, this Short-MDS code construction (special case of Short-Dot with a particular final sparsity pattern on $\boldsymbol{F}$) requires more processing nodes to finish in the worst case than Short-Dot codes with a different sparsity pattern on $\boldsymbol{F}$, e.g., the pattern of Fig. 5. As an example, suppose $s = \frac{3N}{5}$. Then, Short-MDS codes would require $\frac{P}{2}+M$ nodes to finish in the worst case while Short-Dot codes with a different sparsity pattern, similar to Fig. 5, would require only $\frac{2P}{5}+M$ nodes to finish in the worst case, when both the strategies are allowed to compute only one dot product per node of length at most s. We also illustrate this in Fig. 8.

In Table I, we compare the lengths of the dot products (s) and the recovery threshold (K), i.e., the number of processing nodes to wait for in the worst case, for different strategies, accounting for all the integer effects.



(a) Sparsity pattern of $\boldsymbol{F}_{P\times N}$ achievable using Short-MDS (a special case of Short-Dot with $K = P - \frac{P}{\lceil \frac{N}{s} \rceil} + M$).



(b) Different sparsity pattern of $\boldsymbol{F}_{P\times N}$ also achievable using Short-Dot with $K = P - \frac{Ps}{N} + M$.

Fig. 8. Sparsity pattern of $\boldsymbol{F}_{P\times N}$: Here P is the number of processing nodes, $N = 5P$ and allowed length of dot products at each node is $s = 3P$. The sparsity pattern on the left is achievable using Short-MDS and Short-Dot codes using $K = \frac{P}{2} + M$. The pattern on the right is also achievable using Short-Dot codes, but using a lower $K = \frac{2P}{5} + M$.

TABLE I

TRADE-OFF BETWEEN THE LENGTH OF THE DOT PRODUCTS AND RECOVERY THRESHOLD K FOR DIFFERENT STRATEGIES

Strategy	Length of the dot products at each node	Recovery Threshold K
Replication	N	$P - \left\lfloor \frac{P}{M} \right\rfloor + 1$
MDS	N	M
Short-Dot	s	$P - \left\lfloor \frac{Ps}{N} \right\rfloor + M$
Replication with block partition	s	$P - \left\lfloor \frac{P}{M\lceil N/s \rceil} \right\rfloor + 1$
Short-MDS	s	$P - \left\lfloor \frac{P}{\lceil N/s \rceil} \right\rfloor + M$

IV. LIMITS ON TRADE-OFF BETWEEN THE LENGTH OF DOT PRODUCTS (s) AND RECOVERY THRESHOLD (K)

In this section, we derive fundamental trade-offs between the length of the dot products computed at each individual processing node and the number of processing nodes to wait for, i.e., K, which parametrizes the recovery threshold under straggling. First we derive an information-theoretic limit in Theorem 3 that holds for any matrix $\boldsymbol{A}$, such that each column has at least one non-zero entry.[8] In Theorem 4, we show how this bound can be tightened further (improving our previous result in [1]), so that in the limit of large number of columns of matrix $\boldsymbol{A}$, Short-Dot is near-optimal.

A. Fundamental Limits on Sparsity

Theorem 3. *Let $\boldsymbol{A}_{M\times N}$ be any matrix such that each column has at least one non-zero element. For any matrix $\boldsymbol{F}_{P\times N}$ satisfying the property that the span of its any K rows contains*

[8] Note that choice of such a class of matrix $\boldsymbol{A}$ is reasonable, since if say the j-th column of $\boldsymbol{A}$ consists entirely of zeros, then the j-th column and its corresponding entry in unknown vector $\boldsymbol{x}$ can simply be omitted from the problem.

the span of the M rows of $\boldsymbol{A}_{M\times N}$, the average sparsity $\bar{s}$ over the rows of $\boldsymbol{F}_{P\times N}$ must satisfy $\bar{s} \geq \frac{N}{P}\big(P-K+1\big)$.

Proof of Theorem 3. We claim that K is strictly greater than the maximum number of zeros that can occur in any column of the matrix $\boldsymbol{F}$. If not, suppose the j-th column of $\boldsymbol{F}$ has more than K zeros. Then there exists a choice of K rows of $\boldsymbol{F}$ such that any linear combination of these rows will always be 0 at the j-th column index. However, since the j-th column of $\boldsymbol{A}$ has at least one non-zero entry, say at row i, it is not possible to generate the i-th row of $\boldsymbol{A}$ by linearly combining these chosen K rows of $\boldsymbol{F}$. Thus,

$$K \geq 1 + \text{Maximum No. of 0s in any column of } \boldsymbol{F} \quad (10)$$

$$\geq 1 + \text{Average No. of 0s per column of } \boldsymbol{F}. \quad (11)$$

Here the last line follows since maximum value is always greater than average. Note that if $\bar{s}$ is the average sparsity over the rows of $\boldsymbol{F}_{P\times N}$, then the average number of zeros over the columns of $\boldsymbol{F}_{P\times N}$ can be written as $\frac{(N-\bar{s})P}{N}$. Thus, from (11),

$$K \geq 1 + \frac{(N-\bar{s})P}{N}. \quad (12)$$

A slight re-arrangement establishes the lower bound in Theorem 3. □

Recall that, Short-Dot achieves a column sparsity of at most $(P-K+M)$ while a hard lower bound is $(P-K+1)$ from this proof. The bound is tight for $M=1$. The bound on average row-sparsity $\bar{s} \geq \frac{N}{P}\big(P-K+1\big)$ is also tight only for $M=1$ (implicitly assuming P divides N, since $P \ll N$). Now we tighten this bound further for $M>1$.

Remark 6 (Converse to the gradient coding problem)**.** *As discussed before, the concurrent problem of gradient coding [38]–[41] shares similarities with the Short-Dot problem formulation for the case of $M=1$, $N=P$, and the matrix $\boldsymbol{A}$ being equal to a single vector $[1,1,\ldots,1]_{1\times N}$. Thus, Theorem 3 also serves as an alternate and independently proposed converse to the gradient coding problem.*

B. Tighter Fundamental Bounds

Theorem 4. *Let $M>1$. Then there exists a matrix $\boldsymbol{A}_{M\times N}$, such that any $\boldsymbol{F}_{P\times N}$ satisfying the property that any K rows of $\boldsymbol{F}_{P\times N}$ can span all the rows of $\boldsymbol{A}_{M\times N}$, must also satisfy the following property:*

The average sparsity over the rows of $\boldsymbol{F}_{P\times N}$ is lower bounded as

$$\bar{s} > \frac{N}{P}\big(P-K+M\big) - \frac{M^2}{P}\binom{P}{K-M+1}. \quad (13)$$

Moreover, if N is sufficiently large, such that $M^2\binom{P}{K-M+1} = o(N)$, then the average sparsity over the rows of $\boldsymbol{F}_{P\times N}$ is lower bounded as

$$\bar{s} > \frac{N}{P}(P-K+M) - o\Big(\frac{N}{P}\Big). \quad (14)$$

Note that the second term in the lower bound in (13) does not depend on N. Thus, if N is sufficiently larger than P and M, the second term in the lower bound becomes negligible compared to the first term, and the first term is precisely what Short-Dot can achieve. Thus, from this lower bound, we can conclude that when N is large, Short-Dot is near optimal.

Before proceeding with the proof, we give a basic intuition on the proof technique. We essentially divide the columns of $\boldsymbol{F}_{P\times N}$ into two groups, one with at most $(K-M)$ zeros, and other with more than $(K-M)$ zeros. Then we show that there exist matrices $\boldsymbol{A}_{M\times N}$ such that the number of columns in the latter group, i.e., with more than $(K-M)$ zeros is bounded, and this in turn bounds the average sparsity. Now we formally prove the theorem.

Proof of Theorem 4. Let us denote the number of columns of $\boldsymbol{F}_{P\times N}$ with more than $(K-M)$ zeros as λ. We will show later in Lemma 2 that $\lambda < M\binom{P}{K-M+1}$. Now, compute the average number of zeros over the columns of $\boldsymbol{F}$. The columns of $\boldsymbol{F}$ can be divided into two groups : λ columns with greater than $(K-M)$ zeros and $(N-\lambda)$ columns with at-most $(K-M)$ zeros. Recall from (10), that if $\boldsymbol{A}$ is chosen such that every column has at-least one non-zero entry, then the maximum number of zeros in any column of $\boldsymbol{F}$ is upper bounded by $(K-1)$. Thus, the group of λ columns can have at most $K-1$ zeros each. Thus,

$$\begin{aligned}\text{Average No.of 0s per column} &\leq \frac{(K-1)\lambda + (K-M)(N-\lambda)}{N}\\ &= K - M + \frac{\lambda(M-1)}{N}\\ &\overset{Lemma\ 2}{<} K - M + \frac{M^2\binom{P}{K-M+1}}{N}. \quad (15)\end{aligned}$$

If $\bar{s}$ is the average sparsity of each row of $\boldsymbol{F}$, then the average zeros of each column of $\boldsymbol{F}$ is given by $\frac{(N-\bar{s})P}{N}$. Thus,

$$\frac{(N-\bar{s})P}{N} < (K-M) + \frac{M^2}{N}\binom{P}{K-M+1}. \quad (16)$$

After slight re-arrangement, the average sparsity of each row of $\boldsymbol{F}$ can be bounded as:

$$\bar{s} > \frac{N}{P}(P-K+M) - \frac{M^2}{P}\binom{P}{K-M+1}. \quad (17)$$

Thus, the first part of the theorem, i.e., (13) is proved. Using the condition that $M^2\binom{P}{K-M+1} = o(N)$ in (13), we can also obtain (14). Thus,

$$\bar{s} > \frac{N}{P}(P-K+M) - o\left(\frac{N}{P}\right). \quad (18)$$

Thus, the theorem is proved. □

Now it only remains to prove Lemma 2.

Lemma 2. *Let $M>1$. Then there exists a matrix $\boldsymbol{A}_{M\times N}$, such that any $\boldsymbol{F}_{P\times N}$ satisfying the property that any K rows of $\boldsymbol{F}_{P\times N}$ can span all the rows of $\boldsymbol{A}_{M\times N}$, must also satisfy the following property: The number of columns (λ) with more than $K-M$ zeros is upper bounded as $\lambda < M\binom{P}{K-M+1}$.*

Proof of Lemma 2. Assume, $\lambda \geq M\binom{P}{K-M+1}$. Now, a column with more than $(K-M)$ zeros will have at least $(K-M+1)$ zeros. There can be at most $\binom{P}{K-M+1}$ different patterns in which $(K-M+1)$ zeros can occur in a column of length P. Every column with more than $(K-M+1)$ zeros also has one of these $\binom{P}{K-M+1}$ column sparsity pattern, just with more zeros. From a pigeon-hole argument, at least one of these sparsity patterns of $(K-M+1)$ zeros will surely occur in $\frac{\lambda}{\binom{P}{K-M+1}}$ columns or more. Let us consider the sub-matrix of $\boldsymbol{F}$, of size $P \times \frac{\lambda}{\binom{P}{K-M+1}}$, consisting of only the columns of $\boldsymbol{F}$ having $(K-M+1)$ zeros in the same locations, i.e., with similar sparsity pattern. Any K rows of this sub-matrix of $\boldsymbol{F}$ should generate all the rows of a corresponding $M \times \frac{\lambda}{\binom{P}{K-M+1}}$ sub-matrix of the given $\boldsymbol{A}$, consisting of the same columns of $\boldsymbol{A}$ as picked in this sub-matrix of $\boldsymbol{F}$.

There always exists a fully dense matrix $\boldsymbol{A}$ such any $M \times \frac{\lambda}{\binom{P}{K-M+1}}$ sub-matrix of $\boldsymbol{A}$ is full-rank, since $\boldsymbol{A}$ can be arbitrary. This sub-matrix of $\boldsymbol{A}$ is of rank $\min\{M, \frac{\lambda}{\binom{P}{K-M+1}}\} = M$(from assumption). Any K rows of the sub-matrix of $\boldsymbol{F}$, should generate M linearly independent rows of this sub-matrix of $\boldsymbol{A}$. But since the sub-matrix of $\boldsymbol{F}$ has $(K-M+1)$ rows consisting of all zeros, there is a choice of K rows, such that all these zero rows are chosen, and we are only left with at most $M-1$ non-zero rows to generate M linearly independent rows of $\boldsymbol{A}$. This is a contradiction. Thus, we must have $\lambda > M\binom{P}{K-M+1}$. □

Remark 7 (Optimality of Short-Dot codes)**.** *Following the initial publication of this work [1], an improved converse has been obtained in [23] which prove that Short-Dot codes are optimal.*

V. Probabilistic Analysis of Computation Time for Exponential Tail Models

A. Expected Computation Time

In this section, we provide a probabilistic analysis of the computation time required by Short-Dot and compare it with uncoded parallelization, replication and the MDS coding strategy [2] as shown in Fig. 9. We follow the shifted-exponential computation time model as described in [2]. Although the shifted exponential distribution may only be a crude approximation of the delay of real systems, we use the shifted exponential model since it is analytically tractable and allows for a fair comparison with the strategy proposed in [2]. We assume that the time required by a processing node to compute a single dot product of length N has a cumulative distribution as follows:

$$\Pr(T^{(N)} \leq t) = \begin{cases} 1 - e^{(-\mu\ (\frac{t}{N}-1))}, & \forall\ t \geq N \\ 0, & \text{otherwise.} \end{cases} \tag{19}$$

Here, $\mu(>0)$ is the "straggling parameter" that determines the unpredictable latency in computation time. Intuitively, the shifted exponential model states that for a task of size N, there is a minimum time offset proportional to N such that the probability of completion of the task before that time is 0. The probability of task completion is maximum at the time offset and then decays with an exponential tail after that. This nature of the model might be attributed to the fact that while a processing node is most likely to finish its task of size N at a time proportional to N, but an unpredictable latency due to queuing and various other factors causes an exponential tail. For an s length dot product, we simply replace N by s in (19), as suggested in [2]. The analysis of expected computation time requires closed form expressions of the K-th statistic which is simplistic for exponential tails. However a more thorough empirical study is necessary to establish any chosen model for straggling in a particular environment.

