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

Sanghamitra Dutta  
Carnegie Mellon University  
sanghand@andrew.cmu.edu  

Viveck Cadambe  
Pennsylvania State University  
viveck@engr.psu.edu  

Pulkit Grover  
Carnegie Mellon University  
pgrover@andrew.cmu.edu  

Abstract

Faced with saturation of Moore’s law and increasing size and dimension of data, system designers have increasingly resorted to parallel and distributed computing to reduce computation time of machine-learning algorithms. However, distributed computing is often bottlenecked by a small fraction of slow processors called “stragglers” that reduce the speed of computation because the fusion node has to wait for all processors to complete their processing. To combat the effect of stragglers, recent literature proposes introducing redundancy in computations across processors, e.g., using repetition-based strategies or erasure codes. The fusion node can exploit this redundancy by completing the computation using outputs from only a subset of the processors, ignoring the stragglers. In this paper, we propose a novel technique – that we call “Short-Dot” – to introduce redundant computations in a coding theory inspired fashion, for computing linear transforms of long vectors. Instead of computing long dot products as required in the original linear transform, we construct a larger number of redundant and short dot products that can be computed faster and more efficiently at individual processors. In reference to comparable schemes that introduce redundancy to tackle stragglers, Short-Dot reduces the cost of computation, storage and communication since shorter portions are stored and computed at each processor, and also shorter portions of the input is communicated to each processor. Further, only a subset of these short dot products are required at the fusion node to finish the computation successfully, thus enabling us to ignore stragglers. We demonstrate through probabilistic analysis as well as experiments on computing clusters that Short-Dot offers significant speed-up compared to existing techniques. We also derive trade-offs between the length of the dot-products and the resilience to stragglers (number of processors to wait for), for any such strategy and compare it to that achieved by our strategy.

1 Introduction

This work proposes a coding-theory inspired computation technique for speeding up computing linear transforms of high-dimensional data by distributing it across multiple processing units 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 processors that bottleneck the entire computation. To address this problem, we provide techniques (building on [1] [2] [3] [4] [5]) that introduce redundancy in the computation by designing a novel error-correction mechanism that allows the size of individual dot products computed at each processor to be shorter than the length of the input. Shorter dot products offer advantages in computation, storage and communication in distributed linear transforms.

The problem of computing linear transforms of high-dimensional vectors is “the” critical step 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 solutions to various machine learning problems, e.g., regression and classification etc., and are also used in acquiring and pre-processing the data through Fourier transforms, wavelet transforms, filtering, etc. Fast and reliable computation of linear transforms are thus a necessity for low-latency inference [6]. Due to saturation of Moore’s law, increasing speed of computing in a single processor is becoming difficult, forcing practitioners to adopt parallel processing to speed up computing for ever increasing data dimensions and sizes.

Classical approaches of computing linear transforms across parallel processors, e.g., Block-Striped Decomposition [7], Fox’s method [7,8], and Cannon’s method [7,8], rely on dividing the computational task equally among all available processors [4] without any redundant computation. The fusion node collects the outputs from each processors to complete the computation and thus has to wait for all the processors to finish. In almost all distributed systems, a few slow or faulty processors – called “stragglers” [11] – are observed to delay the entire computation. This unpredictable latency in distributed systems is attributed to factors such as network latency, shared resources, maintenance activities, and power limitations. In order to combat with stragglers, cloud computing frameworks like Hadoop [12] employ various straggler detection techniques and usually reset the task allotted to stragglers. Forward error-correction techniques offer an alternative approach to deal with this “straggler effect” by introducing redundancy in the computational tasks across different processors. The fusion node now requires outputs from only a subset of all the processors to successfully finish. In this context, the use of preliminary erasure codes dates back to the ideas of algorithmic fault tolerance [13,14]. Recently optimized Repetition and Maximum Distance Separable (MDS) [15] codes have been explored [2,3,1,16] to speed up computations.

We consider the problem of computing $Ax$ where $A_{(M \times N)}$ is a given matrix and $x_{(N \times 1)}$ is a vector that is input to the computation ($M \ll N$). In contrast with [1], which also uses codes to compute linear transforms in parallel, we allow the size of individual dot products computed at each processor to be smaller than $N$, the length of the input.

**Why might one be interested in computing short dot products while performing an overall large linear transform?**

One reason is straightforward: the computation time depends on the length of the dot-products computed. Processors are also inherently memory limited, which limits the size of dot products that can be computed. In some distributed and cloud computing systems, the computation time is dominated by the time taken to communicate $x$ to the processors. In systems where multi-casting $x$ is not possible or is inefficient, it may be faster to communicate a subset of the co-ordinates of $x$ to each processor. In such systems, we anticipate that communicating shorter vectors, each formed by these subsets of coordinates of $x$, is likely to result in substantial speedups over schemes that require the entire $x$ vector (in particular when multi-casting is difficult) [3]. In Sections 4 and 6, we show both theoretically (under model assumptions inspired from [1] that admit simplified expected time analysis while being a crude approximation of experimental observations) and experimentally that the speed-up using Short-Dot can be increased beyond that obtained using the strategy proposed in [1], in straggler-prone environments.

To summarize, our main contributions are:

1. To compute $Ax$ for a given matrix $A_{(M \times N)}$, we instead compute $Fx$ where we construct $F_{(P \times N)}$ (total no. of processors $P >$ Required no. of dot-products $M$) such that each $N$-length row of $F$ has at most $N(P-K+M)/P$ non-zero elements. Because the locations of zeros in a row of $F$ are known by design, this reduces the complexity of computing dot-products of rows of $F$ with $x$. Here $K$ parameterizes the resilience to stragglers: any $K$ of the $P$ dot products of rows of $F$ with $x$ are sufficient to recover $Ax$, i.e., any $K$ rows of $F$ can be linearly combined to generate the rows of $A$.

---

1Strassen’s algorithm [9] and its generalizations offer a recursive approach to faster matrix multiplications over multiple processors, but they are often not preferred because of their high communication cost [10].

2Another interesting example comes from recent work on designing processing units that exclusively compute dot-products using analog components [17,18]. These devices are prone to errors and increased delays in convergence when designed for larger dot products.
2. We provide fundamental limits on the trade-off between the length of the dot-products and the straggler resilience (number of processors to wait for) for any such strategy in Section 3. This suggests a lower bound on the length of task allotted per processor. Our limits show that Short-Dot is near-optimal.

3. Assuming exponential tails of service-times at each server (used in [1]), we derive the expected computation time required by our strategy and compare it to uncoded parallel processing, repetition strategy and MDS coding [15] (see Fig. 2) based linear computation. We also explicitly show a regime \( M = \frac{\log(P)}{\log(P)} \) where Short-Dot outperforms all its competing strategies in expected computation time, by a factor of \( \frac{\log(P)}{\log(P)} \), that diverges to infinity for large \( P \). In general, Short-Dot is found to be universally faster than all its competing strategies over the entire range of \( M \leq P \). When \( M \) is linear in \( P \), Short-Dot offers speed-up by a factor of \( \Omega(\log(P)) \) over uncoded, parallel processing and repetition. When \( M \) is sub-linear in \( P \), Short-Dot out-performs repetition or MDS coding based linear computations by a factor of \( \Omega \left( \frac{P}{M \log(P/M)} \right) \).

