Rateless Codes for Near-Perfect Load Balancing in Distributed Matrix-Vector Multiplication

Ankur Mallick
ECE Department
Carnegie Mellon University
Pittsburgh, PA 15213, USA
Email: amallic1@andrew.cmu.edu

Malhar Chaudhari
ECE Department
Carnegie Mellon University
Pittsburgh, PA 15213, USA
Email: mschaudh@andrew.cmu.edu

Gauri Joshi
ECE Department
Carnegie Mellon University
Pittsburgh, PA 15213, USA
Email: gaurij@andrew.cmu.edu

***Abstract*—Large-scale machine learning and data mining applications require computer systems to perform massive computations that need to be parallelized across multiple nodes, for example, massive matrix-vector and matrix-matrix multiplication. The presence of straggling nodes – computing nodes that unpredictably slowdown or fail – is a major bottleneck in such distributed computations. We propose a *rateless fountain coding* strategy to alleviate the problem of stragglers in distributed matrix-vector multiplication. Our algorithm creates a stream of linear combinations of the m rows of the matrix, and assigns them to different worker nodes, which then perform row-vector products with the encoded rows. The original matrix-vector product can be decoded as soon as slightly more than m row-vector products are collectively finished by the nodes. This strategy enables fast nodes to steal work from slow nodes, without requiring the master to perform any dynamic load-balancing. Compared to recently proposed fixed-rate erasure coding strategies which ignore partial work done by straggling nodes, rateless coding achieves significantly lower overall delay, as well as small computational and decoding overhead.**

I. Introduction

A. Motivation for Coded Matrix-vector multiplication

Matrix vector multiplications form the core of a plethora of scientific computing and machine learning applications that include solving partial differential equations [1], forward and back propagation in neural networks [2], computing the PageRank of graphs [3] etc. In the present age of Big Data, most of these applications involve multiplying extremely large matrices and vectors and the computations cannot be performed efficiently on a single machine. This has motivated the development of several algorithms [4], [5] that seek to speed up matrix vector multiplication by computing it in a distributed fashion across multiple computation nodes. These algorithms achieve speedup by distributing the computation equally among all the nodes in the system. The individual nodes (the *workers*) perform their respective tasks in parallel while a central node (the *master*) aggregates the output of all the processors to complete the computation. Unfortunately, such approaches are usually bottlenecked by the presence of a few slow workers, called stragglers [6], that cause the entire computation to be delayed since the master needs to wait for all workers to complete the tasks assigned to them.

A natural approach to countering stragglers involves replicating individual tasks at multiple worker nodes [7]–[10], and waiting for any one copy to finish. Replication is a special case of the more general erasure coding framework wherein stragglers are modeled as *erasures* and codes are employed to add redundancy so that only a subset of processors are required to complete the tasks assigned to them. This is analogous to erasure codes for communication channels which can be used to reconstruct the original message from a subset of the transmitted bits.

The usage of codes to provide error-resilience in computation has its origins in works on algorithmic fault tolerance [11]. Recent works such as [12]–[14] have employed Maximum Distance Separable (MDS) codes to speed up the computation of matrix vector products in a distributed setting. To illustrate the key idea of MDS coding, consider the example of multiplying a matrix $\mathbf{A}$ with vector $\mathbf{x}$ using 3 worker nodes and a $(3, 2)$ MDS code. Suppose we split $\mathbf{A}$ along rows into two matrices $\mathbf{A}_1$ and $\mathbf{A}_2$ such that $\mathbf{A} = [\mathbf{A}_1^T \ \mathbf{A}_2^T]^T$. The worker nodes store matrices $\mathbf{A}_1$, $\mathbf{A}_2$ and $\mathbf{A}_1+\mathbf{A}_2$ respectively, and each node multiplies its matrix with $\mathbf{x}$. Results from any two worker nodes are sufficient to obtain $\mathbf{Ax}$, and thus the system is tolerant to 1 straggling node. These codes reduce the overall latency, but they discard the partial work done by straggling nodes, possibly resulting in a large amount of redundant computation.