Short-Dot (Worst Case): In the worst case, Short-Dot codes require the decoder to wait for any K out of P nodes to finish, each computing a dot product of length s. The expected computation time for Short-Dot is therefore the expected value of the K-th order statistic of these P *iid* shifted-exponential random variables, which is given by:

$$\begin{aligned} \mathbb{E}[T_{\text{SD}}] &= s\left(1 + \frac{1}{\mu}\sum_{i=P-K+1}^{P}\frac{1}{i}\right) \approx s\left(1 + \frac{\log(\frac{P}{P-K})}{\mu}\right) \\ &= \frac{N(P-K+M)}{P}\left(1 + \frac{\log(\frac{P}{P-K})}{\mu}\right). \end{aligned} \tag{20}$$

Here, (20) uses the fact that the expected value of the K-th statistic of P *iid* exponential random variables with parameter 1 is $\sum_{i=1}^{P}\frac{1}{i} - \sum_{i=1}^{P-K}\frac{1}{i} \approx \log(P) - \log(P-K)$ [2]. The expected computation time in the RHS of (20) is minimized when $P - K = \Theta(M)$. This minimal expected time is $\mathcal{O}(\frac{MN}{P})$ for M linear in P and is $\mathcal{O}\left(\frac{MN\log(P/M)}{P}\right)$ for M sub-linear in P.

A detailed analysis of the expected computation time for the competing strategies, i.e., uncoded strategy, replication and MDS coding strategy is provided in Appendix B. Table II shows the order-sense expected computation time in the regimes where M is linear and sub-linear in P.

Note that in the regime where M is linear in P, Short-Dot outperforms Uncoded Strategy by a factor diverging to infinity for large P. Similarly, in the regime where M is sub-linear in P, Short-Dot outperforms MDS coding strategy by a factor that diverges to infinity for large P. Thus Short-Dot universally outperforms all its competing strategies over the entire range of M (also see Fig. 9).

Short-Dot (Specific Sparsity Patterns): Recall that, while the worst-case recovery threshold of Short-Dot codes is K, the decoder requires fewer nodes to finish in the best case for certain specific sparsity patterns (e.g. the pattern obtained using Short-MDS codes). For this sparsity pattern, Short-Dot and Short-MDS codes both divide the set of P nodes into $\lceil(N/s)\rceil$ subsets, and wait for any M out of $\frac{P}{\lceil(N/s)\rceil}$ nodes in each subset to finish. The expected computation time is therefore the expected value of the maximum of $\lceil(N/s)\rceil$ *iid* random variables, each of which is distributed as the M-th order statistic out of $\frac{P}{\lceil(N/s)\rceil}$ *iid*[9] shifted-exponentials.

[9] Strictly speaking, when s does not exactly divide N, the statistic of the last group may not be identically distributed as the length of the dot product is $s' < s$. However, for a simplified analysis, we assume they are also identically distributed as if the nodes were also computing dot products of length s. This assumption is in fact a conservative assumption for us because $s' < s$.

TABLE II
PROBABILISTIC COMPUTATION TIMES

Strategy	Expected Time	M linear in P	M sub-linear in P
Only one processing node	$MN\left(1+\frac{1}{\mu}\right)$	$\Theta(MN)$	$\Theta(MN)$
Uncoded (M divides P)[1]	$\frac{MN}{P}\left(1+\frac{\log(P)}{\mu}\right)$	$\Theta\left(\frac{MN}{P}\log(P)\right)$	$\Theta\left(\frac{MN}{P}\log(P)\right)$
Replication (M divides P)[1]	$N\left(1+\frac{M\log(M)}{P\mu}\right)$	$\Theta\left(\frac{MN}{P}\log(P)\right)$	$\Theta(N)$
MDS	$N\left(1+\frac{\log\left(\frac{P}{P-M}\right)}{\mu}\right)$	$\Theta(N)$	$\Theta(N)$
Short-Dot[2]	$\frac{N(P-K+M)}{P}\left(1+\frac{\log\left(\frac{P}{P-K}\right)}{\mu}\right)$	$\mathcal{O}(\frac{MN}{P})$	$\mathcal{O}\left(\frac{MN}{P}\log\left(\frac{P}{M}\right)\right)$

[1] Refer to Appendix B for more accurate analysis taking integer effects into account.
[2] The expected computation time is lower for specific sparsity patterns of Short-Dot codes, e.g., the pattern of Short-MDS codes.

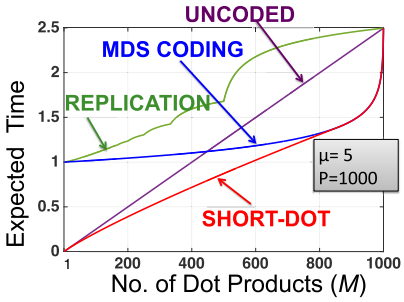


Fig. 9. Expected computation time: Short-Dot (using the worst-case recovery threshold K) is faster than MDS when $M \ll P$ and Uncoded when $M \approx P$, and is universally faster over the entire range of M. For the choice of straggling parameter, replication is slowest. When M does not exactly divide P, the distribution of computation time for replication and uncoded strategies is the maximum of non-identical but independent random variables, which produce the ripples in these curves (see Appendix B for details). The expected computation time of Short-Dot codes can be improved further using specific sparsity patterns, e.g., the pattern of Short-MDS codes.

The computation time at each node is given by: $s + T'$ where T' is an exponential random variable with mean s/μ. Let us denote the M-th order statistic out of $\frac{P}{\lceil (N/s) \rceil}$ such *iid* random variables as $T'_{(M)}$. The cumulative distribution of $T'_{(M)}$ is obtained as:

$$\Pr(T'_{(M)} \leq t) = \sum_{i=M}^{\frac{P}{\lceil (N/s) \rceil}} \binom{\frac{P}{\lceil (N/s) \rceil}}{i} \left(1 - e^{-\frac{\mu t}{s}}\right)^i \left(e^{-\frac{\mu t}{s}}\right)^{\frac{P}{\lceil (N/s) \rceil} - i}. \quad (21)$$

The expected computation time of Short-Dot and Short-MDS codes ($T_{\text{SD/SMDS}}$) is given by: $s + T'_{\text{SD/SMDS}}$ where $T'_{\text{SD/SMDS}}$ is the maximum of $\lceil (N/s) \rceil$ *iid* random variables distributed as $T'_{(M)}$. Thus,

$$\Pr(T'_{\text{SD/SMDS}} < t) = \left(\Pr(T'_{(M)} \leq t)\right)^{\lceil (N/s) \rceil}. \quad (22)$$

Finally, the expected computation time is obtained as:

$$\begin{aligned}\mathbf{E}[T_{\text{SD/SMDS}}] &= s + \mathbf{E}[T'_{\text{SD/SMDS}}] \\ &= s + \int_0^\infty \Pr(T'_{\text{SD/SMDS}} > t)dt \\ &= s + \int_0^\infty \left(1 - \left(\Pr(T'_{(M)} \leq t)\right)^{\lceil (N/s) \rceil}\right) dt.\end{aligned}$$

Asymptotic Analysis: In the asymptotic regime, when M, P, s and N are all very large, we can analyze the expected computation time of this strategy using techniques from [22]. Please see Appendix C for a background.

For simplicity, we let the pdf $f(t) = \mu e^{-\mu t}$ and the cdf $F(t) = 1 - e^{-\mu t}$ for an exponential distribution with parameter μ. Then, the distribution of $T'_{(M)}$ converges in probability to a Gaussian random variable distributed as: $\mathcal{N}(st_r, \frac{s^2 r(1-r)}{nf^2(t_r)})$ where r is the ratio of M and $\frac{P}{\lceil \frac{N}{s} \rceil}$ which is assumed to be fixed between 0 and 1, and $t_r = F^{-1}(r) = \frac{1}{\mu}\log\left(\frac{P}{P - M\lceil \frac{N}{s} \rceil}\right)$. Now, $T'_{\text{SD/SMDS}}$ is the maximum of $\lceil \frac{N}{s} \rceil$ *iid* random variables distributed as $T'_{(M)}$. In this asymptotic regime, the maximum of $\lceil \frac{N}{s} \rceil$ *iid* Gaussian random variables, each distributed as $\mathcal{N}(st_r, \frac{s^2 r(1-r)}{nf^2(t_r)})$, can be approximated as follows:

$$\begin{aligned}\mathbf{E}[T'_{\text{SD/SMDS}}] &\approx st_r + \sqrt{\frac{s^2 r(1-r)}{nf^2(t_r)}}\sqrt{2\log\lceil (N/s) \rceil} \\ &= s\left(1 + \frac{\log\frac{P}{P - M\lceil (N/s) \rceil}}{\mu} + \frac{\lceil (N/s) \rceil\sqrt{2M\log\lceil (N/s) \rceil}}{\mu\sqrt{(P)(P - M\lceil (N/s) \rceil)}}\right).\end{aligned}$$

Now we explicitly provide a regime, where the speed-ups from Short-Dot (even in the worst-case) diverges to infinity for large P, in comparison to all three competing strategies - MDS Coding, Replication or Uncoded strategies. Interestingly, as we discuss in Remark 8, in this regime a further reduction in computation time by a factor of $\log(\log P)$ is obtained when we use Short-Dot codes with specific sparsity patterns (e.g. the pattern obtained using Short-MDS codes) as compared to waiting for the worst-case K every time.

Theorem 5. *Suppose M scales as $\Theta\left(\frac{P}{\log P}\right)$. Then, Short-Dot with $K = P - \frac{M}{2}$ has an expected computation time (scaled by N) as $\frac{\mathrm{E}[T_{\text{SD}}]}{N} = O(\frac{\log(\log P)}{\log P})$ that decays to 0 as $P \to \infty$. In contrast, the expected computation time (scaled by N) for MDS coding, replication and uncoded strategies scale as $\Omega(1)$ and thus do not decay to 0 as $P \to \infty$.*

Proof of Theorem 5. For the proof of this theorem, we simply substitute the values of M and K in the expressions of expected computation time as follows. We assume

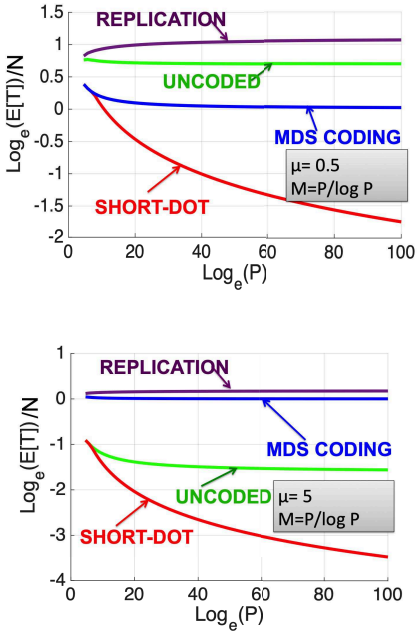


Fig. 10. Scaling sense comparison of expected time of Short-Dot with other strategies: Plot of the Log of Expected Computation Time (scaled down by N) with $\log(P)$ where P is the number of processing nodes, for the regime $M = P/\log P$. We observe that Short-Dot offers significant speed-ups compared to uncoded, replication and MDS coding that diverge for large values of P.