4. We also provide experimental results showing that Short-Dot is faster than existing strategies.

In a concurrent work, in [19], Tandon et al. consider a coded computation problem similar to ours for the special case where \( M \), the number of \( N \)-length dot-products to be computed, is 1 and the given matrix \( A_{M \times N} \) (in this case just a single row vector) is \([1, 1, \ldots, 1]_{1 \times N}\). We note that for \( M = 1 \), the gain of using coded strategies over replication-based strategies is bounded even as \( N \) and \( P \to \infty \) for \( s = \Theta(N/P) \). Our paper differs from [19] in that we consider the more general case \( M \geq 1 \), and observe that the gains over replication can be unbounded with this scaling in the regime \( s = \Theta(MN/P) \). For \( M > 1 \), the number of operations per processor using our strategy is lower than an application of [19] for the same worst-case straggler resilience. To see this, note that a straightforward extension of the strategy proposed in [19] that encodes each row of \( A \) separately for \( M(>1) \) rows would require \( M \) dot-products of length \( \frac{N(P-K+1)}{P} \) at each processor while using a “joint” encoding across rows, Short-Dot only requires a single dot-product of length \( \frac{N(P-K+1)}{P} \) (note that \( N(P-K+M)/P < M \times N(P-K+1) \) at each processor, while still requiring the same number of processors (any \( K \) out of \( P \)) to finish. Further, we also provide a tighter converse for \( M > 1 \) that proves that Short-Dot is near-optimal. It is worth noting that [19] also introduces the notion of partial stragglers, which is outside the scope of our paper.

For the rest of the paper, we define the sparsity of a vector \( u \in \mathbb{R}^N \) as the number of nonzero elements in the vector, i.e., \( \|u\|_0 = \sum_{j=1}^{N} I(u_j \neq 0) \). We also assume \( N \) is quite large compared to \( P \), so that it is reasonable to assume that \( P \) divides \( N \) (\( P \ll N \)).

### 1.1 Comparison with existing strategies:

Consider the problem of computing a single dot product of an input vector \( x \in \mathbb{R}^N \) with a prespecified vector \( a \in \mathbb{R}^N \). By an “uncoded” parallel processing strategy (which includes Block Striped Decomposition [7]), 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 processors. E.g. \( a \) can be divided into \( P \) parts – constructing \( P \) short vectors of sparsity \( N/P \) – with each vector stored in a different processor (as shown in Fig. 1 (left)). Only the nonzero values of the vector need to be stored since the locations of the nonzero values is known apriori at every node. One might expect the computation time for each processor to reduce by a factor of \( P \). However, now the fusion node has to wait for all the \( P \) processors 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 \( x \) are sufficient to compute \( \langle a, x \rangle \)? A simple coded strategy is Repetition with block partitioning i.e., constructing \( L \) vectors of sparsity \( N/L \) by partitioning the vector of length \( N \) into \( L \) parts (each \( \leq P \)), and repeating the \( L \) vectors \( P/L \) times so as to obtain \( P \) vectors of sparsity \( N/L \) as shown in Fig. 1 (right). For each of the \( L \) parts of the vector, the fusion node only needs the output of one processor among all its repetitions. Instead of a single dot-product, if one requires the dot-product of \( x \) with \( M \) vectors \( \{a_1, \ldots, a_M\} \), one can simply repeat the aforementioned strategy \( M \) times.
A dot-product of length $N = 12$ is being computed parallely using $P = 6$ processors. (Left) Uncoded Parallel Processing - Divide into $P$ parts, (Right) Repetition with block partitioning.

For multiple dot-products, an alternative repetition-based strategy is to compute $M$ dot products $P/M$ times in parallel at different processors. Now we only have to wait for at least one processor corresponding to each of the $M$ vectors to finish (see Fig. 2b). Improving upon repetition, it is shown in [1] that an $(P,M)$-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). This strategy is shown, both experimentally and theoretically, to perform better than repetition and uncoded strategies.

Can we go beyond MDS codes? MDS codes-based strategies require $N$-length dot-products to be computed on each processor. Short-Dot instead constructs $P$ vectors of sparsity $s$ (less than $N$), such that the dot product of $x$ with any $K \geq M$ out of these $P$ short vectors is sufficient to compute the dot-product of $x$ with all the $M$ given vectors (see Fig. 2d). Compared to MDS Codes, Short-Dot is more flexible as it waits for some more processors (since $K \geq M$), but each processor computes a shorter dot product. Short-Dot also effectively reduces the communication cost since only a shorter portion of the input vector is to be communicated to each processor. We also propose Short-MDS, an
2 Our coded parallelization strategy: Short-Dot

In this section, we provide our strategy of computing the linear transform $Ax$ where $x \in \mathbb{R}^N$ is the input vector and $A_{(M \times N)} = [a_1, a_2, \ldots, a_M]^T$ is a given matrix. Short-Dot constructs a $P \times N$ matrix $F = [f_1, f_2, \ldots, f_P]^T$ such that $M$ predetermined linear combinations of any $K$ rows of $F$ are sufficient to generate each of $\{a_1^T, a_2^T, \ldots, a_M^T\}$, and any row of $F$ has sparsity at most $s = \frac{N}{P} (P - K + M)$. Each sparse row of $F$ (say $f_i^T$) is sent to the $i$-th processor ($i = 1, \ldots, P$) and dot-products of $x$ with all sparse rows are computed in parallel. Let $S_i$ denote the support (set of non-zero indices) of $f_i$. Thus, for any unknown vector $x$, short dot products of length $|S_i| \leq s = \frac{N}{P} (P - K + M)$ are computed on each processor. Since the linear combination of any $K$ rows of $F$ can generate the rows of $A$, i.e., $\{a_1^T, a_2^T, \ldots, a_M^T\}$, the dot-product from the earliest $K$ out of $P$ processors can be linearly combined to obtain the linear transform $Ax$. Before formally stating our algorithm, we first provide an insight into why such a matrix $F$ exists in the following theorem, and develop an intuition on the construction strategy.

`Figure 3: Short-Dot: Distributes short dot-products over $P$ parallel processors, such that outputs from any $K$ out of $P$ processors are sufficient to compute successfully.`

**Theorem 1** Given row vectors $\{a_1^T, a_2^T, \ldots, a_M^T\}$, there exists a $P \times N$ matrix $F$ such that a linear combination of any $K (> M)$ rows of the matrix is sufficient to generate the row vectors and each row of $F$ has sparsity at most $s = \frac{N}{P} (P - K + M)$, provided $P$ divides $N$.