B. Benefits of using Rateless Codes

We propose the use of rateless fountain codes [15]–[18] for distributed matrix-vector multiplication of a $m \times n$ matrix $\mathbf{A}$ with a $n \times 1$ vector $\mathbf{x}$. The rateless coded matrix-vector multiplication algorithm generates coded linear combinations of the m rows of matrix $\mathbf{A}$ and distributes them across p worker nodes. Each node also receives a copy of the vector $\mathbf{x}$. The master node needs to wait for any $m' = m(1 + \epsilon)$ row-vector products to be completed in total by the nodes, where ϵ is a small decoding overhead such that $\epsilon \to 0$ as $m \to \infty$. Rateless codes offer the following key benefits over previously proposed coding techniques based on MDS codes.

1) Near-perfect Load Balancing: In order to adjust to varying speeds of worker nodes and minimize the overall time to complete the multiplication $\mathbf{Ax}$, one can use a perfect load-balancing scheme that dynamically assigns one row-vector product computation task to each worker node, as soon as the node finishes its current task. Thus, faster nodes complete more tasks than slower nodes, and the p nodes collectively finish m row-vector products. Our rateless coding strategy achieves nearly the same load balancing benefit without the communication overhead of dynamically allocating the tasks, one row-vector product at a time. In our strategy, the nodes need to collectively finish $m' = m(1+\epsilon)$ row-vector products, for small ϵ. In contrast, MDS coding strategies do not adjust to different degrees of node slowdown; they use the results from k nodes, and ignore the remaining $p-k$ nodes. As a result rateless code achieve a much lower delay than MDS coding strategies.

2) Negligible Redundant Computation: A major drawback of MDS coding is that if there is no straggling, the worker nodes collectively perform mp/k row-vector products, instead of m. Our strategy performs a maximum of $m' = m(1+\epsilon)$, where, the overhead ϵ goes to zero as m, the number of rows in the matrix $\mathbf{A}$ increases.

3) Maximum straggler tolerance: A (p, k) MDS coded distributed computation is robust to $p-k$ straggling nodes, for $k \in [1, 2, \ldots p]$. Reducing k increases straggler tolerance but also adds more redundant computation. Rateless coding can tolerate to up to $p-1$ stragglers, with negligible redundant computation overhead.

4) Low Decoding Complexity: Rateless codes offer a low decoding complexity: $O(m \log m)$ for LT codes [15], and $O(m)$ for Raptor codes [16]. On the other hand, the decoding complexity of MDS coding strategies is $O(m^3 + m^2)$.

Rateless fountain codes have demonstrated superior performance over MDS codes in providing robustness against erasures in communication channels which has led to their adoption in a variety of communication standards [17]. Due to the aforementioned benefits we believe that they can enjoy similar success in the realm of coded distributed computation as well. In the sequel we describe the details of the rateless coded algorithm for distributed matrix vector multiplication and validate its effectiveness in mitigating stragglers through theoretical analysis and numerical simulations.

C. Organization

The rest of the paper is organized as follows. In Section II we review previous works on coded distributed computing. Section III formally presents the system model, performance criterion and comparison benchmarks. In Section IV we describe the proposed rateless fountain coding strategy for distributed matrix vector multiplication. Section V and Section VI show theoretical analysis and simulation results comparing rateless codes with other strategies in terms of delay and number of redundant computations. Finally, Section VII presents ongoing and future research directions.

II. Background and Related Work

A. The Problem of Stragglers

Stragglers are slow computing nodes that slow down any distributed computation that depends on all nodes completing the task assigned to them. Straggling of nodes is widely observed in cloud infrastructure [6] and existing systems like MapReduce [19] and Spark [20] generally deal with this problem by launching replicas of straggling tasks, which are referred to as *back-up* tasks. This strategy of task replication has many variants such as [7], [8], and has been theoretically analyzed in [9] and [10] where schemes for adding redundant copies based on the tail of the runtime distribution at the workers are proposed.

Erasure codes were first employed to overcome the issue of straggling nodes in the context of content download from distributed storage [21], [22]. A content file that is divided into k chunks and coded using a (p, k) maximum-distance-separable code, can be recovered by downloading any k out of p encoded chunks. Queueing models to analyze the latency of coded content download jobs were proposed and analyzed in [23]–[27]. However, unlike distributed storage, erasure coding of computing jobs is not straightforward. A job with n parallel tasks needs to be designed such that the execution of any k out of n tasks is sufficient to complete the job.