$c\frac{P}{\log P} \leq M \leq C\frac{P}{\log P}$ for constants c and C respectively. For uncoded strategy, we thus obtain,

$$\frac{\mathbb{E}[T_{\text{UC}}]}{N} = \frac{M}{P}\left(1+\frac{\log(P)}{\mu}\right) \geq \frac{c}{\log P}\left(1+\frac{\log(P)}{\mu}\right) \geq \frac{c}{\mu} = \Omega(1) \text{ where } c \text{ is a constant.} \quad (23)$$

For replication, we obtain,

$$\frac{\mathbb{E}[T_{\text{RP}}]}{N} = \left(1+\frac{M\log(M)}{P\mu}\right) \geq 1 = \Omega(1). \quad (24)$$

For the MDS Coding strategy, we obtain,

$$\frac{\mathbb{E}[T_{\text{MDS}}]}{N} = 1+\frac{\log\left(\frac{P}{P-M}\right)}{\mu} \geq 1 = \Omega(1). \quad (25)$$

Now, we consider the Short-Dot strategy with $K = P - \frac{M}{2}$. Note that the inequality $K > M$ only requires that $P \geq \frac{3}{2}M$ which is easily satisfied for $\log P > \frac{3}{2}C$. Now let us calculate the expected computation time for Short-Dot.

$$\frac{\mathbb{E}[T_{\text{SD}}]}{N} = \frac{(P-K+M)}{P}\left(1+\frac{\log\left(\frac{P}{P-K}\right)}{\mu}\right) \leq \frac{3C}{2\log P}\left(1+\frac{\log(2\log P)}{\mu}\right) \leq \mathcal{O}\left(\frac{\log(\log P)}{\log P}\right).$$

Thus, the speed-up offered by Short-Dot in this regime is $\frac{\log P}{\log(\log P)}$, and thus diverges to infinity for large P, as illustrated in Fig. 10. □

Remark 8 (Improvement using Short-Dot with specific sparsity patterns)**.** *Recall from Equation* (23) *that Short-Dot codes with specific sparsity pattern (and Short-MDS codes), in the asymptotic regime, have an expected computation time of*

$$s\left(1+\frac{\log\frac{P}{P-M\lceil(N/s)\rceil}}{\mu}+\frac{\lceil(N/s)\rceil\sqrt{2M\log\lceil(N/s)\rceil}}{\mu\sqrt{(P)(P-M\lceil(N/s)\rceil)}}\right).$$

Ignoring integer effects, for the case where M scales as $\Theta\left(\frac{P}{\log P}\right)$ and $K = P - \frac{M}{2}$, the expected computation time scales as follows:

$$\frac{\mathbb{E}[T_{\text{SD/SMDS}}]}{N} = \mathcal{O}\left(\frac{1}{\log P}\right).$$

Thus, it offers a speed up of $\log P$ over competing strategies and $\log(\log P)$ over Short-Dot codes with a worst case K.

B. Deadline Exponents

Here we include a brief discussion on deadline exponents (that were originally introduced in [47] for coded convolutions) for Short-Dot codes under different sparsity patterns. The deadline exponent is defined as:

$$\lim_{t\to\infty}\frac{-P_f(t)}{t} \quad (26)$$

where $P_f(t)$ is the probability of failure at time t, *i.e.*, the probability that an insufficient number of nodes have finished their computation within the deadline t in the limit of large t. Interestingly, this deadline exponent is determined by the worst-case recovery threshold, even though the best case and average case recovery threshold may be lower. What this implies is that a sparsity pattern with lower worst-case recovery threshold has a better deadline exponent, *i.e.*, the rate of decay of the probability of failure to finish within a deadline t when the deadline is large, even though it may differ in the number of nodes required to finish in the best-case and average-case and hence, also in the expected computation time.

We now derive the deadline exponent for Short-Dot codes (with the sparsity pattern of Short-MDS codes). As a first step, we compute the probability that the M-th statistic among $\frac{P}{\lceil(N/s)\rceil}$ does not finish by time t.

$$\Pr\left(T'_{(M)} > t-s\right) = \sum_{i=0}^{M-1}\binom{\frac{P}{\lceil(N/s)\rceil}}{i}\left(1-e^{-\frac{\mu(t-s)}{s}}\right)^i\left(e^{-\frac{\mu(t-s)}{s}}\right)^{\frac{P}{\lceil(N/s)\rceil}-i} \approx \Theta\left(e^{-\frac{\mu(t-s)P}{s\lceil(N/s)\rceil}}\left(e^{\frac{\mu(t-s)}{s}}-1\right)^{M-1}\right) \text{ when } t \text{ is large.} \quad (27)$$

Now, $P_f(t)$ is obtained by computing the probability that at least one out of the $\lceil(N/s)\rceil$ subsets did not have M nodes finish. Thus,

$$P_f(t) = \sum_{i=1}^{\lceil(N/s)\rceil}\binom{\lceil(N/s)\rceil}{i}\left(\Pr\left(T'_{(M)} > t-s\right)\right)^i \left(\Pr\left(T'_{(M)} \leq t-s\right)\right)^{\lceil(N/s)\rceil-i}$$

$$= 1 - \left(1 - \Pr\left(T'_{(M)} > t - s\right)\right)^{\lceil (N/s) \rceil}$$
$$\approx \Theta\left(\lceil (N/s) \rceil e^{-\frac{\mu(t-s)P}{s\lceil (N/s) \rceil}} \left(e^{\frac{\mu(t-s)}{s}} - 1\right)^{M-1}\right) \quad \text{when } t \text{ is large.} \quad (28)$$

Finally,

$$\lim_{t\to\infty} \frac{-P_f(t)}{t} = \frac{\mu}{s}\left(\frac{P}{\lceil (N/s) \rceil} - M + 1\right) = \frac{\mu}{s}\left(P - K_{\text{SD/SMDS}} + 1\right), \quad (29)$$

where $K_{\text{SD/SMDS}} = P - \frac{P}{\lceil (N/s) \rceil} + M$.

In comparison, we also derive an upper bound on the deadline exponent for Short-Dot codes with other sparsity patterns with the worst-case recovery threshold $K = P - \frac{Ps}{N} + M$ as follows:

$$P_f(t) \leq \Pr\left(K\text{-th statistic among } P \text{ does not finish by time } t\right)$$
$$= \sum_{i=0}^{K-1} \binom{P}{i} \left(1 - e^{\frac{-\mu(t-s)}{s}}\right)^i \left(e^{-\frac{\mu(t-s)}{s}}\right)^{P-i}$$
$$\approx \Theta\left(e^{-\frac{\mu(t-s)P}{s}} \left(e^{\frac{\mu(t-s)}{s}} - 1\right)^{K-1}\right) \text{when } t \text{ is large.} \quad (30)$$

Therefore,

$$\lim_{t\to\infty} \frac{-P_f(t)}{t} \leq \frac{\mu}{s}\left(P - K + 1\right), \quad (31)$$

where $K = P - \frac{Ps}{N} + M$. Thus, the deadline exponent is determined by the worst-case recovery threshold even though the expected computation time depends on the number of nodes required in the best case and average case. The sparsity pattern with the lowest worst-case recovery threshold also has the best deadline exponent, *i.e.*, steeper rate of decay of the failure probability when the deadline is large.

VI. Encoding and Decoding Complexity

In this section, we analyze the encoding and decoding overhead of our proposed Short-Dot codes.

A. Encoding Complexity

Even though encoding is a pre-processing step (since $\boldsymbol{A}$ is assumed to be given in advance), we include a complexity analysis for the sake of completeness. Recall from Section III that we first choose an appropriate matrix $\boldsymbol{B}$ of dimension $P \times K$, such that every $K \times K$ square sub-matrix is invertible and all $(K-M) \times (K-M)$ sub-matrices in the last $(K-M)$ columns are invertible. Now, for each of the N columns of the given matrix $\boldsymbol{A}$, we perform the following.

Set $\mathcal{U} \leftarrow (\{(j-1), \ldots, (j+K-M-1)\} \mod P) + 1$
Set $\boldsymbol{B}^{\mathcal{U}} \leftarrow$ Rows of $\boldsymbol{B}$ indexed by $\mathcal{U}$
Solve for $\boldsymbol{Z}_j$: $(\boldsymbol{B}^{\mathcal{U}}_{col\ M+1:K})[\ \boldsymbol{Z}_j\ ] = -\ \boldsymbol{B}^{\mathcal{U}}_{col\ 1:M}[\boldsymbol{A}_j]$
Set $\boldsymbol{F}_j = \boldsymbol{B}\begin{bmatrix}\boldsymbol{A}_j \\ \boldsymbol{Z}_j\end{bmatrix}$

For each of the N columns, the encoding requires a matrix inversion of size $(K-M) \times (K-M)$ to solve a linear system of aligns, a matrix-vector product of size $(K-M) \times M$ and another matrix vector product of size $P \times K$.

The naive encoding complexity is therefore $\mathcal{O}(N((K-M)^3 + (K-M)M + PK))$. Note that effectively there are only $\frac{N}{P}$ different column sparsity patterns for this particular design discussed in this paper. Thus, there are effectively $\frac{N}{P}$ unique $\boldsymbol{B}^{\mathcal{U}}$s, and thus $\frac{N}{P}$ unique matrix inversions can suffice for all the N columns, as sparsity pattern is repeated. Thus, the complexity can be reduced to $\mathcal{O}(\frac{N}{P}(K-M)^3 + (K-M)MN + PKN) = \mathcal{O}(\frac{N}{P}(K-M)^3 + 2PKN)$.

This is higher than the MDS coding strategy [2] that has an encoding complexity of $\mathcal{O}(MNP)$), but it is only a one-time cost that provides savings in online steps (as discussed earlier in this section).

Reduced Complexity using Vandermonde matrices: The encoding complexity can be reduced further for special choices of the matrix $\boldsymbol{B}$. Let us choose $\boldsymbol{B}$ to be a Vandermonde matrix as given by:

$$\boldsymbol{B} = \begin{bmatrix} h_1^{K-1} & \ldots & h_1 & 1 \\ h_2^{K-1} & \ldots & h_2 & 1 \\ \vdots & \ddots & \vdots & \vdots \\ h_P^{K-1} & \ldots & h_P & 1 \end{bmatrix}. \quad (32)$$

Here, $h_1, h_2, \ldots, h_K \in \mathbb{R}$, and are all distinct. This matrix $\boldsymbol{B}$ satisfies all the requirements of the encoding matrix. All $K \times K$ sub-matrices of $\boldsymbol{B}$ are invertible, and all $(K-M) \times (K-M)$ sub-matrices in the last $(K-M)$ columns are also invertible. Thus, this matrix can be used to encode the matrix $\boldsymbol{F}$. For each of the N columns of $\boldsymbol{F}$, the encoding requires solving a linear system of aligns for $\boldsymbol{Z}_j$, as given by:

$$(\boldsymbol{B}^{\mathcal{U}}_{col\ M+1:K})\boldsymbol{Z}_j = -\ \boldsymbol{B}^{\mathcal{U}}_{col\ 1:M}[\boldsymbol{A}_j]. \quad (33)$$

Here $\mathcal{U}$ denotes a set of $(K-M)$ indices $\in \{1, 2, \ldots, P\}$. The matrix-vector product $\boldsymbol{B}^{\mathcal{U}}_{col\ 1:M}[\boldsymbol{A}_j]$ is equivalent to the evaluation of a polynomial of degree $(K-1)$ with the K coefficients as $[\boldsymbol{A}_j^T \boldsymbol{0}_{(K-M)\times 1}]$ at $(K-M)$ arbitrary points given by $\{h_l | l \in \mathcal{U}\}$. Once this product is obtained, the linear system of aligns reduces to the interpolation of the $(K-M)$ unknown co-efficients of a polynomial of degree $(K-M-1)$ (which is $\boldsymbol{Z}_j$), from its value at $(K-M)$ arbitrary points as given by $\{h_l | l \in \mathcal{U}\}$. Once $\boldsymbol{Z}_j$ is obtained, we perform the following operation:

$$\boldsymbol{F}_j = \boldsymbol{B}\begin{bmatrix}\boldsymbol{A}_j \\ \boldsymbol{Z}_j\end{bmatrix}. \quad (34)$$

This step is equivalent to the evaluation of a polynomial of degree $(K-1)$ at P points given by $\{h_l | l = 1, 2, \ldots P\}$. Thus we decompose our encoding problem for each column of $\boldsymbol{A}$ into a bunch of polynomial evaluation and interpolation problems, all of degree less than P. Now, from [64], [65], we know that both the interpolation and the evaluation of a polynomial of degree less than P, at P arbitrary points is $\mathcal{O}(P \log^2(P))$. Thus, the complexity of encoding is $\mathcal{O}(NP \log^2(P))$.

B. Decoding Complexity

During decoding, we get K dot products from the first K processing nodes out of P. We then perform the following operations.

Set $\mathcal{X} \leftarrow$ Indices of the nodes that finished first
Set $\boldsymbol{B}^{\mathcal{X}} \leftarrow$ Rows of $\boldsymbol{B}$ indexed by $\mathcal{X}$
Set $\boldsymbol{v}_{K\times 1} \leftarrow \boldsymbol{F}^{\mathcal{X}}\boldsymbol{x}$
Set $\boldsymbol{Ax} = [\langle \boldsymbol{a}_1, \boldsymbol{x}\rangle, \ldots, \langle \boldsymbol{a}_M, \boldsymbol{x}\rangle]^T \leftarrow [(\boldsymbol{B}^{\mathcal{X}})^{-1}]^{1:M}\boldsymbol{v}$
Output: $\langle \boldsymbol{a}_1, \boldsymbol{x}\rangle, \ldots, \langle \boldsymbol{a}_M, \boldsymbol{x}\rangle$

We solve a system of K linear aligns in K variables and use only M values of the obtained solution vector. Thus, effectively we do a single matrix inversion of size $K \times K$ followed by a matrix-vector product of size $K \times M$. The decoding complexity of Short-Dot is thus $\mathcal{O}(K^3 + KM)$ which does not depend on N when $M, K \ll N$. This is nearly the same as $\mathcal{O}(M^3 + M^2)$ which is the complexity of the MDS coding strategy [2].

Reduced Complexity using Vandermonde matrices: Similar to encoding, using Vandermonde matrices can reduce the decoding complexity further. As already discussed, we choose the encoding matrix $\boldsymbol{B}$ as a Vandermonde matrix as described in (32). The decoding problem consists of solving a system of K linear aligns in K variables.

$$[\boldsymbol{B}^{\mathcal{X}}]\boldsymbol{w} = \boldsymbol{v} \tag{35}$$

Here $\mathcal{X}$ is a set of K indices $\subset \{1, 2, \ldots, P\}$. The problem of finding $\boldsymbol{w}$ is equivalent to the interpolation of the coefficients of a polynomial of degree $(K-1)$, from its values at K arbitrary points given by $\{h_l |\ l \in \mathcal{X}\}$. Again, from [64], [65], the interpolation of a polynomial of degree $(K-1)$, at K arbitrary points can be done in $\mathcal{O}(K\log^2(K))$, which thus becomes the decoding complexity.