**Proof:** We may append $(K - M)$ rows to $A = [a_1, a_2, \ldots, a_M]^T$, to form a $K \times N$ matrix $\tilde{A} = [a_1, a_2, \ldots, a_M, z_1, \ldots, z_{K-M}]^T$. The precise choice of these additional vectors will be made explicit later. Next, we choose $B$, a $P \times K$ matrix such that any square sub-matrix of $B$ is invertible.\(^3\)

The following lemma shows that any $K$ rows of the matrix $B \tilde{A}$ are sufficient to generate any row of $\tilde{A}$, including $\{a_1^T, a_2^T, \ldots, a_M^T\}$.

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

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

\(^3\)This condition is relaxed in Remark 1.

5
Lemma 2  Given an $M \times N$ matrix $A = [a_1, \ldots, a_M]^T$, let $\tilde{A} = [a_1, \ldots, a_M, z_1, \ldots, z_{K-M}]^T$ be a $K \times N$ matrix formed by appending $K-M$ row vectors to $A$. Also let $B$ be a $P \times K$ matrix such that every square matrix is invertible. Then there exists a choice of the appended vectors $z_1, \ldots, z_{K-M}$ such that each row of $F = B\tilde{A}$ has sparsity at most $s = \frac{N}{P}(P - K + M)$.

Proof: We select a sparsity pattern that we want to enforce on $F$ and then show that there exists a choice of the appended vectors $z_1, \ldots, z_{K-M}$ such that the pattern can be enforced.

Sparsity Pattern enforced on $F$: This is illustrated in Fig. 4. First, we construct a $P \times P$ “unit block” with a cyclic structure of nonzero entries, where $(K - M)$ zeros in each row and column are arranged as shown in Fig. 4. Each row and column have at most $s_c = P - K + M$ non-zero entries. This unit block is replicated horizontally $N/P$ times to form an $P \times N$ matrix with at most $s_c$ non-zero entries in each column, and at most $s = Ns_c/P$ non-zero entries in each row. We now show how choice of $z_1, \ldots, z_{K-M}$ can enforce this pattern on $F$.

From $F = B\tilde{A}$, the $j$-th column of $F$ can be written as, $F_j = B\tilde{A}_j$. Each column of $F$ has at least $K - M$ zeros at locations indexed by $U \subset \{1, 2, \ldots, P\}$. Let $B^U$ denote a $((K-M) \times K)$ sub-matrix of $B$ consisting of the rows of $B$ indexed by $U$. Thus, $B^U\tilde{A}_j = [0]_{(K-M) \times 1}$.

Divide $\tilde{A}_j$ into two portions of lengths $M$ and $K - M$ as follows:

$\tilde{A}_j = [A_j^T \mid z^T]^T = [a_1(j) a_2(j) \ldots a_M(j) \ z_1(j) \ldots z_{K-M}(j)]^T$

Here $A_j = [a_1(j) a_2(j) \ldots a_M(j)]^T$ is actually the $j$-th column of given matrix $A$ and $z = [z_1(j), \ldots, z_{K-M}(j)]^T$ depends on the choice of the appended vectors. Thus,

$$B^U_{\text{cols } 1:M} A_j + B^U_{\text{cols } M+1:K} z = [0]_{K-M \times 1}$$

$$\Rightarrow B^U_{\text{cols } M+1:K} z = -B^U_{\text{cols } 1:M} [A_j]$$

(1)

$$\Rightarrow [z] = -(B^U_{\text{cols } M+1:K})^{-1} B^U_{\text{cols } 1:M} [A_j]$$

(2)

where the last step uses the fact that $[B^U_{\text{cols } M+1:K}]$ is invertible because it is a $(K-M) \times (K-M)$ square sub-matrix of $B$. This explicitly provides the vector $z$ which completes the $j$-th column of $\tilde{A}$. The other columns of $\tilde{A}$ can be completed similarly, proving the lemma.

From Lemmas 1 and 2, for a given $M \times N$ matrix $A$, there always exists a $P \times N$ matrix $F$ such that a linear combination of any $K$ columns of $F$ is sufficient to generate our given vectors and each row of $F$ has sparsity at most $s = \frac{N}{P}(P - K + M)$. This proves the theorem.

Remark 1: Relaxed conditions on matrix $B$

It has been stated in Lemmas 1 and 2 that all square sub-matrices of $B$ 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 $B$ in Lemmas 1 and 2 can be relaxed, as evident from the proof. For matrix $B_{P \times K}$ we only need two conditions. (1) All $K \times K$ square sub-matrices are invertible. (2) All $(K-M) \times (K-M)$ square sub-matrices in the last $K-M$ columns of $B$ 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 computation strategy:

Algorithm 1 Short-Dot

[A] Pre-Processing Step: Encode $F$ (Performed Offline)
Given: $A_{M \times N} = [a_1, \ldots, a_M]^T = [A_1, A_2, \ldots, A_N]$, parameter $K$, Matrix $B_{P \times K}$

1: For $j = 1$ to $N$ do
2:   Set $U \leftarrow ((j - 1), \ldots, (j + K - M - 1) \mod P) + 1$
3:   $\triangleright$ The set of $(K - M)$ indices that are 0 for the $j$-th column of $F$
4:   Set $B^U \leftarrow$ Rows of $B$ indexed by $U$
5:   $\triangleright$ $z_{(K-M)\times1}$ is a row vector.
6:   Set $F_j = B[A_j^T|z^T]^T$, $\triangleright$ $F_j$ is a column vector $(j$-th col of $F)$

Encoded Output: $F_{P \times N} = [f_1, f_2, \ldots, f_P]^T$ $\triangleright$ Row representation of matrix $F$

7: For $i = 1$ to $P$ do
8:   Store $S_i \leftarrow \text{Support}(f_i)$ $\triangleright$ Indices of non-zero entries in the $i$-th row of $F$
9:   Send $f^{S_i}$ to $i$-th processor $\triangleright$ $i$-th row of $F$ sent to $i$-th processor

[B] Online computations
External Input: $x$
Resources: $P$ parallel processors ($P > M$)

[B1] Parallelization Strategy: Divide task among parallel processors:
1: For $i = 1$ to $P$ do
2:   Send $x^{S_i}$ to $i$-th processor
3:   Compute at $i$-th processor: $(f_i^{S_i}, x^{S_i})$ $\triangleright$ $u^S$ denotes only the rows of vector $u$ indexed by $S$

Output: $(f_i^{S_i}, x^{S_i})$ from $K$ earliest processors

[B2] Fusion Node: Decode the dot-products from the processor outputs:
1: Set $V \leftarrow$ Indices of the $K$ processors that finished first
2: Set $B^V \leftarrow$ Rows of $B$ indexed by $V$
3: Set $v_{K \times 1} \leftrightarrow [(f_i^{S_i}, x^{S_i}), \forall i \in V]$ $\triangleright$ Col Vector of outputs from first $K$ processors
4: Set $A x = [(a_1, x), \ldots, (a_M, x)]^T \leftarrow [(B^V)^{-1}]_{rows \ 1:M \ v}$
5: Output: $(x, a_1), \ldots, (x, a_M)$