B. Distributed Coded Computation

The idea that error-correcting codes can be used for fault-tolerance in computing tasks has its roots in works on algorithmic fault tolerance [4], [5]. Recently, this approach has spurred a rich body of research that uses erasure codes in modern distributed computing tasks including matrix-vector multiplication, matrix-matrix multiplication, and distributed gradient descent.

Matrix-vector multiplication: Two major approaches to computing distributed matrix vector products are the uncoded and MDS coded approaches. In the uncoded case the $m \times n$ matrix $\mathbf{A}$ is divided into p submatrices along the rows, and each worker multiplies one sub-matrix with the vector $\mathbf{x}$. The MDS coded approach of [12] provides straggler tolerance since the matrix-vector product can be decoded from the results of computations at any k out of p nodes.

Subsequent works like [14], [28] build upon the MDS coded approach by designing a sparse encoded matrix thus reducing the amount of computation at each node. Applications of coded matrix vector multiplication that have been proposed include computing the Fourier Transform of a signal [29] and the Page Rank of a graph [30].

In this work we introduce a scheme for coded distributed matrix vector multiplication that makes use of rateless erasure codes, specifically LT codes [15]. While the use of LT codes for matrix-vector multiplication has been recently proposed in [31], it does not utilize partial work done by straggling nodes, which is the key novel contribution of our work. Due to this our approach achieves near optimal straggler tolerance as well as negligible overhead of redundant computation.

Matrix-matrix multiplication: A natural generalization of matrix vector multiplication is matrix-matrix multiplication. In the distributed setting this involves splitting the 2 matrices to be multiplied into submatrices that are distributed and multiplied across the worker nodes of the system with the results being aggregated at the master. The authors of [32]

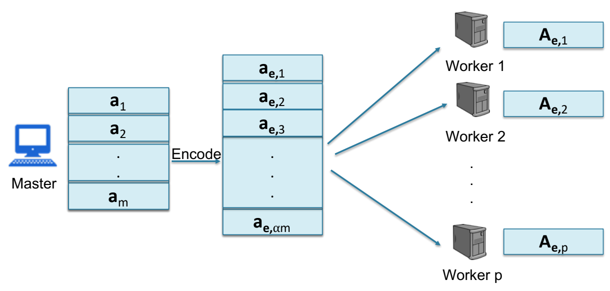


Fig. 1: The system model for coded distributed matrix vector multiplication with a master-worker framework. The master generates the encoded matrix $\mathbf{A_e}$ by applying a coding scheme to the rows of $\mathbf{A}$. $\mathbf{a}_1, \mathbf{a}_2, \ldots \mathbf{a}_m$ are the rows of $\mathbf{A}$, $\mathbf{a}_{\mathbf{e},1}, \mathbf{a}_{\mathbf{e},2}, \ldots \mathbf{a}_{\mathbf{e},\alpha m}$ are the rows of $\mathbf{A_e}$. Worker i stores a submatrix of $\mathbf{A_e}$ denoted by $\mathbf{A}_{\mathbf{e},i}$ for $i = 1, \ldots, p$.

propose a coding scheme called Polynomial Codes in which each node stores a linear combination of submatrices and the final product can be recovered using polynomial interpolation on the encoded products of submatrices obtained from the non-straggling nodes. Works that build upon this approach show that the minimum number of workers that need to complete their task for the decoding to succeed can be reduced further at the cost of increasing communication between the master and worker nodes [33], and also construct codes that can preserve the sparsity of input matrices by ensuring that the each encoded submatrix is a linear combination of a small number of submatrices of the original matrix [34].

While these are the major coded distributed computing tasks that have been studied, there are also works dealing with coded gradient descent [35]–[37], coded convolution [38], and coded distributed optimization [39]. In this work we address the original problem of coded distributed matrix vector multiplication and propose a new encoding scheme that uses the work done by each node in the system (including the stragglers) and can approach perfect load balancing in an asymptotic sense. In the future we plan to extend this approach to other applications like computing matrix-matrix products and coded gradient descent using the same underlying principles as in the present work.