VII. Short-Dot With Errors Instead of Erasures

While we focus on the problem of erasures in this paper, Short-Dot can also be used to correct errors. Consider the scenario when instead of straggling or failures, some processing nodes return entirely faulty or garbage outputs, in a distributed system and we do not know which of the outputs are erroneous. Observe that the encoding matrix $\boldsymbol{B}$ of Short-Dot is actually the transposed generator matrix of a (P, K) MDS code as it requires all $K \times K$ square submatrices to be invertible.[10] From coding theoretic arguments, Short-Dot codes designed to tolerate $(P-K)$ erasures, can also correct $\lfloor \frac{(P-K)}{2} \rfloor$ errors. First observe that if the code can tolerate $(P-K)$ erasures, then the Hamming Distance between any two code-words should at least be $(P-K+1)$. Hence, the number of errors that can be corrected is $\lfloor \frac{(\text{Hamming Distance -1})}{2} \rfloor$ which is $\lfloor \frac{(P-K)}{2} \rfloor$. However observe that Short-Dot is a real-number error correcting code, and thus requires a decoding algorithm that satisfies the following:

- The decoding works for real numbers and is not limited to finite fields alone.
- The decoding algorithm works for various choices of the encoding matrix $\boldsymbol{B}$ such as Gaussian random matrices[11] and is not limited to Vandermonde matrices, i.e. polynomial based MDS constructions such as Reed Solomon codes [18].

We observe that the real number decoding problem can be recast as a sparse reconstruction problem borrowing ideas from standard compressive sensing literature [66] which also yields concrete, decoding algorithms. We show that the problem reduces to an l_0 minimization problem, which can be relaxed into an l_1 minimization, or solved using alternate sparse reconstruction techniques, under certain constraints on the encoding matrix $\boldsymbol{B}$ as derived in [66]. Clearly these decoding algorithms which are effectively solving a sparse reconstruction problem, work for real numbers and are not limited to finite fields alone. They also works for different choices of the encoding matrix $\boldsymbol{B}$ such as Gaussian random matrices.

For the sake of completeness, we include a derivation of how the decoding problem reduces to a sparse reconstruction problem here. Algorithms to solve such sparse reconstruction problems have been widely studied in compressive sensing literature [66].

Recall that, we proposed a strategy to compute $\boldsymbol{Ax}$ for a given matrix $\boldsymbol{A}_{N\times M} = [\boldsymbol{a}_1, \boldsymbol{a}_2, \ldots, \boldsymbol{a}_M]^T$, by first constructing a matrix $\boldsymbol{F}_{P\times M}$ with $(P > M)$ such that each N-length row of $\boldsymbol{F}$, i.e., $\boldsymbol{f}_i^T$ has at most $\frac{N(P-K+M)}{P}$ non-zero elements and then computing $\boldsymbol{f}_i^T\boldsymbol{x}$ in P parallel processing nodes. In this section, we show that using this strategy, we can successfully compute $\boldsymbol{Ax}$, if the number of errors is at most $\lfloor \frac{(P-K)}{2} \rfloor$ out of P parallel dot products, as stated in the following theorem.

Theorem 6. *Suppose that a matrix* $\boldsymbol{A}_{M\times N} = [\boldsymbol{a}_1, \boldsymbol{a}_2, \ldots, \boldsymbol{a}_M]^T$ *is encoded using Short-Dot code into a matrix* $\boldsymbol{F}_{P\times N}$ *with* $(P > M)$ *such that each* N*-length row of* $\boldsymbol{F}$*, i.e.,* $\boldsymbol{f}_i^T$ *has at most* $\frac{N(P-K+M)}{P}$ *non-zero elements. Now, for any vector* $\boldsymbol{x}$*, the matrix-vector product* $\boldsymbol{Ax}$ *can be recovered from* $\boldsymbol{Fx}$*, provided there are at most* $\lfloor \frac{(P-K)}{2} \rfloor$ *errors in* $\boldsymbol{Fx}$*.*

The proof is provided in Appendix D.

VIII. Discussion

A. More Dot Products Than Processing Nodes

For the regime where $M \geq P$, the same encoding technique can be applied by first dividing the matrix $\boldsymbol{A}$ into $\frac{M}{m}$ smaller sub-matrices of size $m \times N$ such that $m \leq P$, and applying Short-Dot on each of these sub-matrices. Each processing node now computes $\frac{M}{m}$ dot products each of length $s = \frac{N(P-K+m)}{P}$ instead of a single dot product, and the fusion node only waits for the first K out of P nodes to finish. If the rows with same sparsity pattern are grouped together and stored in the same processing node, then the communication cost is significantly reduced during the online computations, since the

[10] However all generator matrices of MDS code might not be an encoding matrix for Short-Dot as it requires an additional condition that all $(P-K)\times(P-K)$ square sub-matrices in the last $(P-K)$ columns of $\boldsymbol{B}$ are invertible.

[11] For certain applications, Gaussian random matrices might sometimes be more preferable over polynomial based code constructions over the field of real numbers. One reason is that the condition number of Vandermonde matrices are particularly high, causing more numerical errors.

same s elements of the unknown vector $\boldsymbol{x}$ are accessed by a particular node.

However, in this regime, the number of dot products per-node, i.e., $\frac{M}{m}$ can also be varied across different strategies, varying the per-node computation complexity. Thus $K = \kappa(s, m)$ depends on both s and m. In fact, $K = \kappa(s, m) = \frac{Ps}{N} - P + m$. The MDS coding strategy is a special case of Short-Dot with $s = N$ and $K = \kappa(N, m) = m$. If the full vector $\boldsymbol{x}$ can be communicated during computation, i.e., s can be as high as N, then the MDS coding strategy (Short-Dot with $s = N$) may be used because its computation complexity per-node can be lowered by varying m for the same straggler tolerance as compared to Short-Dot with $s < N$. To see this, let us take an MDS coding strategy with $K = m$ and choose another Short-Dot strategy with $s < N$ such that $\kappa(s, m') = m$. The computational complexity of MDS coding (Short-Dot with $s = N$) is actually smaller than that of Short-Dot with $s < N$ (since $\frac{MN}{m} < \frac{M}{m'}s = \frac{MN(P-m+m')}{Pm'}$). However, using $s < N$ is beneficial in communication constrained scenarios where communicating the entire $\boldsymbol{x}$ is much more expensive than computation. For a communication limited regime, one might therefore prefer to use Short-Dot with $s < N$ and $K = \kappa(s, m') = \frac{Ps}{N} - P + m'$ over the MDS coding strategy which sets $s = N$.

B. Communication and Computation Benefits of Shorter Dot Products

The major advantage of using Short-Dot codes over the MDS coding strategy in [2] is that the length of the communicated input (portions of $\boldsymbol{x}$) as well as the pre-stored vectors (rows of $\boldsymbol{F}$) is shorter than N. It is thus applicable in systems where the principle bottlenecks in computation time is in communicating the input $\boldsymbol{x}$ to all the processing nodes, and it may not be feasible to broad-cast (multi-cast) $\boldsymbol{x}$ to all processing nodes at the same time. Thus, it is also useful in applications where communication costs are predominant over computation costs. For the case where $M < P$, i.e. the number of dot products is less than the number of available nodes, Short-Dot also has additional advantages over [2] in computation time under stragglers under an exponential time model.

C. Computation Time Model

Following [2], we assume that the computation time at each processing node has an independent and identical exponential distribution with parameter μ where μ scales with the task size. Exponential distributions and independence across nodes admit simplified expected time analysis while being a crude approximation of experimental observations. In several cases, even though the distribution is not exactly exponential, but the tail might be exponential. Analysis of the expected computation time under non-exponential computation time distributions (say Pareto or Weibull to start with) would be an interesting direction of future work.

D. What if the Input $\boldsymbol{x}$ Can be Encoded in Addition to Encoding of the Matrix $\boldsymbol{A}$?

Our recent work [30] (that builds on [27]–[29]) as well as concurrent work of Yu *et al.* [31] allows encoding of $\boldsymbol{x}$ as well. This improves the recovery threshold, but requires encoding of $\boldsymbol{x}$ on the fly, which in turn requires an encoding overhead of $\Theta(PN)$ at a centralized master node or requires each of the worker nodes to access to all the entries of $\boldsymbol{x}$ which might be too expensive for certain applications, such as, in fully distributed and/or time-critical scenarios.

APPENDIX A
PROOF OF THEOREM 2

Proof of Theorem 2. Recall from the Short-Dot code construction that,

$$\boldsymbol{B}_{P\times K}\widetilde{\boldsymbol{A}} = \boldsymbol{B}_{P\times K}\begin{bmatrix}\boldsymbol{A}_{M\times N}\\ \boldsymbol{Z}_{(K-M)\times N}\end{bmatrix} = \boldsymbol{F}_{P\times N}, \tag{36}$$

where $\boldsymbol{F}$ is our final encoded matrix of desired sparsity. Here we choose $\boldsymbol{B}$ to be a matrix whose all $K \times K$ sub-matrices are invertible. We impose the block-diagonal sparsity pattern, as shown in Fig. 11, on matrix $\boldsymbol{F}$ and then use a Short-Dot code construction with $K = P - \frac{P}{\lceil N/s\rceil} + M$ as discussed before. Note that, this sparsity pattern is achievable by Short-Dot as each column of $\boldsymbol{F}$ has $P - \frac{P}{\lceil N/s\rceil} = K - M$ zeros. Each node stores a different row of $\boldsymbol{F}$ and computes its dot product with the relevant portion of $\boldsymbol{x}$.

Let us now divide all the nodes into $\lceil N/s\rceil$ equal subsets, such that, the first subset consists of the nodes that store the first $\frac{P}{\lceil N/s\rceil}$ rows of $\boldsymbol{F}$, the second subset stores the next $\frac{P}{\lceil N/s\rceil}$ rows and so on. It is sufficient to show that *any M nodes from every subset suffices to reconstruct all the required dot products.*

We rewrite Equation (36) using the following matrix partitioning notations as follows.

$$\begin{aligned}\boldsymbol{B}\begin{bmatrix}\boldsymbol{A}\\ \boldsymbol{Z}\end{bmatrix} &= \boldsymbol{B}\begin{bmatrix}\boldsymbol{A}_{col\ \mathcal{W}_1}|\boldsymbol{A}_{col\ \mathcal{W}_2}|\ldots|\boldsymbol{A}_{col\ \mathcal{W}_{\lceil\frac{N}{s}\rceil}}\\ \boldsymbol{Z}_{col\ \mathcal{W}_1}|\boldsymbol{Z}_{col\ \mathcal{W}_2}|\ldots|\boldsymbol{Z}_{col\ \mathcal{W}_{\lceil\frac{N}{s}\rceil}}\end{bmatrix}\\ &= \begin{bmatrix}\boldsymbol{F}_{col\ \mathcal{W}_1}|\boldsymbol{F}_{col\ \mathcal{W}_2}|\ldots|\boldsymbol{F}_{col\ \mathcal{W}_{\lceil\frac{N}{s}\rceil}}\end{bmatrix}.\end{aligned} \tag{37}$$

Here $\boldsymbol{A}$, $\boldsymbol{Z}$ and $\boldsymbol{F}$ are all partitioned vertically into $\lceil\frac{N}{s}\rceil$ blocks, each of width s, except possibly the last block which may be of width $s' < s$ when s does not exactly divide N. The index sets are as follows:

$$\begin{aligned}&\mathcal{W}_1 = 1:s,\\ &\mathcal{W}_2 = s+1:2s,\\ &\vdots\\ &\mathcal{W}_{\lceil\frac{N}{s}\rceil} = (\lceil\frac{N}{s}\rceil - 1)s + 1 : N,\end{aligned}$$

or more generally $\mathcal{W}_i = (i-1)s + 1 : \min\{is, N\}$. Thus, $\boldsymbol{A}_{col\ \mathcal{W}_i}$, $\boldsymbol{Z}_{col\ \mathcal{W}_i}$ and $\boldsymbol{F}_{col\ \mathcal{W}_i}$ are sub-matrices of dimensions $M \times s$, $(K-M)\times s$ and $P \times s$ respectively (s may be replaced by s' for $i = \lceil\frac{N}{s}\rceil$), consisting of the columns of the respective matrices indexed by $\mathcal{W}_i$. Also observe that,

$$\boldsymbol{A}\boldsymbol{x} = \sum_{i=1}^{\lceil\frac{N}{s}\rceil}\boldsymbol{A}_{col\ \mathcal{W}_i}\boldsymbol{x}^{\mathcal{W}_i}.$$

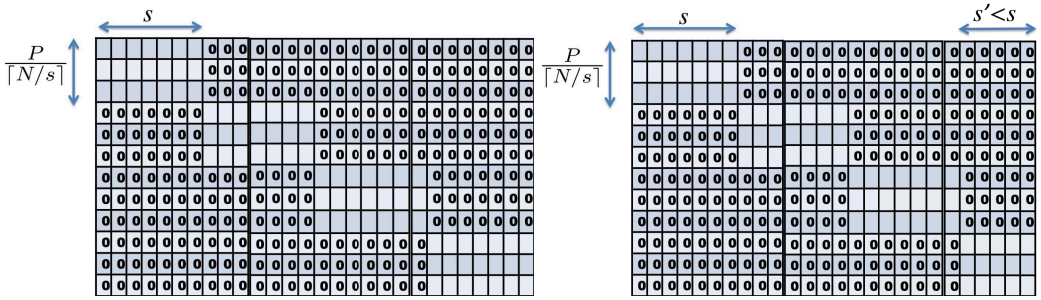


Fig. 11. Sparsity pattern of matrix $\boldsymbol{F}_{P\times N}$: (Left) When s divides N (Right) When s does not divide N. Note that, both these patterns can be achieved by Short-Dot code with $K = P - \frac{P}{\lceil N/s\rceil} + M$ as every column has $P - \frac{P}{\lceil N/s\rceil} = K - M$ zeros.