Table 1: Trade-off between the length of the dot-products and parameter $K$ for different strategies

| Strategy   | Length | Parameter $K$ |
|------------|--------|---------------|
| Repetition | $N$    | $P - \left\lfloor \frac{P}{M} \right\rfloor + 1$ |
| MDS        | $N$    | $M$           |
| Short-Dot  | $s$    | $P - \left\lfloor \frac{P}{N} \right\rfloor + M$ |

| Strategy   | Length | Parameter $K$ |
|------------|--------|---------------|
| Repetition with block partition | $s$    | $P - \left\lfloor \frac{P}{M \times [N/s]} \right\rfloor + 1$ |
| Short-MDS  | $s$    | $P - \left\lfloor \frac{P}{[N/s]} \right\rfloor + M$ |

Remark 2: Short-MDS - a special case of Short-Dot

An extension of the MDS codes-based strategy proposed in [1], that we call Short-MDS can be designed to achieve row-sparsity $s$. First block-partition the matrix of $N$ columns, into $\lceil N/s \rceil$ sub-matrices of size $M \times s$, and also divide the total processors $P$ equally into $\lceil N/s \rceil$ parts. Now, each sub-matrix can be encoded using a $(\frac{P}{\lceil N/s \rceil}, M)$ MDS code. In the worst case, including all integer effects, this strategy requires $K = P - \left\lfloor \frac{P}{M \times [N/s]} \right\rfloor + M$ processors to finish. In comparison, Short-Dot requires $K = P - \left\lfloor \frac{P}{N} \right\rfloor + M$ processors to finish. In the regime where, $s$ exactly divides $N$, Short-MDS can be viewed as a special case of Short-Dot, as both the expressions match. However, in the regime where $s$ does not exactly divide $N$, Short-MDS requires more processors to finish in the worst case than Short-Dot. Short-Dot is a generalized framework that can achieve a wider variety of pre-specified sparsity patterns as required by the application. In Table 1 we compare the lengths of the dot-products and straggler resilience $K$, i.e., the number of processors to wait for in worst case, for different strategies.
3 Limits on trade-off between the length of dot-products and parameter $K$

In this section, we derive fundamental trade-offs between the length of the dot-products computed at each individual processor and the number of processors to wait for, i.e., $K$, which parametrizes the resilience to stragglers. First we derive an information-theoretic limit in Theorem 2 that holds for any matrix $A$, such that each column has at least one non-zero entry. In Theorem 3, we show how this bound can be tightened further, so that in the limit of large number of columns of matrix $A$, Short-Dot is near-optimal.

**Theorem 2** Let $A_{M \times N}$ be any matrix such that each column has at least one non-zero element. For any matrix $F_{P \times N}$ satisfying the property that the span of its any $K$ rows contains the span of the $M$ rows of $A_{M \times N}$, the average sparsity $\bar{s}$ over the rows of $F_{P \times N}$ must satisfy $\bar{s} \geq \frac{N}{P}(P - K + 1)$.

**Proof:** We claim that $K$ is strictly greater than the maximum number of zeros that can occur in any column of the matrix $F$. If not, suppose the $j$-th column of $F$ has more than $K$ zeros. Then there exists a choice of $K$ rows of $F$ 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 $A$ has at least one non-zero entry, say at row $i$, it is not possible to generate the $i$-th row of $A$ by linearly combining these chosen $K$ rows of $F$. Thus,

$$K \geq 1 + \text{Max No. of } 0\text{s in any column of } F$$

$$\geq 1 + \text{Avg. No. of } 0\text{s over columns of } F$$

Here the last line follows since maximum value is always greater than average. Note that if $s$ is the average sparsity over the rows of $F_{P \times N}$, then the average number of zeros over the columns of $F_{P \times N}$ can be written as $\frac{(N-s)P}{N}$. Thus, from (4),

$$K \geq 1 + \frac{(N-s)P}{N}. \quad (5)$$

A slight re-arrangement establishes the lower bound in Theorem 2.

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 $\bar{s} \geq \frac{N}{P}(P - K + 1)$ is also tight only for $M = 1$ (implicitly assuming $P$ divides $N$, since $P \ll N$). Now we tighten this bound further for $M > 1$.

### 3.1 Tighter Fundamental Bounds

**Theorem 3** Let $M > 1$. Then there exists a matrix $A_{M \times N}$, such that any $F_{P \times N}$ satisfying the property that any $K$ rows of $F_{P \times N}$ can span all the rows of $A_{M \times N}$, must also satisfy the following property:

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

$$\bar{s} > \frac{N}{P}(P - K + M) - \frac{M^2}{P}(\frac{P}{K-M+1}) \quad (6)$$

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

$$\bar{s} \geq \frac{N}{P}(P - K + M) - o\left(\frac{N}{P}\right) \quad (7)$$

Note that the second term in the lower bound in (6) 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.

---

\(^4\)Note that choice of such a class of matrix $A$ is reasonable, since if say the $j$-th column of $A$ consists entirely of zeros, then the $j$-th column and its corresponding entry in unknown vector $x$ can simply be omitted from the problem.
Before proceeding with the proof, we give a basic intuition on the proof technique. We basically divide the columns of $F_{P \times N}$ 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 $A_{M \times N}$ 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:** Let us denote the number of columns of $F_{P \times N}$ with more than $(K - M)$ zeros as $\lambda$. We will show later in Lemma 3 that $\lambda < M \binom{P}{K - M + 1}$. Now, compute the average number of zeros over the columns of $F$. The columns of $F$ can be divided into two groups: $\lambda$ columns with greater than $(K - M)$ zeros and $(N - \lambda)$ columns with at-most $(K - M)$ zeros. Recall from (3), that if $A$ is chosen such that every column has at-least one non-zero entry, then the maximum number of zeros in any column of $F$ is upper bounded by $(K - 1)$. Thus, the group of $\lambda$ columns can have at-most $K - 1$ zeros each. Thus,

$$\text{Avg. no. of 0s per column} \leq \frac{(K - 1)\lambda + (K - M)(N - \lambda)}{N} = K - M + \lambda \frac{(M - 1)}{N} \text{ Lemma 3} < K - M + \frac{M^2 \binom{P}{K - M + 1}}{N} \quad (8)$$

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

$$\frac{(N - \bar{s})P}{N} < (K - M) + \frac{M^2 \binom{P}{K - M + 1}}{N} \quad (9)$$

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

$$\bar{s} > \frac{N}{P} (P - K + M) - \frac{M^2}{P} \binom{P}{K - M + 1} \quad (10)$$

Thus, the first part of the theorem, i.e., (10) is proved. Using the condition that $M^2 \binom{P}{K - M + 1} = o(N)$ in (9), we can also obtain (7). Thus,

$$s > \frac{N}{P} (P - K + M) - o \left( \frac{N}{P} \right) \quad (11)$$

Thus, the theorem is proved. \hfill \blacksquare

Now it only remains to prove Lemma 3.