III. Problem Formulation

A. System Model

Consider the problem of multiplying a $m \times n$ matrix $\mathbf{A}$ with a $n \times 1$ vector $\mathbf{x}$ using p worker nodes. A central fusion node, referred to as the master node collects the results of computing tasks assigned to the worker nodes. The task allocation is shown in Fig. 1. The worker nodes can only communicate with the master, and cannot directly communicate with other workers. The goal is to compute the result $\mathbf{b} = \mathbf{A}\mathbf{x}$ in a distributed fashion and mitigate the effect of unpredictable node slowdown or straggling.

Straggler mitigation is achieved through coding wherein the $m \times n$ matrix $\mathbf{A}$ is encoded using an error correcting code which operates on the rows of the matrix as source symbols to give the $m_e \times n$ encoded matrix $\mathbf{A_e}$. The workers compute the product $\mathbf{A_e x}$ in a distributed fashion. Every coding scheme adds some amount of redundant computations due to which $m_e > m$. This added redundancy mitigates the effect of stragglers because we only need a subset of the elements of $\mathbf{A_e x}$ to recover the desired matrix-vector product $\mathbf{Ax}$. We quantify the amount of redundancy added by the parameter $\alpha = m_e/m$.

Matrix $\mathbf{A_e}$ is split along its rows to give submatrices $\mathbf{A}_{\mathbf{e},1}, \ldots, \mathbf{A}_{\mathbf{e},p}$ with the task of computing the product $\mathbf{A}_{\mathbf{e},i}\mathbf{x}$ assigned to worker i. We assume that the submatrix $\mathbf{A}_{\mathbf{e},i}$ is stored in memory at worker i to enable fast access which is the case in real world distributed computing systems like Spark [20]. The vector $\mathbf{x}$ is provided as input to the system and is communicated by the master to all the workers when the computation of the product $\mathbf{Ax}$ is required.

Each worker needs to compute a sequence of row vector products of the form $\mathbf{a}_{\mathbf{e},j}\mathbf{x}$ where $\mathbf{a}_{\mathbf{e},j}$ is the j^{th} row of $\mathbf{A_e}$. The time taken by a worker node to finish computing one or more row-vector products may be random due to variability in the node speed, or variability in the complexity of the computation task itself. Nodes that run on a shared computing infrastructure may slow down significantly and unpredictably due to several factors such as outages, virtualization, congestion etc., as observed in the systems literature [6], [8]. We theoretically analyze the effect of such slow-down on various distributed computing schemes in Section V where we model the time taken by a worker node to finish computing as a random variable, and study how its probability distribution affects the performance of different coding strategies.

The master node aggregates the computations of all, or a subset of, the workers into the vector $\mathbf{b_e}$, which is then decoded to give the final result $\mathbf{b} = \mathbf{Ax}$. If $\mathbf{b_e}$ is not decodable, the master waits until more row-vector products are completed by the workers.

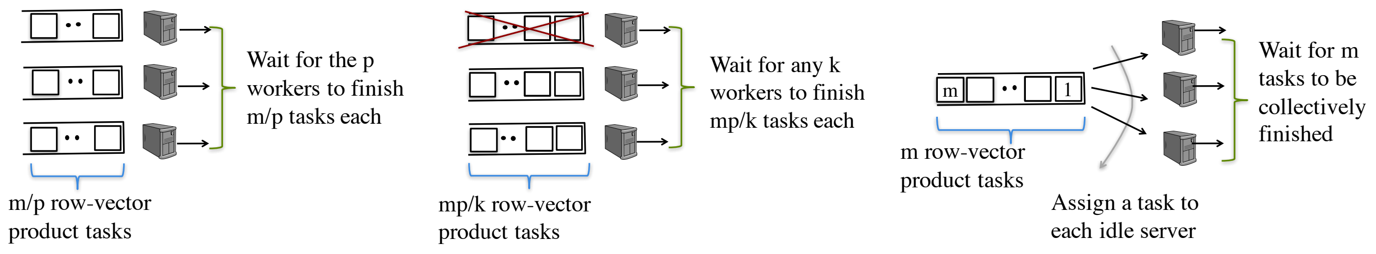
B. Performance Criterion

We use the following metrics to compare different coding schemes for distributed matrix vector multiplication.