Now,

$$\boldsymbol{B}\begin{bmatrix}\boldsymbol{A}_{col\ \mathcal{W}_i}\\ \boldsymbol{Z}_{col\ \mathcal{W}_i}\end{bmatrix} = \begin{bmatrix}\boldsymbol{F}_{col\ \mathcal{W}_i}\end{bmatrix}$$
$$\implies \boldsymbol{B}\begin{bmatrix}\boldsymbol{A}_{col\ \mathcal{W}_i}\\ \boldsymbol{Z}_{col\ \mathcal{W}_i}\end{bmatrix}\boldsymbol{x}^{\mathcal{W}_i} = \begin{bmatrix}\boldsymbol{F}_{col\ \mathcal{W}_i}\end{bmatrix}\boldsymbol{x}^{\mathcal{W}_i}. \tag{38}$$

For the chosen sparsity pattern, only $\frac{P}{\lceil \frac{N}{s}\rceil}(= P - K + M)$ rows of $\boldsymbol{F}_{\mathcal{W}_i}$ are non-zero and all the rest are all 0. And interestingly, these $\frac{P}{\lceil \frac{N}{s}\rceil}$ non-zero rows of $\boldsymbol{F}_{col\ \mathcal{W}_i}$ turn out to be the rows that are essentially stored in the i-th subset of nodes, one at each node (see Fig. 11). If we are able to show that any M rows out of these $\frac{P}{\lceil \frac{N}{s}\rceil}$ non-zero rows of $\boldsymbol{F}_{col\ \mathcal{W}_i}$ can linearly span all the rows of $\boldsymbol{A}_{col\ \mathcal{W}_i}$, then we can conclude that any M dot products from the i-th subset of nodes can suffice to reconstruct the product $\boldsymbol{A}_{col\ \mathcal{W}_i}\boldsymbol{x}^{\mathcal{W}_i}$, and the theorem would be proved.

Without loss of generality, we show this for $i = 1$. The top $\frac{P}{\lceil \frac{N}{s}\rceil}(= P - K + M)$ rows of $\boldsymbol{F}_{col\ \mathcal{W}_1}$ are non-zero. Let $\mathcal{U} = \{\frac{P}{\lceil \frac{N}{s}\rceil}+1, \frac{P}{\lceil \frac{N}{s}\rceil}+2, \ldots, P\}$ denote the indices of the remaining rows that are 0 in $\boldsymbol{F}_{col\ \mathcal{W}_1}$. Clearly $\mathcal{U}$ has $P - \frac{P}{\lceil \frac{N}{s}\rceil} = K - M$ indices. We also let $\mathcal{U}^c$ denote the top $\frac{P}{\lceil \frac{N}{s}\rceil}(= P - K + M)$ indices of rows that are non-zero in $\boldsymbol{F}_{col\ \mathcal{W}_1}$. Observe that,

$$\boldsymbol{B}\begin{bmatrix}\boldsymbol{A}_{col\ \mathcal{W}_1}\\ \boldsymbol{Z}_{col\ \mathcal{W}_1}\end{bmatrix} = \begin{bmatrix}\boldsymbol{B}^{\mathcal{U}^c}_{col\ 1:M} & \boldsymbol{B}^{\mathcal{U}^c}_{col\ M+1:K}\\ \boldsymbol{B}^{\mathcal{U}}_{col\ 1:M} & \boldsymbol{B}^{\mathcal{U}}_{col\ M+1:K}\end{bmatrix}\begin{bmatrix}\boldsymbol{A}_{col\ \mathcal{W}_1}\\ \boldsymbol{Z}_{col\ \mathcal{W}_1}\end{bmatrix}$$
$$= \begin{bmatrix}\boldsymbol{F}^{\mathcal{U}^c}_{col\ \mathcal{W}_1}\\ \boldsymbol{F}^{\mathcal{U}}_{col\ \mathcal{W}_1}\end{bmatrix} = \begin{bmatrix}\boldsymbol{F}^{\mathcal{U}^c}_{col\ \mathcal{W}_1}\\ \boldsymbol{0}\end{bmatrix}. \tag{39}$$

This leads to,

$$\boldsymbol{B}^{\mathcal{U}^c}_{col\ 1:M}\boldsymbol{A}_{col\ \mathcal{W}_1} + \boldsymbol{B}^{\mathcal{U}^c}_{col\ M+1:K}\boldsymbol{Z}_{col\ \mathcal{W}_1} = \boldsymbol{F}^{\mathcal{U}^c}_{col\ \mathcal{W}_1}, \text{ and} \tag{40}$$

$$\boldsymbol{B}^{\mathcal{U}}_{col\ 1:M}\boldsymbol{A}_{col\ \mathcal{W}_1} + \boldsymbol{B}^{\mathcal{U}}_{col\ M+1:K}\boldsymbol{Z}_{col\ \mathcal{W}_1} = \boldsymbol{0}. \tag{41}$$

From Equation (41), $\boldsymbol{Z}_{col\ \mathcal{W}_1} = -[\boldsymbol{B}^{\mathcal{U}}_{col\ M+1:K}]^{-1}\boldsymbol{B}^{\mathcal{U}}_{col\ 1:M}\boldsymbol{A}_{col\ \mathcal{W}_1}$ (recall that $[\boldsymbol{B}^{\mathcal{U}}_{col\ M+1:K}]$ is invertible). Substituting this value in Equation (40),

$$(\boldsymbol{B}^{\mathcal{U}^c}_{col\ 1:M} - \boldsymbol{B}^{\mathcal{U}^c}_{col\ M+1:K}[\boldsymbol{B}^{\mathcal{U}}_{col\ M+1:K}]^{-1}\boldsymbol{B}^{\mathcal{U}}_{col\ 1:M})\boldsymbol{A}_{col\ \mathcal{W}_1} = \boldsymbol{F}^{\mathcal{U}^c}_{col\ \mathcal{W}_1}. \tag{42}$$

Next, we will show that *any* M rows of $\boldsymbol{F}^{\mathcal{U}^c}_{col\ \mathcal{W}_1}$ can be linearly combined to generate all the M rows of $\boldsymbol{A}_{col\ \mathcal{W}_1}$. This will be possible if the $\frac{P}{\lceil \frac{N}{s}\rceil} \times M$ matrix $\left(\boldsymbol{B}^{\mathcal{U}^c}_{col\ 1:M} - \boldsymbol{B}^{\mathcal{U}^c}_{col\ M+1:K}[\boldsymbol{B}^{\mathcal{U}}_{col\ M+1:K}]^{-1}\boldsymbol{B}^{\mathcal{U}}_{col\ 1:M}\right)$ is such that *any* M rows are linearly independent.

Consider a sub-matrix formed by picking any M rows of $\left(\boldsymbol{B}^{\mathcal{U}^c}_{col\ 1:M} - \boldsymbol{B}^{\mathcal{U}^c}_{col\ M+1:K}[\boldsymbol{B}^{\mathcal{U}}_{col\ M+1:K}]^{-1}\boldsymbol{B}^{\mathcal{U}}_{col\ 1:M}\right)$. Observe that, this sub-matrix can be represented as:

$$\left(\boldsymbol{B}^{\mathcal{S}}_{col\ 1:M} - \boldsymbol{B}^{\mathcal{S}}_{col\ M+1:K}[\boldsymbol{B}^{\mathcal{U}}_{col\ M+1:K}]^{-1}\boldsymbol{B}^{\mathcal{U}}_{col\ 1:M}\right)$$

where $\mathcal{S} \subset \mathcal{U}^c$ such that $|\mathcal{S}| = M$. Note that, the sub-matrix $\left(\boldsymbol{B}^{\mathcal{S}}_{col\ 1:M} - \boldsymbol{B}^{\mathcal{S}}_{col\ M+1:K}[\boldsymbol{B}^{\mathcal{U}}_{col\ M+1:K}]^{-1}\boldsymbol{B}^{\mathcal{U}}_{col\ 1:M}\right)$ of dimension $M \times M$ is the Schur complement of $\boldsymbol{B}^{\mathcal{U}}_{col\ M+1:K}$ of dimension $(K - M) \times (K - M)$ in the bigger $K \times K$ matrix:

$$\begin{bmatrix}\boldsymbol{B}^{\mathcal{S}}_{col\ 1:M} & \boldsymbol{B}^{\mathcal{S}}_{col\ M+1:K}\\ \boldsymbol{B}^{\mathcal{U}}_{col\ 1:M} & \boldsymbol{B}^{\mathcal{U}}_{col\ M+1:K}\end{bmatrix}.$$

Therefore,

$$\text{Det}\left(\begin{bmatrix}\boldsymbol{B}^{\mathcal{S}}_{col\ 1:M} & \boldsymbol{B}^{\mathcal{S}}_{col\ M+1:K}\\ \boldsymbol{B}^{\mathcal{U}}_{col\ 1:M} & \boldsymbol{B}^{\mathcal{U}}_{col\ M+1:K}\end{bmatrix}\right) =$$
$$\text{Det}(\boldsymbol{B}^{\mathcal{U}}_{col\ M+1:K})\times$$
$$\text{Det}\left(\boldsymbol{B}^{\mathcal{S}}_{col\ 1:M} - \boldsymbol{B}^{\mathcal{S}}_{col\ M+1:K}[\boldsymbol{B}^{\mathcal{U}}_{col\ M+1:K}]^{-1}\boldsymbol{B}^{\mathcal{U}}_{col\ 1:M}\right), \tag{43}$$

where $\text{Det}(\cdot)$ denotes the determinant of a matrix. Because $\begin{bmatrix}\boldsymbol{B}^{\mathcal{S}}_{col\ 1:M} & \boldsymbol{B}^{\mathcal{S}}_{col\ M+1:K}\\ \boldsymbol{B}^{\mathcal{U}}_{col\ 1:M} & \boldsymbol{B}^{\mathcal{U}}_{col\ M+1:K}\end{bmatrix}$ is again a $K \times K$ sub-matrix of $\boldsymbol{B}$ and the matrix $\boldsymbol{B}$ was chosen such that all $K \times K$ sub-matrices are invertible, we have

$$\text{Det}\left(\begin{bmatrix}\boldsymbol{B}^{\mathcal{S}}_{col\ 1:M} & \boldsymbol{B}^{\mathcal{S}}_{col\ M+1:K}\\ \boldsymbol{B}^{\mathcal{U}}_{col\ 1:M} & \boldsymbol{B}^{\mathcal{U}}_{col\ M+1:K}\end{bmatrix}\right) \neq 0,$$

implying

$$\text{Det}\left(\boldsymbol{B}^{\mathcal{S}}_{col\ 1:M} - \boldsymbol{B}^{\mathcal{S}}_{col\ M+1:K}[\boldsymbol{B}^{\mathcal{U}}_{col\ M+1:K}]^{-1}\boldsymbol{B}^{\mathcal{U}}_{col\ 1:M}\right) \neq 0,$$

using Equation (43). Therefore, any M rows of the matrix $\left(\boldsymbol{B}^{\mathcal{U}^c}_{col\ 1:M} - \boldsymbol{B}^{\mathcal{U}^c}_{col\ M+1:K}[\boldsymbol{B}^{\mathcal{U}}_{col\ M+1:K}]^{-1}\boldsymbol{B}^{\mathcal{U}}_{col\ 1:M}\right)$ are always linearly independent. In fact, the matrix $\left(\boldsymbol{B}^{\mathcal{U}^c}_{col\ 1:M} - \boldsymbol{B}^{\mathcal{U}^c}_{col\ M+1:K}[\boldsymbol{B}^{\mathcal{U}}_{col\ M+1:K}]^{-1}\boldsymbol{B}^{\mathcal{U}}_{col\ 1:M}\right)$ is the transpose of the generator matrix of a $(\frac{P}{\lceil \frac{N}{s}\rceil}, M)$ MDS code. □

APPENDIX B
ANALYSIS OF EXPONENTIAL DISTRIBUTIONS

We assume that the time required by a processing node to compute a single dot product follows an exponential distribution and is independent of other parallel processing nodes.