**Lemma 3:** Let $M > 1$. Then there exists a matrix $A_{M \times N}$, such that any $F_{P \times N}$ satisfying the property that any $K$ rows of $F_{P \times N}$ can span all the rows of $A_{M \times N}$, must also satisfy the following property: The number of columns ($\lambda$) with more than $K - M$ zeros is upper bounded as $\lambda < M \binom{P}{K - M + 1}$.

**Proof:** 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 $F$, of size $P \times \frac{\lambda}{\binom{P}{K - M + 1}}$, consisting of only the columns of $F$ having $(K - M + 1)$ zeros in the same locations, i.e., with similar sparsity pattern. Any $K$ rows of this sub-matrix of $F$ should generate all the rows of a corresponding $M \times \frac{\lambda}{\binom{P}{K - M + 1}}$ sub-matrix of the given $A$, consisting of the same columns of $A$ as picked in this sub-matrix of $F$.

There always exists a fully dense matrix $A$ such any $M \times \frac{\lambda}{\binom{P}{K - M + 1}}$ sub-matrix of $A$ is full-rank, since $A$ can be arbitrary. This sub-matrix of $A$ is of rank $\min \{ M, \frac{\lambda}{\binom{P}{K - M + 1}} \} = M$ (from assumption). Any $K$ rows of the sub-matrix of $F$ should generate $M$ linearly independent rows of this sub-matrix of $A$. But since the sub-matrix of $F$ 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 $A$. This is a contradiction. Thus, we must have $\lambda > M \binom{P}{K - M + 1}$. \hfill \blacksquare
4 Analysis of expected computation time for exponential tail models

We now provide a probabilistic analysis of the computation time required by Short-Dot and compare it with uncoded parallel processing, repetition and MDS coding based linear computation scheme as shown in Fig. 5. We follow the shifted-exponential computation time model as described in [1]. 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 [1]. We assume that the time required by a processor to compute a single dot-product of length $N$ be distributed as:

$$
\Pr(T_N \leq t) = \begin{cases} 
1 - \exp\left(-\mu \left(\frac{t}{N} - 1\right)\right), & \forall \ t \geq N \\
0, & \text{otherwise}
\end{cases}
$$

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 processor 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 (12), as suggested in [1]. 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.

The expected computation time for Short-Dot is the expected value of the $K$-th order statistic of these $P$ iid exponential random variables, which is given by:

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

Here, (13) uses the fact that the expected value of the $K$-th statistic of $P$ iid exponential random variables with parameter 1 is $\sum_{i=1}^{P-K} \frac{1}{i} - \sum_{i=1}^{P-K} \frac{1}{i} \approx \log(P) - \log(P - K)$ [1]. The expected computation time in the RHS of (13) is minimized when $P - K = \Theta(M)$. This minimal expected time is $O\left(\frac{MN}{P}\right)$ for $M$ linear in $P$ and is $O\left(\frac{MN \log(P/M)}{P}\right)$ for $M$ sub-linear in $P$.

Figure 5: Expected computation time: Short-Dot 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, Repetition is slowest. When $M$ does not exactly divide $P$, the distribution of computation time for repetition and uncoded strategies is the maximum of non-identical but independent random variables, which produce the ripples in these curves (see Appendix for details).

A detailed analysis of the expected computation time for the competing strategies, i.e., uncoded strategy, repetition and MDS coding strategy is provided in the Appendix. Table 2 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...
Table 2: Probabilistic Computation Times

| Strategy          | Expected Time                                      | $M$ linear in $P$ | $M$ sub-linear in $P$ |
|-------------------|----------------------------------------------------|-------------------|------------------------|
| Only one Processor | $MN \left(1 + \frac{1}{\mu} \right)$              | $\Theta(MN)$      | $\Theta(MN)$           |
| Uncoded (M divides $P$) | $\frac{MN}{P} \left(1 + \log \frac{P}{\mu} \right)$ | $\Theta \left(\frac{MN}{P} \log(P)\right)$ | $\Theta \left(\frac{MN}{P} \log(P)\right)$ |
| Repetition (M divides $P$) | $N \left(1 + \frac{M \log(M)}{P \mu} \right)$ | $\Theta \left(\frac{MN}{P} \log(P)\right)$ | $\Theta(N)$            |
| MDS               | $N \left(1 + \log \frac{P}{P - M} \right)$        | $\Theta(N)$       | $\Theta(N)$            |
| Short-Dot         | $\frac{N(P - K + M)}{P} \left(1 + \frac{\log \left(\frac{P}{P - M} \right)}{\mu} \right)$ | $O \left(\frac{MN}{P} \log \left(\frac{P}{M} \right)\right)$ | $O \left(\frac{MN}{P} \log \left(\frac{P}{M} \right)\right)$ |

2 Refer to Appendix for more accurate analysis taking integer effects into account.

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 $M$.

Now we explicitly provide a regime, where the speed-ups from Short-Dot diverges to infinity for large $P$, in comparison to all three competing strategies - MDS Coding, Repetition or Uncoded strategies.

**Theorem 4** Suppose $M$ scales as $\frac{P}{\log P}$. Then, Short-Dot with $K = P - \frac{M}{2}$ has an expected computation time (scaled by $N$) as $E \left[ \frac{T_{SD}}{N} \right] = O \left( \log \left( \log \frac{P}{\mu} \right) \right)$ that decays to 0 as $P \to \infty$. In contrast, the expected computation time (scaled by $N$) for MDS coding, repetition and uncoded strategies scale as $\Omega(1)$ and thus do not decay to 0 as $P \to \infty$.

**Proof:** 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 let $M = \frac{P}{\log P}$ for all the strategies. For uncoded strategy, we thus obtain,

$$E \left[ \frac{T_{UC}}{N} \right] = \frac{M}{P} \left(1 + \frac{\log(P)}{\mu} \right) = \frac{1}{\log P} \left(1 + \frac{\log(P)}{\mu} \right) \geq \frac{1}{\mu} = \Omega(1) \quad (14)$$

For repetition, we obtain,

$$E \left[ \frac{T_{Rep}}{N} \right] = \left(1 + \frac{M \log(M)}{P \mu} \right) \geq 1 = \Omega(1) \quad (15)$$

For MDS Coding based linear computation, we obtain,

$$E \left[ \frac{T_{MDS}}{N} \right] = 1 + \frac{\log \left(\frac{P}{P - M} \right)}{\mu} = 1 + \frac{\log \left(\frac{\log P}{\log P - 1} \right)}{\mu} \geq 1 = \Omega(1) \quad (16)$$

Now, we consider the Short-Dot strategy with $K = P - \frac{M}{2} = P - \frac{P}{2 \log P}$. Note that the inequality $K > M$ is satisfied for $\log P > \frac{3}{2}$. Now let us calculate the expected computation time for Short-Dot.
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. [6]

5 Encoding and Decoding Complexity

5.1 Encoding Complexity:

Even though encoding is a pre-processing step (since $A$ is assumed to be given in advance), we include a complexity analysis for the sake of completeness. Recall from Section 2 that we first choose an appropriate matrix $B$ 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 $A$, we perform the following.

- Set $U \leftarrow \{(j - 1), \ldots, (j + K - M - 1) \mod P\} + 1$
- $B^U \leftarrow$ Rows of $B$ indexed by $U$
- Solve for $z$ : $(B^U_{cols,M+1:M})[z] = -B^U_{cols\ 1:M}[A_j]$
- Set $F_j = B[A_j^T \{z^T\}]^T$

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

This is higher than MDS coding based linear computation that has an encoding complexity of $O(NMP)$, but it is only a one-time cost that provides savings in online steps (as discussed earlier in this section).

5.1.1 Reduced Complexity using Vandermonde matrices:

The encoding complexity can be reduced further for special choices of the matrix $B$. Let us choose $B$ to be a Vandermonde matrix as given by

$$B = \begin{bmatrix} h_1^{K-1} & \cdots & h_1 & 1 \\ h_2^{K-1} & \cdots & h_2 & 1 \\ \vdots & \ddots & \vdots & \vdots \\ h_p^{K-1} & \cdots & h_p & 1 \end{bmatrix} \quad (18)$$

Here, $h_1, h_2, \ldots, h_K \in \mathbb{R}$, and are all distinct. This matrix $B$ satisfies all the requirements of the encoding matrix. All $K \times K$ sub-matrices of $B$ 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 $F$. For each of the $N$ columns of $F$, the encoding requires solving a linear system of equations for $z$, as given by:

$$(B^U_{cols\ M+1:M})[z] = -B^U_{cols\ 1:M}[A_j] \quad (19)$$

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

\[
F_j = B[A_j^T | z^T]^T
\]  

(20)

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 \(A\) into a bunch of polynomial evaluation and interpolation problems, all of degree less than \(P\). Now, from [20], [21], we know that both the interpolation and the evaluation of a polynomial of degree less than \(P\), at \(P\) arbitrary points is \(O(P \log^2(P))\). Thus, the complexity of encoding is \(O(NP \log^2(P))\).

5.2 Decoding Complexity:

During decoding, we get \(K\) dot-products from the first \(K\) processors out of \(P\). We then perform the following operations.

Set \(V \leftarrow \) Indices of the processors that finished first
Set \(B^V \leftarrow \) Rows of \(B\) indexed by \(V\)
Set \(u_{K \times 1} \leftarrow \{(f_i^{S_i}, x^{S_i}), \forall i \in V\} \quad \triangleright \) Col Vector of outputs from first \(K\) processors
Solve for \(w_{K \times 1} : \quad [(B^V)]w = v\)
Output: \(Ax = [w_1, w_2, \ldots, w_M]^T\) \quad \triangleright \) First \(M\) values of \(w\)

We solve a system of \(K\) linear equations 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 \(O(K^3 + KM)\) which does not depend on \(N\) when \(M, K \ll N\). This is nearly the same as \(O(M^3 + M^2)\) complexity of MDS coding based linear computation.

5.2.1 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 \(B\) as a Vandermonde matrix as described in (13). The decoding problem consists of solving a system of \(K\) linear equations in \(K\) variables.

\[
[(B^V)]w = v
\]  

(21)

Here \(V\) is a set of \(K\) indices \(\in \{1, 2, \ldots, P\}\). The problem of finding \(w\) is equivalent to the interpolation of the co-efficients of a polynomial of degree \((K - 1)\), from its values at \(K\) arbitrary points given by \(\{h_l | l \in V\}\). Again, from [20], [21], the interpolation of a polynomial of degree \((K - 1)\), at \(K\) arbitrary points can be done in \(O(K \log^2(K))\), which thus becomes the decoding complexity.

6 Experimental Results

We perform experiments on computing clusters at CMU to test the computational time. We use HTCondor [22] to schedule jobs simultaneously among the \(P\) processors. We compare the time required to classify 10000 handwritten digits of the MNIST [23] database, assuming we are given a trained 1-layer Neural Network. We separately trained the Neural network using training samples, to form a matrix of weights, denoted by \(A_{10 \times 785}\). For testing, the multiplication of this given \(10 \times 785\) matrix, with the test data matrix \(X_{785 \times 10000}\) is considered. The total number of processors was 20.

Assuming that \(A_{10 \times 785}\) is encoded into \(F_{20 \times 785}\) in a pre-processing step, we store the rows of \(F\) in each processor apriori. Now portions of the data matrix \(X\) of size \(s \times 10000\) are sent to each of the \(P\) parallel processors as input. We also send a C-program to compute dot-products of length \(s = \frac{1}{P}(P - K + M)\) with appropriate rows of \(F\) using command \texttt{condor-submit}. Each processor outputs the value of one dot-product. The computation time reported in Fig. 7 includes the total time required to communicate inputs to each processor, compute the dot-products in parallel, fetch the required outputs, decode and classify all the 10000 test-images, based on 35 experimental runs.

**Key Observations:** (See Table 3 for detailed results). Computation time varies based on nature of straggling, at the particular instant of the experimental run. Short-Dot outperforms both MDS and Uncoded, in mean computation time. Uncoded is faster than MDS since per-processor computation

\[13\]
Figure 7: Experimental results: (Left) Mean computation time for Uncoded Strategy, Short-Dot (K=18) and MDS codes: Short-Dot is faster than MDS by 32% and Uncoded by 12%. (Right) Scatter plot of computation time for different experimental runs: Short-Dot is faster most of the time.

| Strategy  | Parameter K | Mean  | STDEV | Minimum Time | Maximum Time |
|-----------|-------------|-------|-------|--------------|--------------|
| Uncoded   | 20          | 11.8653 | 2.8427 | 9.5192       | 27.0818      |
| Short-Dot | 18          | 10.4306 | 0.9253 | 8.2145       | 11.8340      |
| MDS       | 10          | 15.3411 | 0.8987 | 13.8232      | 17.5416      |

time for MDS is larger, and it increases the straggling, even though MDS waits for only for 10 out of 20 processors. However, note that Uncoded has more variability than both MDS and Short-Dot, and its maximum time observed during the experiment is much greater than both MDS and Short-Dot. The classification accuracy was 85.98% on test data.

**Comment:** The experimental times are quite high due to some limitations of the experimental platform used. The time includes some overhead to start the cluster, and communicate data in the form of text files to all the processors, and also collect the output data files back from all the processors. The read time also depends on the size of file to be read. Currently, we are looking at performing these experiments in alternate distributed computing platforms, with better communication protocols.

## 7 Discussion

### 7.1 Storage and Communication benefits of Shorter Dot Products:

The major advantage of using Short-Dot codes over the MDS coding strategy in [1] is that the length of the pre-stored vectors (rows of $F$) as well as the communicated input (portions of $x$) is shorter than $N$. It is thus applicable when processing units have limitations of memory, and it is not possible to pre-store the long vectors of length $N$. Short-dot also has advantages over [1] in systems where the principle bottlenecks in computation time is in communicating the input $x$ to all the processors, and it may not be feasible to broadcast (multicast) $x$ to all processors at the same time. Thus, it is also useful in applications where communication costs are predominant over computation costs.