Let us assume, the time required to compute a single dot product of length N, follow the distribution:

$$\Pr(T^{(N)} \leq t) = \begin{cases} 1 - \exp\left(-\mu \ \left(\frac{t}{N} - 1\right)\right) & \forall\ t \geq N \\ 0 & \text{otherwise.} \end{cases} \quad (44)$$

Here, $\mu (> 0)$ is a straggling parameter, that determines the "unpredictable latency" in computation time. We also assume, that if the length of the dot product is s where s is the sparsity of the vector, the probability distribution of the computational time varies as:

$$\Pr(T^{(s)} \leq t) = \begin{cases} 1 - \exp\left(-\mu \ \left(\frac{t}{s} - 1\right)\right) & \forall\ t \geq s \\ 0 & \text{otherwise.} \end{cases} \quad (45)$$

Now we derive the expected computation time using our proposed strategy and compare it with existing strategies in the regimes where the number of dot products M is linear and sub-linear in P.

Table II shows the order-sense expected computation time in the regimes where M is linear and sub-linear in P.

A. Short-Dot Coding Strategy

The computation time over each of the P processing nodes behaves as *iid* exponential random variables following the distribution:

$$\Pr(T^{(s)} \leq t) = 1 - \exp\left(-\mu \ \left(\frac{t}{s} - 1\right)\right) \quad \forall\ t \geq s\ . \quad (46)$$

Now, the expected computation time is the expected value of the K-th order statistic of these P *iid* exponential random variables, which is given by:

$$\mathbf{E}[T_{\text{SD}}] \approx s\left(1 + \frac{\log(\frac{P}{P-K})}{\mu}\right) = \frac{N(P-K+M)}{P}\left(1 + \frac{\log(\frac{P}{P-K})}{\mu}\right). \quad (47)$$

Here we use the result (from [2]) that the K - th order statistic of P *iid* exponential random variables with parameter 1, is given by:

$$\sum_{i=1}^{P} \frac{1}{i} - \sum_{i=1}^{P-K} \frac{1}{i}.$$

For large P and $K < P$, we can approximate the following:

$$\sum_{i=1}^{P} \frac{1}{i} - \sum_{i=1}^{P-K} \frac{1}{i} \approx \log(P) - \log(P-K). \quad (48)$$

Note that the expected computation time is minimized when $K = P - \Theta(M)$, and is given by:

$$\mathbf{E}[T^*_{\text{SD}}] = \mathcal{O}\left(\frac{MN}{P}\left(1 + \frac{\log(P/M)}{\mu}\right)\right). \quad (49)$$

If $M = \Theta(P)$, the expected time is $\mathcal{O}(\frac{MN}{P})$. If $M = o(P)$, the expected time is $\mathcal{O}\left(\frac{MN\log(P/M)}{P}\right)$. Note that $s = \frac{(P-K+M)N}{P}$ is actually an upper bound on the length of each dot product achieved using Short-Dot. Thus the expression obtained in (49) is an upper bound for the actual expected computation time. Thus we use $\mathcal{O}(\cdot)$ instead of $\Theta(\cdot)$.

B. Existing Strategies

1) One Single Processing Node: For one single processing node to compute all M dot products of length N, the computation time is distributed as

$$\Pr(T^{(MN)} \leq t) = 1 - \exp\left(-\mu \ \left(\frac{t}{MN} - 1\right)\right) \ \forall\ t \geq MN.$$

Thus, the expected computation time can be easily derived to be

$$\mathbf{E}[T_{1P}] = MN\left(1 + \frac{1}{\mu}\right). \quad (50)$$

2) Uncoded Parallelization Strategy: Now, consider an uncoded strategy where the computation is simply divided into P dot products and allocated to P processing nodes. We assume that each processing node is computing only one dot product at a time. We wait for all the processing nodes to finish computation. Note that integer effects arise when M does not exactly divide P. Some rows can be divided among $\lceil \frac{P}{M} \rceil$ processing nodes, while the remaining are divided among $\lfloor \frac{P}{M} \rfloor$ processing nodes. Let m_1 and m_2 denote the number of rows that get $\lceil \frac{P}{M} \rceil$ processing nodes and $\lfloor \frac{P}{M} \rfloor$ processing nodes respectively. Clearly the values can be obtained by solving:-

$$\begin{bmatrix} 1 & 1 \\ \lceil \frac{P}{M} \rceil & \lfloor \frac{P}{M} \rfloor \end{bmatrix} \begin{bmatrix} m_1 \\ m_2 \end{bmatrix} = \begin{bmatrix} M \\ P \end{bmatrix}. \quad (51)$$

Now, we have two groups of exponential variables: one group consisting of $m_1 \lceil \frac{P}{M} \rceil$ *iid* exponential random variables of task size $\frac{N}{\lceil \frac{P}{M} \rceil}$, and another group consisting of $m_2 \lfloor \frac{P}{M} \rfloor$ *iid* exponential random variables of task size $\frac{N}{\lfloor \frac{P}{M} \rfloor}$. The two groups are independent of each other. Note that we assume that N is large compared to P and is divisible by $P, \lfloor \frac{P}{M} \rfloor, \lfloor \frac{P}{M} \rfloor$, so that the integer effects with respect to N do not appear and the plots can be scaled with respect to N for ease of understanding.

The expected computation time is thus given by the expectation of the maximum of all these $P = m_1 \lceil \frac{P}{M} \rceil + m_2 \lfloor \frac{P}{M} \rfloor$ exponential random variables.

$$\Pr(T_{\text{UC}} \leq t) = \left(1 - \exp\left(-\mu\left(\frac{\lceil \frac{P}{M} \rceil t}{N} - 1\right)\right)\right)^{m_1 \lceil \frac{P}{M} \rceil} \times \left(1 - \exp\left(-\mu\left(\frac{\lfloor \frac{P}{M} \rfloor t}{N} - 1\right)\right)\right)^{m_2 \lfloor \frac{P}{M} \rfloor} \forall\ t \geq \frac{N}{\lfloor \frac{P}{M} \rfloor}.$$

The expectation is thus obtained as

$$\mathbf{E}[T_{\text{UC}}] = \int_0^\infty (1 - \Pr(T_{\text{UC}} \leq t))\, dt. \tag{52}$$

This expression is numerically computed using MATLAB and plotted in the plot of theoretical computation time in Fig. 9. When M divides P exactly, the expressions are simpler. The computation time for each processing node is distributed as

$$\Pr(T_{\text{UC}} \leq t) = 1 - \exp\left(-\mu \left(\frac{Pt}{MN} - 1\right)\right) \quad \forall\ t \geq \frac{MN}{P}.$$

The expected computation time is the maximum of P such independent and identically distributed random variables, as given by:

$$\mathbf{E}[T_{\text{UC}}] = \frac{MN}{P}\left(1 + \frac{\log(P)}{\mu}\right). \tag{53}$$

The expected time for this uncoded strategy is $\Theta\left(\frac{MN\log(P)}{P}\right)$ regardless of whether M is linear or sub-linear in P. Our strategy Short-Dot thus offers a speed-up of $\Omega(\log(P))$ in expected computation time when M is linear in P, as mentioned in (49), and thus outperforms by a factor that diverges to infinity for large P.

3) Replication Strategy: When a (P, M) replication strategy is used, we separate the matrix into M rows and repeat each row P/M times, so as to obtain a total of P tasks. Note that integer effects arise when M does not exactly divide P. Some rows are repeated $\lceil \frac{P}{M} \rceil$ times, while the remaining are repeated $\lfloor \frac{P}{M} \rfloor$ times. Let m_1 and m_2 denote the number of rows that are repeated $\lceil \frac{P}{M} \rceil$ times and $\lfloor \frac{P}{M} \rfloor$ times respectively. Clearly the values can be obtained by solving:-

$$\begin{bmatrix} 1 & 1 \\ \lceil \frac{P}{M} \rceil & \lfloor \frac{P}{M} \rfloor \end{bmatrix} \begin{bmatrix} m_1 \\ m_2 \end{bmatrix} = \begin{bmatrix} M \\ P \end{bmatrix}. \tag{54}$$

Now, the minimum of $\lceil \frac{P}{M} \rceil$ (or similarly $\lfloor \frac{P}{M} \rfloor$) *iid* exponential random variables is also exponential with parameter scaled by $\lceil \frac{P}{M} \rceil$ (or similarly $\lfloor \frac{P}{M} \rfloor$). The expected computation time is thus given by the expectation of the maximum of m_1 independent exponential variables with parameter scaled by $\lceil \frac{P}{M} \rceil$ and m_2 independent exponential variables with parameter scaled by $\lfloor \frac{P}{M} \rfloor$.

$$\Pr(T_{\text{RP}} \leq t) = \left(1 - \exp\left(-\mu \left\lceil \frac{P}{M} \right\rceil \left(\frac{t}{N} - 1\right)\right)\right)^{m_1} \times \left(1 - \exp\left(-\mu \left\lfloor \frac{P}{M} \right\rfloor \left(\frac{t}{N} - 1\right)\right)\right)^{m_2} \quad \forall\ t \geq N.$$

The expectation is thus obtained as:

$$\mathbb{E}[T_{\text{RP}}] = \int_0^\infty (1 - \Pr(T_{\text{RP}} \leq t))\, dt. \tag{55}$$

This expression is computed using MATLAB in the plot of theoretical expected computation time (Fig. 9). When M exactly divides P, the analysis is simpler, and the two types of exponential distributions are identical. Following an analysis similar to [2], it simplifies to the expectation of the maximum of M *iid* exponential random variables, each of which is the minimum of P/M *iid* exponential random variables.

$$\mathbf{E}[T_{\text{RP}}] = N\left(1 + \frac{M\log(M)}{P\mu}\right). \tag{56}$$

When M is linear in P, the expected computation time is $\Theta(\frac{MN}{P}\log(P))$ while our strategy achieves $\mathcal{O}(N)$ in this regime. When M is sub-linear in P, the expected computation time is $\Theta(N)$ while Short-Dot achieves $\mathcal{O}\left(\frac{MN\log(P/M)}{P}\right)$ that offers speed-up by a factor diverging to infinity.

4) MDS Coding Strategy: The matrix is separated into M rows and coded into P rows using a (P, M) MDS code. Thus, each processing node effectively computes a dot product of length N. We have to wait for any M processing nodes to finish. Assuming the computation of each processing node is independent, following an analysis similar to [2], we obtain that,

$$\mathbf{E}[T_{\text{MDS}}] = N\left(1 + \frac{\log(P)}{\mu} - \frac{\log(P - M)}{\mu}\right). \tag{57}$$

When M is linear in P, the expected computation time is $\Theta(N)$ as compared to our strategy that achieves $\mathcal{O}\left(\frac{MN}{P}\right)$. However, in the regime where M is sub-linear in P, the expected computation time is also $\Theta(N)$ while our strategy achieves $\mathcal{O}\left(\frac{MN\log(P/M)}{P}\right)$, and thus outperforms MDS codes by a factor that diverges to infinity for large P.

Appendix C
Order Statistics

In this appendix, we include two important results on order statistics from [22] that are used in the derivation of expected computation time for Short-Dot and Short-MDS codes. Let us denote the pdf and cdf of an exponential random variable with parameter μ be $f(t) = \mu e^{-\mu t}$ and $F(t) = 1 - e^{-\mu t}$. Then the following results hold.

Lemma 3 (Central Order Statistics)**.** *For a fixed r lying in $(0, 1)$, let $T'_{(rn)}$ denote the (rn)-th order statistic out of n iid random variables. Then,*

$$T'_{(rn)} \xrightarrow{p} \mathcal{N}\left(t_r, \frac{r(1-r)}{nf^2(t_r)}\right),$$

where $\xrightarrow{p}$ denotes convergence in probability and $t_r = F^{-1}(r)$.

Lemma 4 (Maximum of normal random variables)**.** *The expectation of the maximum of n iid random variables, each with distribution $\mathcal{N}(\mu, \sigma^2)$, is approximately equal to $\mu + \sigma\sqrt{2\log n}$ for large n.*

Appendix D
Short-Dot for Errors Instead of Erasures

Proof of Theorem 6. Recall the construction of $\boldsymbol{F}_{P\times M}$.

$$\boldsymbol{F}_{P\times M} = \boldsymbol{B}_{P\times K}\widetilde{\boldsymbol{A}}_{K\times N}. \tag{58}$$

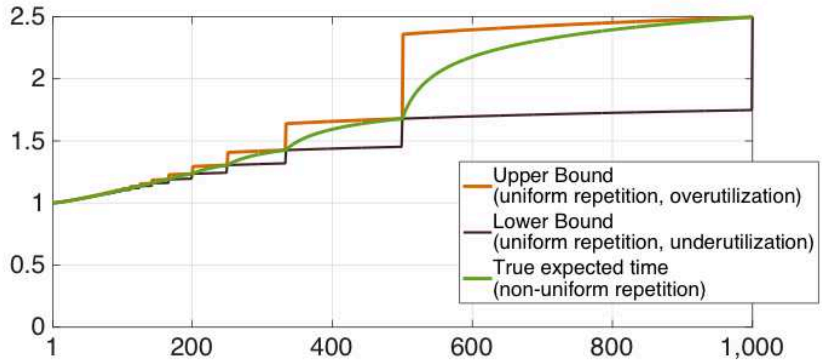


Fig. 12. Theoretical plot of expected computation time of replication taking integer effects into account: straggling parameter $\mu = 5$, total processing nodes $P = 1000$ and number of dot products M is varied from 1 to P.