### 7.2 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 processors return entirely faulty or garbage outputs, in a distributed system and we do not know which of the outputs are erroneous. We argue from coding theoretic arguments that Short-Dot codes designed to tolerate $(P-K)$ stragglers, can also correct $\left\lfloor \frac{(P-K)}{2} \right\rfloor$ errors. First observe that if the code can tolerate $(P-K)$ stragglers, 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 $\left\lfloor \frac{\text{Hamming Distance} - 1}{2} \right\rfloor$ which is $\left\lfloor \frac{(P-K)}{2} \right\rfloor$. The same result can also be derived by recasting the decoding problem as a sparse reconstruction problem, and borrowing ideas from standard compressive sensing literature [24] which also yields a concrete, decoding algorithm. 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 $B$.

### 7.3 More dot-products than processors

While we have presented the case of $M < P$ here, Short-Dot easily generalizes to the case where $M \geq P$. The matrix can be divided horizontally into several chunks along the row dimension (shorter matrices) and Short-Dot can be applied on each of those chunks one after another. Moreover if rows with same sparsity pattern are grouped together and stored in the same processor initially, then the communication cost is also significantly reduced during the online computations, since only some elements of the unknown vector $\mathbf{x}$ are sent to a particular processor.

**Acknowledgments:** Systems on Nanoscale Information fabriCs (SONIC), one of the six SRC STARnet Centers, sponsored by MARCO and DARPA. We also acknowledge NSF Awards 1350314, 1464336 and 1553248. S Dutta also received Prabhu and Poonam Goel Graduate Fellowship.

**References**

[1] Kangwook Lee, Maximilian Lam, Ramtin Pedarsani, Dimitris Papailiopoulos, and Kannan Ramchandran. Speeding Up Distributed Machine Learning Using Codes. *NIPS Workshop on Learning Systems*, 2015.

[2] Da Wang, Gauri Joshi, and Gregory Wornell. Using straggler replication to reduce latency in large-scale parallel computing. *ACM SIGMETRICS Performance Evaluation Review*, 43(3):7–11, 2015.

[3] Da Wang, Gauri Joshi, and Gregory Wornell. Efficient Task Replication for Fast Response Times in Parallel Computation. *ACM SIGMETRICS Performance Evaluation Review*, 42(1):599–600, 2014.

[4] Gauri Joshi, Yanpei Liu, and Emina Soljanin. On the delay-storage trade-off in content download from coded distributed storage systems. *IEEE Journal on Selected Areas in Communications*, 32(5):989–997, 2014.

[5] Longbo Huang, Sameer Pawar, Hao Zhang, and Kannan Ramchandran. Codes can reduce queuing delay in data centers. In *Proceedings IEEE International Symposium on Information Theory (ISIT)*, pages 2766–2770, 2012.

[6] William Dally. High-performance hardware for machine learning. *NIPS Tutorial*, 2015.

[7] Vipin Kumar, Ananth Grama, Gupta Anshul, and George Karypis. *Introduction to Parallel Computing: Design and Analysis of Algorithms*. The Benjamin/Cummings Publishing Company, Inc., Redwood City, 1994.

[8] Geoffrey C Fox, Steve W Otto, and Anthony JG Hey. Matrix algorithms on a hypercube I: Matrix multiplication. *Parallel computing*, 4(1):17–31, 1987.

[9] Volker Strassen. Gaussian elimination is not optimal. *Numerische Mathematik*, 13(4):354–356, 1969.

[10] Grey Ballard, James Demmel, Olga Holtz, and Oded Schwartz. Communication costs of strassen’s matrix multiplication. *Communications of the ACM*, 57(2):107–114, 2014.

[11] Jeffrey Dean and Luiz André Barroso. The tail at scale. *Communications of the ACM*, 56(2):74–80, 2013.

[12] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. The Hadoop Distributed File System. In *Proceedings IEEE Symposium on Mass Storage Systems and Technologies (MSST)*, pages 1–10, 2010.

[13] Kuang-Hua Huang and Jacob A. Abraham. Algorithm-based fault tolerance for matrix operations. *IEEE transactions on computers*, 100(6):518–528, 1984.

[14] Thomas Herault and Yves Robert. *Fault-Tolerance Techniques for High Performance Computing*. Springer, 2015.

[15] William Ryan and Shu Lin. *Channel codes: Classical and Modern*. Cambridge University Press, 2009.
8 Appendix

We now provide a probabilistic analysis of the computational time required by Short-Dot and compare it with uncoded parallel processing, repetition and MDS code based linear computation as shown in Fig. 5. We assume that the time required by a processor to compute a single dot-product follows an exponential distribution and is independent of other parallel processors.

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

$$
\Pr(T_N \leq t) = F(t) = \begin{cases} 
1 - \exp\left(-\mu \left(\frac{t}{N} - 1\right)\right), & \forall t \geq N \\
0, & \text{otherwise} 
\end{cases}
$$

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:

$$
\Pr(T_s \leq t) = F\left(\frac{Nt}{s}\right) = \begin{cases} 
1 - \exp\left(-\mu \left(\frac{t}{s} - 1\right)\right), & \forall t \geq s \\
0, & \text{otherwise} 
\end{cases}
$$

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

8.1 Proposed Strategy – Short-Dot:

The computation time over each of the $P$ processors behaves as iid exponential random variables following the distribution:

$$
\Pr(T_s \leq t) = F\left(\frac{Nt}{s}\right) = 1 - \exp\left(-\mu \left(\frac{t}{s} - 1\right)\right) \quad \forall t \geq s .
$$

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:

$$
\mathbb{E}[T_{SD}] = s \left(1 + \frac{\log(P - K)}{\mu}\right) = \frac{(P - K + M)N}{P} \left(1 + \frac{\log(P - K)}{\mu}\right)
$$
Here we use the result \(\text{from } [\Pi]\) that the \(K\)-th order statistic of \(P\) exponential random variables that are \(iid\) as \(\sim \exp (-T) \forall T \geq 0\) 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:
\[
\sum_{i=1}^{P} \frac{1}{i} - \sum_{i=1}^{P-K} \frac{1}{i} \approx \log(P) - \log(P - K)
\] (26)

Note that the expected computation time is minimized when \(K = P - \Theta(M)\), and is given by:-
\[
\mathbb{E}[T_{SD}] = O \left( \frac{MN}{P} \left( 1 + \frac{\log(P/M)}{P} \right) \right)
\] (27)

If \(M = \Theta(P)\), the expected time is \(O\left(\frac{MN}{P}\right)\). If \(M = o(P)\), the expected time is \(O\left(\frac{MN \log(P/M)}{P}\right)\).

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 (27) is an upper bound for the actual expected computation time. Thus we use \(O(.)\) instead of \(\Theta(.)\).