Here $\widetilde{\boldsymbol{A}}_{K\times N}$ is the matrix formed by appending $(K - M)$ rows after $\boldsymbol{A}_{M\times N}$, so as to enforce desired sparsity on $\boldsymbol{F}_{P\times M}$. And $\boldsymbol{B}_{P\times K}$ is a matrix such that every square sub-matrix is invertible. Now assume we obtain erroneous dot products in the form

$$\boldsymbol{y}_{P\times 1} = \boldsymbol{F}_{P\times M}\boldsymbol{x}_{N\times 1} + \boldsymbol{e}_{P\times 1}, \tag{59}$$

where $\boldsymbol{e}_{P\times 1}$ denotes the error vector that corrupts some of the values of $\boldsymbol{F}_{P\times M}\boldsymbol{x}_{N\times 1}$.

$$\begin{aligned}\boldsymbol{y}_{P\times 1} &= \boldsymbol{F}_{P\times M}\boldsymbol{x}_{N\times 1} + \boldsymbol{e}_{P\times 1}\\ &= \boldsymbol{B}_{P\times K}\widetilde{\boldsymbol{A}}_{K\times N}\boldsymbol{x}_{N\times 1} + \boldsymbol{e}_{P\times 1}\\ &= \boldsymbol{B}_{P\times K}\boldsymbol{q}_{K\times 1} + \boldsymbol{e}_{P\times 1}.\end{aligned} \tag{60}$$

Here we define $\boldsymbol{q}_{K\times 1} = \widetilde{\boldsymbol{A}}_{K\times N}\boldsymbol{x}_{N\times 1}$ as a $K \times 1$ vector such that its first M values correspond to the M dot products we originally wanted to compute. Thus, reconstructing $\boldsymbol{q}$ in the presence of errors is sufficient to reconstruct our M original dot products. We first claim that there exists a unique $\boldsymbol{q}$ such that $\boldsymbol{y} = \boldsymbol{B}\boldsymbol{q} + \boldsymbol{e}$ as long as $||\boldsymbol{e}||_0 \leq \frac{(P-K)}{2}$ and $\boldsymbol{B}$ is a matrix such that every square sub-matrix is invertible. Since $\boldsymbol{B}$ is full rank, there exists a full-rank annihilating matrix $\boldsymbol{H}$ of dimension $(P - K) \times P$ such that $\boldsymbol{H}\boldsymbol{B} = 0$. Thus,

$$\tilde{\boldsymbol{y}} = \boldsymbol{H}\boldsymbol{y} = \boldsymbol{H}(\boldsymbol{B}\boldsymbol{q} + \boldsymbol{e}) = \boldsymbol{H}\boldsymbol{e}. \tag{61}$$

Now, if we can uniquely find $\boldsymbol{e}$, then $\boldsymbol{q}$ can also be uniquely found since $\boldsymbol{B}\boldsymbol{q} = \boldsymbol{y} - \boldsymbol{e}$ and $\boldsymbol{B}$ is full-rank. Thus we are only required to prove the uniqueness of $\boldsymbol{e}$. We will use the following lemmas.

Lemma 5. *There exists a unique vector $\boldsymbol{e}$ such that $\tilde{\boldsymbol{y}} = \boldsymbol{H}\boldsymbol{e}$ if and only if*

$$||\boldsymbol{e}||_0 < \frac{\text{Spark}(\boldsymbol{H})}{2}.$$

Proof of Lemma 5. Recall $\text{Spark}(\boldsymbol{H})$ is the minimum number of columns of $\boldsymbol{H}$ that are linearly dependent. For contradiction, assume there exists two vectors $\boldsymbol{e}_1$ and $\boldsymbol{e}_2$ such that $||\boldsymbol{e}_1||_0 < \frac{\text{Spark}(\boldsymbol{H})}{2}$ and $||\boldsymbol{e}_2||_0 < \frac{\text{Spark}(\boldsymbol{H})}{2}$. We also assume that, $\tilde{\boldsymbol{y}} = \boldsymbol{H}\boldsymbol{e}_1 = \boldsymbol{H}\boldsymbol{e}_2$.

Thus, $\boldsymbol{H}(\boldsymbol{e}_1 - \boldsymbol{e}_2) = 0$. Or, $(\boldsymbol{e}_1 - \boldsymbol{e}_2)$ is a non-zero vector, that lies in the null space of $\boldsymbol{H}$. Now

$$\begin{aligned}||\boldsymbol{e}_1 - \boldsymbol{e}_2||_0 &\leq |\text{Support}(\boldsymbol{e}_1) \cup \text{Support}(\boldsymbol{e}_2)|\\ &\leq |\text{Support}(\boldsymbol{e}_1)| + |\text{Support}(\boldsymbol{e}_2)|\\ &< \text{Spark}(\boldsymbol{H}).\end{aligned} \tag{62}$$

Thus there exists a non-zero vector in the null space of $\boldsymbol{H}$ such that its number of non-zero entries is less than $\text{Spark}(\boldsymbol{H})$. Contradiction! Thus, $\boldsymbol{e}_1 = \boldsymbol{e}_2$. This means that if there is a vector $\boldsymbol{e}$ such that $||\boldsymbol{e}||_0 < \frac{\text{Spark}(\boldsymbol{H})}{2}$, then it is always unique. Now we prove it the other way round.

Assume that there exists a unique $\boldsymbol{e}$ such that $\tilde{\boldsymbol{y}} = \boldsymbol{H}\boldsymbol{e}$. For contradiction, assume that $\text{Spark}(\boldsymbol{H}) \leq 2||\boldsymbol{e}||_0$. Then, there exists a non-zero vector $\boldsymbol{\delta}$ of sparsity at most $2||\boldsymbol{e}||_0$ such that $\boldsymbol{H}\boldsymbol{\delta} = 0$. Let $\boldsymbol{e}_0 = \boldsymbol{e} + \boldsymbol{\delta}$. Then, $\tilde{\boldsymbol{y}} = \boldsymbol{H}\boldsymbol{e} = \boldsymbol{H}(\boldsymbol{e}_0 - \boldsymbol{\delta}) = \boldsymbol{H}\boldsymbol{e}_0$. Thus $\boldsymbol{e}$ is not the unique solution. Contradiction! Thus, $\text{Spark}(\boldsymbol{H}) > 2||\boldsymbol{e}||_0$ only if $\boldsymbol{e}$ is unique. □

Lemma 6. *Let $\boldsymbol{B}$ be a matrix of dimension $P \times K$, such that every square sub-matrix of $\boldsymbol{B}$ is invertible. If $\boldsymbol{H}$ is chosen as a full-rank annihilating matrix of $\boldsymbol{B}$ of dimension $(P - K) \times P$, then* $\text{Spark}(\boldsymbol{H}) = (P - K + 1)$.

Proof of Lemma 6. Note that, the rank of $\boldsymbol{H}$ is $(P - K)$ as $\boldsymbol{H}$ is full rank. (From rank-nullity theorem, there always exists $(P - K)$ linearly independent row-vectors such that each row multiplied with $\boldsymbol{B}$ gives 0. From the condition of the theorem, $\boldsymbol{H}$ is chosen as the matrix of all those $(P - K)$ linearly independent rows.) Thus, any $(P - K + 1)$ columns of $\boldsymbol{H}$ is always linearly dependent. Or, $\text{Spark}(\boldsymbol{H}) \leq (P - K + 1)$. Now, to prove the equality, let us assume for contradiction that, $\text{Spark}(\boldsymbol{H}) < (P - K + 1)$, i.e., $\text{Spark}(\boldsymbol{H})$ is strictly less than $(P - K + 1)$. Thus, there exists a non-zero vector $\boldsymbol{v}$ such that $||\boldsymbol{v}||_0 < (P - K + 1)$ and $\boldsymbol{H}\boldsymbol{v} = 0$.

Now, note that dimension of null-space of $\boldsymbol{H}$ is K as the K linearly independent columns of the matrix $\boldsymbol{B}$ lie in the null-space of $\boldsymbol{H}$. Thus, the K columns of $\boldsymbol{B}$ form a basis for the null space of $\boldsymbol{H}$. Thus, the columns of $\boldsymbol{B}$ can be linearly combined to generate $\boldsymbol{v}$, a vector in null-space of $\boldsymbol{H}$ with $||\boldsymbol{v}||_0 < (P - K + 1)$. Let

$$\boldsymbol{B}\boldsymbol{\alpha} = \boldsymbol{v}. \tag{63}$$

The vector $\boldsymbol{v}$ has strictly more than $(P-(P-K+1))$ zeros, i.e., $K-1$ zeros. Or, $\boldsymbol{v}$ has at least K zeros. Let $\{l_1, l_2, \ldots, l_K\}$ denote any K indices of $\boldsymbol{v}$ that are zero. Consider the $(K \times K)$ sub-matrix consisting of the rows of $\boldsymbol{B}$ indexed in $\{l_1, l_2, \ldots, l_K\}$ as $\tilde{\boldsymbol{B}}$. Clearly $\tilde{\boldsymbol{B}}\boldsymbol{\alpha} = 0$, which contradicts that every square sub-matrix of $\boldsymbol{B}$ is invertible. Contradiction! Thus, $\text{Spark}(\boldsymbol{H}) = (P-K+1)$. □

From Lemmas 5 and 6 above, we can thus say that if $2||\boldsymbol{e}||_0 < (P-K+1)$ or, $2||\boldsymbol{e}||_0 \leq (P-K)$, then there exists a unique vector $\boldsymbol{e}$ that satisfies $\tilde{\boldsymbol{y}} = \boldsymbol{H}\boldsymbol{e}$, and thus, there exists a unique $\boldsymbol{q}$ such that $\boldsymbol{y} = \boldsymbol{B}\boldsymbol{q} + \boldsymbol{e}$. As $||\boldsymbol{e}||_0$ is an integer, the bound becomes $||\boldsymbol{e}||_0 \leq \lfloor \frac{P-K}{2} \rfloor$. □

ACKNOWLEDGMENTS

We thank Yaoqing Yang, Haewon Jeong, Praveen Venkatesh, Mohammad Fahim, Farzin Haddadpour, Ankur Mallick, Gauri Joshi and Tze Meng Low for valuable discussions. S Dutta also received the Prabhu and Poonam Goel Graduate Fellowship.

REFERENCES

[1] S. Dutta, V. Cadambe, and P. Grover, "Short-dot: Computing large linear transforms distributedly using coded short dot products," in *Proc. Adv. Neural Inf. Process. Syst. (NIPS)*, 2016, pp. 2092–2100.

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

[3] D. Wang, G. Joshi, and G. Wornell, "Using straggler replication to reduce latency in large-scale parallel computing," *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 43, no. 3, pp. 7–11, 2015.

[4] D. Wang, G. Joshi, and G. Wornell, "Efficient task replication for fast response times in parallel computation," *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 42, no. 1, pp. 599–600, Jun. 2014.

[5] G. Joshi, Y. Liu, and E. Soljanin, "On the delay-storage trade-off in content download from coded distributed storage systems," *IEEE J. Sel. Areas Commun.*, vol. 32, no. 5, pp. 989–997, May 2014.

[6] I. Nahlus, E. P. Kim, N. R. Shanbhag, and D. Blaauw, "Energy-efficient dot product computation using a switched analog circuit architecture," in *Proc. IEEE/ACM Int. Symp. Low Power Electron. Design (ISLPED)*, Aug. 2014, pp. 315–318.

[7] N. C. Wang, S. K. Gonugondla, I. Nahlus, N. Shanbhag, and E. Pop, "GDOT: A graphene-based nanofunction for dot-product computation," in *Proc. IEEE Symp. VLSI Technol.*, 2016, pp. 1–2.

[8] A. Geist, "Supercomputing's monster in the closet," *IEEE Spectr.*, vol. 53, no. 3, pp. 30–35, Mar. 2016.

[9] D. William, "High-performance hardware for machine learning," in *Proc. Adv. Neural Inf. Process. Syst. (NIPS)*, 2015, pp. 1–120.

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

[11] V. Kumar, A. Grama, G. Anshul, and G. Karypis, *Introduction to Parallel Computing: Design Anal. Algorithms*. Redwood City, CA, USA: Benjamin/Cummings Publishing Company, Inc., 1994.

[12] G. C. Fox, S. W. Otto, and A. J. Hey, "Matrix algorithms on a hypercube I: Matrix multiplication," *Parallel Comput.*, vol. 4, no. 1, pp. 17–31, Feb. 1987.

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

[14] G. Ballard, J. Demmel, O. Holtz, and O. Schwartz, "Communication costs of Strassen's matrix multiplication," *Commun. ACM*, vol. 57, no. 2, pp. 107–114, Feb. 2014.

[15] T. Herault and Y. Robert, *Fault-Tolerance Techniques for High-Performance Computing*. Berlin, Germany: Springer, 2015.

[16] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The Hadoop distributed file system," in *Proc. IEEE Symp. Mass Storage Syst. Technol. (MSST)*, May 2010, pp. 1–10.

[17] K.-H. Huang and J. A. Abraham, "Algorithm-based fault tolerance for matrix operations," *IEEE Trans. Comput.*, vol. 100, no. 6, pp. 518–528, Jun. 1984.

[18] W. Ryan and S. Lin, *Channel Codes: Classical and Modern*. Cambridge, U.K.: Cambridge Univ. Press, 2009.

[19] S. Li, M. A. Maddah-Ali, and A. S. Avestimehr, "Coded MapReduce," in *Proc. 53rd Annu. Allerton Conf. Commun., Control, Computing (Allerton)*, Sep./Oct. 2015, pp. 964–971.

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

[21] V. Cadambe and P. Grover, "Codes for distributed computing: A tutorial," *IEEE Inf. Theory Soc. News Lett.*, vol. 67, no. 4, pp. 3–15, Dec. 2017.

[22] K. Lee, C. Suh, and K. Ramchandran, "High-dimensional coded matrix multiplication," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Jun. 2017, pp. 2418–2422.

[23] G. Suh, K. Lee, and C. Suh, "Matrix sparsification for coded matrix multiplication," in *Proc. 55th Annu. Allerton Conf. Commun., Control, Comput. (Allerton)*, Oct. 2017, pp. 1271–1278.

[24] S. Li, M. A. Maddah-Ali, and A. S. Avestimehr, "A unified coding framework for distributed computing with straggling servers," in *Proc. IEEE Globecom Workshops (GC Wkshps)*, Dec. 2016, pp. 1–6.

[25] T. Baharav, K. Lee, O. Ocal, and K. Ramchandran, "Straggler-proofing massive-scale distributed matrix multiplication with D-dimensional product codes," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Jun. 2018, pp. 1993–1997.

[26] K. Lee, R. Pedarsani, D. Papailiopoulos, and K. Ramchandran, "Coded computation for multicore setups," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Jun. 2017, pp. 2413–2417.

[27] Q. Yu, M. A. Maddah-Ali, and A. S. Avestimehr, "Polynomial codes: An optimal design for high-dimensional coded matrix multiplication," in *Proc. Adv. Neural Inf. Process. Syst. (NIPS)*, 2017, pp. 1–11.

[28] M. Fahim, H. Jeong, F. Haddadpour, S. Dutta, V. Cadambe, and P. Grover, "On the optimal recovery threshold of coded matrix multiplication," in *Proc. 55th Annu. Allerton Conf. Commun., Control, Comput. (Allerton)*, Oct. 2017, pp. 1264–1270.

[29] S. Dutta, M. Fahim, F. Haddadpour, H. Jeong, V. Cadambe, and P. Grover, "On the optimal recovery threshold of coded matrix multiplication," *IEEE Trans. Inf. Theory*, to be published.

[30] S. Dutta, Z. Bai, H. Jeong, T. M. Low, and P. Grover, "A unified coded deep neural network training strategy based on generalized polydot codes," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Jun. 2018, pp. 1585–1589.

[31] Q. Yu, M. A. Maddah-Ali, and A. S. Avestimehr, "Straggler mitigation in distributed matrix multiplication: Fundamental limits and optimal coding," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Jun. 2018, pp. 2022–2026.

[32] S. Wang, J. Liu, and N. Shroff, "Coded sparse matrix multiplication," in *Proc. Int. Conf. Mach. Learn. (ICML)*, Apr. 2018, pp. 5139–5147.

[33] S. Wang, J. Liu, N. Shroff, and P. Yang, "Fundamental limits of coded linear transform," 2018, *arXiv:1804.09791*. [Online]. Available: https://arxiv.org/abs/1804.09791

[34] A. Mallick, M. Chaudhari, and G. Joshi, "Fast and efficient distributed matrix-vector multiplication using rateless fountain codes," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, May 2019, pp. 8192–8196.

[35] A. Severinson, A. G. I. Amat, and E. Rosnes, "Block-diagonal and LT codes for distributed computing with straggling servers," *IEEE Trans. Commun.*, vol. 67, no. 3, pp. 1739–1753, Mar. 2019.

[36] F. Haddadpour and V. R. Cadambe, "Codes for distributed finite alphabet matrix-vector multiplication," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Jun. 2018, pp. 1625–1629.

[37] F. Haddadpour, Y. Yang, V. Cadambe, and P. Grover, "Cross-iteration coded computing," in *Proc. 56th Annu. Allerton Conf. Commun., Control, Comput. (Allerton)*, Oct. 2018, pp. 196–203.

[38] R. Tandon, Q. Lei, A. G. Dimakis, and N. Karampatziakis, "Gradient Coding," in *Proc. Adv. Neural Inf. Process. Syst. (NIPS)*, 2016, pp. 1–19.

[39] R. Tandon, Q. Lei, A. G. Dimakis, and N. Karampatziakis, "Gradient coding: Avoiding stragglers in distributed learning," in *Proc. Int. Conf. Mach. Learn. (ICML)*, Jul. 2017, pp. 3368–3376.

[40] N. Raviv, R. Tandon, A. Dimakis, and I. Tamo, "Gradient coding from cyclic MDS codes and expander graphs," in *Proc. Int. Conf. Mach. Learn. (ICML)*, Jul. 2018, pp. 4302–4310.

[41] W. Halbawi, N. Azizan-Ruhi, F. Salehi, and B. Hassibi, "Improving distributed gradient descent using Reed–Solomon codes," 2017, *arXiv:1706.05436*. [Online]. Available: https://arxiv.org/abs/1706.05436

[42] M. Ye and E. Abbe, "Communication-computation efficient gradient coding," in *Proc. Int. Conf. Mach. Learn. (ICML)*, Feb. 2018, pp. 5606–5615.
[43] A. Reisizadeh and R. Pedarsani, "Latency analysis of coded computation schemes over wireless networks," in *Proc. 55th Annu. Allerton Conf. Commun., Control, Comput. (Allerton)*, Oct. 2017, pp. 1256–1263.
[44] A. Reisizadeh, S. Prakash, R. Pedarsani, and A. S. Avestimehr, "Coded computation over heterogeneous clusters," *IEEE Trans. Inf. Theory*, vol. 65, no. 7, pp. 4227–4242, Jul. 2019.
[45] M. F. Aktas, P. Peng, and E. Soljanin, "Effective straggler mitigation: Which clones should attack and when?" *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 45, no. 2, pp. 12–14, 2017.
[46] M. F. Aktas, P. Peng, and E. Soljanin, "Straggler mitigation by delayed relaunch of tasks," *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 45, no. 3, pp. 224–231, 2018.
[47] S. Dutta, V. Cadambe, and P. Grover, "Coded convolution for parallel and distributed computing within a deadline," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Jun. 2017, pp. 2403–2407.
[48] Y. Yang, P. Grover, and S. Kar, "Fault-tolerant parallel linear filtering using compressive sensing," in *Proc. IEEE Int. Symp. Turbo Codes Iterative Inf. Process. (ISTC)*, Sep. 2016, pp. 201–205.
[49] Y. Yang, P. Grover, and S. Kar, "Fault-tolerant distributed logistic regression using unreliable components," in *Proc. 54th Annu. Allerton Conf. Commun., Control, Comput. (Allerton)*, Sep. 2016, pp. 940–947.
[50] Y. Yang, P. Grover, and S. Kar, "Computing linear transformations with unreliable components," *IEEE Trans. Inf. Theory*, vol. 63, no. 6, pp. 3729–3756, Jun. 2017.
[51] S. Li, M. A. Maddah-Ali, Q. Yu, and A. S. Avestimehr, "A fundamental tradeoff between computation and communication in distributed computing," *IEEE Trans. Inf. Theory*, vol. 64, no. 1, pp. 109–128, Jan. 2018.
[52] N. Azizan-Ruhi, F. Lahouti, A. S. Avestimehrm, and B. Hassibi, "Distributed solution of large-scale linear systems via accelerated projection-based consensus," *IEEE Trans. Signal Process.*, vol. 67, no. 14, pp. 3806–3817, 2019.
[53] L. Huang, S. Pawar, H. Zhang, and K. Ramchandran, "Codes can reduce queueing delay in data centers," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Jul. 2012, pp. 2766–2770.
[54] K. Lee, N. B. Shah, L. Huang, and K. Ramchandran, "The MDS queue: Analysing the latency performance of erasure codes," *IEEE Trans. Inf. Theory*, vol. 63, no. 5, pp. 2822–2842, May 2017.
[55] H. Jeong, T. M. Low, and P. Grover, "Masterless coded computing: A fully-distributed coded FFT algorithm," in *Proc. 56th Annu. Allerton Conf. Commun., Control, Comput. (Allerton)*, Oct. 2018, pp. 887–894.
[56] Q. Yu, M. A. Maddah-Ali, and A. S. Avestimehr, "Coded Fourier transform," 2017, *arXiv:1710.06471*. [Online]. Available: https://arxiv.org/abs/1710.06471
[57] S. Dutta, Z. Bai, T. M. Low, and P. Grover, "CodeNet: Training large scale neural networks in presence of soft-errors," in *Proc. Workshop Coding Theory Large-Scale Mach. Learn., Int. Conf. Mach. Learn. (ICML)*, Mar. 2019, pp. 1–54.
[58] Y. Yang, P. Grover, and S. Kar, "Coded distributed computing for inverse problems," in *Proc. Adv. Neural Inf. Process. Syst. (NIPS)*, 2017, pp. 709–719.
[59] C. Karakus, Y. Sun, and S. Diggavi, "Encoded distributed optimization," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Jun. 2017, pp. 2890–2894.
[60] Y. Yang, P. Grover, and S. Kar, "Coding for a single sparse inverse problem," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Jun. 2018, pp. 1575–1579.
[61] C. Karakus, Y. Sun, S. Diggavi, and W. Yin, "Straggler mitigation in distributed optimization through data encoding," in *Proc. Adv. Neural Inf. Process. Syst. (NIPS)*, 2017, pp. 5440–5448.
[62] S. Liu, A. Vempaty, M. Fardad, E. Masazade, and P. K. Varshney, "Energy-aware sensor selection in field reconstruction," *IEEE Signal Process. Lett.*, vol. 21, no. 12, pp. 1476–1480, Dec. 2014.
[63] R. M. Roth, "Fault-tolerant dot-product engines," *IEEE Trans. Inf. Theory*, vol. 65, no. 4, pp. 2046–2057, Apr. 2019.
[64] H. T. Kung, "Fast evaluation and interpolation," Dept. Comput. Sci., Carnegie Mellon Univ., Pittsburgh, PA, USA, Tech. Rep., 1973.
[65] L. Li, "On the arithmetic operational complexity for solving Vandermonde linear equations," *Jpn. J. Ind. Appl. Math.*, vol. 17, no. 1, pp. 15–18, Feb. 2000.
[66] E. J. Candès and T. Tao, "Decoding by linear programming," *IEEE Trans. Inf. Theory*, vol. 51, no. 12, pp. 4203–4215, Dec. 2005.

Sanghamitra Dutta (S'15) received her B.Tech in electronics and electrical communication engineering from the Indian Institute of Technology, Kharagpur, India, in 2015.

She is a doctoral candidate in the Department of Electrical and Computer Engineering at Carnegie Mellon University, PA, USA. She was a summer research intern at the IBM TJ Watson Research Center from May 2017 to August 2017. Her main contributions to science are towards developing novel erasure-codes for reliable computing in presence of faults, stragglers and errors, and deriving fundamental information-theoretic limits on their performance. She is interested in novel algorithmic solutions for reliable and trustworthy machine learning, that address computational challenges of large-scale machine learning as well as the emerging trust issues concerning fairness and privacy.

Ms. Dutta is a recipient of the 2019 Axel Berny Presidential Graduate Fellowship, 2017 Tan Endowed Graduate Fellowship, 2016 Prabhu and Poonam Goel Graduate Fellowship and the 2014 HONDA Young Engineer and Scientist Award.

Viveck Cadambe (M'06) received his Ph.D in electrical and computer engineering from the University of California, Irvine, CA, USA in 2011. He received his B.Tech and M.Tech in electrical engineering from the Indian Institute of Technology Madras, Chennai, India, in 2006.

He is an Assistant Professor in the Department of Electrical Engineering at Pennsylvania State University, University Park, PA USA. Between 2011 and 2014, he was a postdoctoral researcher, jointly with the Electrical and Computer Engineering (ECE) department at Boston University, and the Research Laboratory of Electronics (RLE) at the Massachusetts Institute of Technology (MIT). His research uses tools of information theory, error correcting codes and theory of distributed systems to understand fundamental engineering trade-offs in data communication, storage and computing systems.

Dr. Cadambe is a recipient of the 2009 IEEE Information Theory Society Best Paper Award, 2011 CPCC Best Dissertation Award from University of California, Irvine, the 2014 IEEE International Symposium on Network Computing and Applications (NCA) Best Paper Award, the 2015 NSF CRII Award, the 2016 NSF Career Award and a finalist for the 2016 Bell Labs Prize. He has served as an Associate Editor for the IEEE TRANSACTIONS ON WIRELESS COMMUNICATIONS since December 2014.

Pulkit Grover (SM'16) Ph.D. UC Berkeley'10. He is an Associate Professor at CMU in Electrical and Computer Engineering and Carnegie Mellon Neurosciences Institute. His main contributions to science are towards developing and experimentally validating a new theory of information (fundamental limits, practical designs) for optimizing designs of artificial, as well as understanding designs of biological, communication and computing systems. This includes developing formal mathematical tools for estimating flows of information and minimizing energy in computing. Pulkit received the 2010 best student paper award at IEEE Conference on Decision and Control; the 2011 Eli Jury Dissertation Award from UC Berkeley; the 2012 IEEE Leonard G. Abraham journal paper award; a 2014 NSF CAREER award; a 2015 Google Research Award; and the 2019 Best Tutorial Paper Award from IEEE ComSoc. In 2018, he received the inaugural award from the Chuck Noll Foundation for Brain Injury Research and the Joel and Ruth Spira Excellence in Teaching Award.