8.2 Existing Strategies

**One Single Processor:** For one single processor to compute all \(M\) dot-products of length \(N\), the computation time is distributed as
\[
\Pr(T_{NM} \leq t) = F(t/M) = 1 - \exp \left( -\mu \left( \frac{t}{NM} - 1 \right) \right) \forall t \geq NM
\] (28)

Thus, the expected computation time can be easily derived to be
\[
\mathbb{E}[T_{1P}] = MN \left( 1 + \frac{1}{\mu} \right)
\] (29)

**Uncoded - Divide into \(P\) parts and wait for all:** Now, consider an uncoded strategy where the computation is simply divided into \(P\) dot-products and sent to \(P\) processors. We assume that each processor is sent only one dot-product at a time. We wait for all the processors to finish computation. Note that integer effects arise when \(M\) does not exactly divide \(P\). Some rows can be divided among \(\left\lfloor \frac{P}{M} \right\rfloor\) processors, while the remaining are divided among \(\left\lfloor \frac{P}{M} \right\rfloor\) processors. Let \(m_1\) and \(m_2\) denote the number of rows that get \(\left\lfloor \frac{P}{M} \right\rfloor\) processors and \(\left\lfloor \frac{P}{M} \right\rfloor\) processors respectively. Clearly the values can be obtained by solving:
\[
\begin{bmatrix}
\frac{1}{\lfloor P/M \rfloor} & \frac{1}{\lfloor P/M \rfloor} \\
\end{bmatrix}
\begin{bmatrix}
m_1 \\
m_2 \\
\end{bmatrix} = \begin{bmatrix} M \end{bmatrix}
\] (30)

Now, we have two groups of exponential variables: one group consisting of \(m_1 \left\lfloor \frac{P}{M} \right\rfloor\) \(iid\) exponential random variables of task size \(\frac{N}{\lfloor M \rfloor}\), and another group consisting of \(m_2 \left\lfloor \frac{P}{M} \right\rfloor\) \(iid\) exponential random variables of task size \(\frac{N}{\lfloor M \rfloor}\). The two groups are independent of each other. Note that we assume that \(N\) is large compared to \(P\) and is divisible by \(P\), \(\left\lfloor \frac{P}{M} \right\rfloor\), \(\left\lfloor \frac{P}{M} \right\rfloor\), 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 \(P = m_1 \left\lfloor \frac{P}{M} \right\rfloor + m_2 \left\lfloor \frac{P}{M} \right\rfloor\) exponential random variables.
\[
\Pr(T_{UC} \leq t) = \left( 1 - \exp \left( -\mu \left( \frac{\left\lfloor \frac{P}{M} \right\rfloor t}{N} - 1 \right) \right) \right)^{m_1 \left\lfloor \frac{P}{M} \right\rfloor} \times \left( 1 - \exp \left( -\mu \left( \frac{\left\lfloor \frac{P}{M} \right\rfloor t}{N} - 1 \right) \right) \right)^{m_2 \left\lfloor \frac{P}{M} \right\rfloor} \forall t \geq \frac{N}{\lfloor P/M \rfloor}
\] (31)
The expectation is thus obtained as

$$E[T_{UC}] = \int_{0}^{\infty} (1 - \Pr(T_{UC} \leq t)) \, dt$$  \hspace{1cm} (32)$$

This expression is numerically computed using MATLAB and plotted in the plot of theoretical computation time in Fig. 5. When \( M \) divides \( P \) exactly, the expressions are simpler. The computation time for each processor is distributed as

$$\Pr(T_{UC} \leq t) = F \left( \frac{t}{M} \right) = 1 - \exp \left( -\mu \left( \frac{Pt}{MN} - 1 \right) \right) \quad \forall \, t \geq NM/P$$  \hspace{1cm} (33)$$

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

$$E[T_{UC}] = MN \left( 1 + \frac{\log(P)}{\mu} \right)$$  \hspace{1cm} (34)$$

The expected time for this uncoded strategy is \( \Theta \left( MN \log \left( \frac{P}{\mu} \right) \right) \) regardless of whether \( M \) is linear or sub-linear in \( P \). Our strategy Short-Dot thus offers a speed-up of \( \Omega \left( \log \left( \frac{P}{\mu} \right) \right) \) in expected computation time when \( M \) is linear in \( P \), as mentioned in (27), and thus outperforms by a factor that diverges to infinity for large \( P \).

**Repetition:** When a \((P, M)\) repetition 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 \( \left\lceil \frac{P}{M} \right\rceil \) times, while the remaining are repeated \( \left\lfloor \frac{P}{M} \right\rfloor \) times. Let \( m_1 \) and \( m_2 \) denote the number of rows that are repeated \( \left\lceil \frac{P}{M} \right\rceil \) times and \( \left\lfloor \frac{P}{M} \right\rfloor \) times respectively. Clearly the values can be obtained by solving:

$$\begin{bmatrix} 1 & \left\lfloor \frac{P}{M} \right\rfloor \\ \left\lceil \frac{P}{M} \right\rceil & \left\lfloor \frac{P}{M} \right\rfloor \end{bmatrix} \begin{bmatrix} m_1 \\ m_2 \end{bmatrix} = \begin{bmatrix} M \\ P \end{bmatrix}$$  \hspace{1cm} (35)$$

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

$$\Pr(T_{Rep} \leq t) = \left( 1 - \exp \left( -\mu \left( \frac{P}{M} \right) \left( \frac{t}{N} - 1 \right) \right) \right)^{m_1} \times \left( 1 - \exp \left( -\mu \left( \frac{P}{M} \right) \left( \frac{t}{N} - 1 \right) \right) \right)^{m_2} \quad \forall \, t \geq N$$  \hspace{1cm} (36)$$

The expectation is thus obtained as

$$E[T_{Rep}] = \int_{0}^{\infty} (1 - \Pr(T_{Rep} \leq t)) \, dt$$  \hspace{1cm} (37)$$

This expression is computed using MATLAB in the plot of theoretical expected computation time (Fig. 5). When \( M \) exactly divides \( P \), the analysis is simpler, and the two types of exponential distributions are identical. Following an analysis similar to [1], it simplifies to the expectation of the maximum of \( M \) \( \text{iid} \) exponential random variables, each of which is the minimum of \( P/M \) \( \text{iid} \) exponential random variables.

$$E[T_{Rep}] = N \left( 1 + \frac{M \log(M)}{P\mu} \right)$$  \hspace{1cm} (38)$$

When \( M \) is linear in \( P \), the expected computation time is \( \Theta \left( \frac{MN \log(P)}{P} \right) \) 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.
Figure 8: Theoretical Plot of expected computation time of repetition taking integer effects into account: straggling parameter $\mu = 5$, total processors $P = 1000$ and number of dot-products $M$ is varied from 1 to $P$.

**MDS codes-based strategy:** The matrix is separated into $M$ rows and coded into $P$ rows using a $(P, M)$ MDS code. Thus, each processor effectively computes a dot-product of length $N$. We have to wait for any $M$ processors to finish. Assuming the computation of each processor is independent, following an analysis similar to [1], we obtain that,

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

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