Definition 1 (Latency (T)). *We define latency T as the time required by the system to complete enough number of computations so that the result $\mathbf{b}$ can be successfully decoded from the worker computations aggregated in $\mathbf{b_e}$.*

Definition 2 (Computations (C)). *he number of computations C is defined as the total number of row-vector products $\mathbf{a}_{\mathbf{e},j}\mathbf{x}$ performed collectively by the worker nodes until the vector $\mathbf{b} = \mathbf{Ax}$ is decoded.*

The matrix in question can be the original matrix $\mathbf{A}$ in the uncoded case, or the encoded matrix $\mathbf{A_e}$ in the coded case. The cost of a single computation in terms of number of symbol operations remains the same in both cases since encoding does not change the size of the matrix rows. Thus the overall computation cost is proportional to the total number of computations (C) performed by the system. The minimum


Wait for the p workers to finish m/p tasks each
m/p row-vector product tasks
Wait for any k workers to finish mp/k tasks each
mp/k row-vector product tasks
m
1
m row-vector product tasks
Assign a task to each idle server
Wait for m tasks to be collectively finished

(a) Uncoded Strategy (b) MDS-Coding Strategy (c) Dynamic Load-balancing

Fig. 2: In the uncoded scheme, we wait for the p workers to finish m/p row-vector products each. With MDS-coding, we only need to wait for k out of p workers, but each worker has to complete mp/k. The dynamic load-balancing strategy has a central queue of the m tasks, which are dynamically assigned to idle worker nodes.

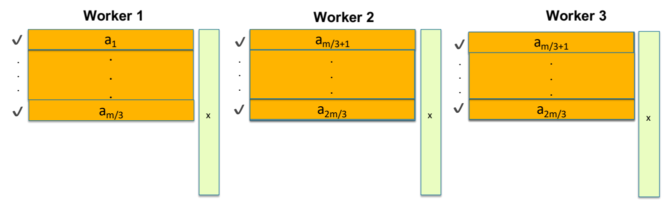


Fig. 3: In the uncoded Strategy we to wait for each worker to finish all m/p row-vector products that are assigned to that. Orange shading and check marks indicate completed row-vector products at the time of successful decoding of $\mathbf{b} = \mathbf{A}\mathbf{x}$.

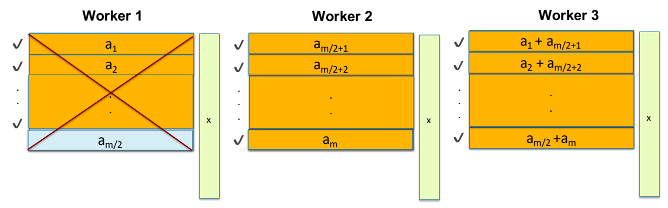


Fig. 4: In the (p, k) MDS-coded strategy we need to wait for any k out of p workers to finish m/k row-vector products each, and ignore the partial work done by straggling nodes. In this case we discard Worker 1's partial work.

number of computations that must be performed by the system is m since we wish to compute an m-dimensional matrix-vector product.

In Section V we analyze the trade-off between latency and computations by modeling the delay at each worker node as a random variable that depends on the number of row-vector product tasks performed by that worker. Our coded computing scheme achieves a win-win in the latency-computation trade-off by giving significantly lower expected latency than existing schemes for the same amount of redundant computation. In Section VI we present simulation results to validate the theoretical findings.

C. Benchmarks for Comparison

We compare the performance of the proposed rateless coded strategy with three benchmarks: the uncoded, MDS-coded, and dynamic load-balancing strategies, which are described formally below. Fig. 2 is an illustration of the differences in the way row-vector product tasks are completed by the worker nodes in each of these 3 strategies.

1) The Uncoded Strategy: A simple method to parallelize the multiplication of a $m \times n$ matrix $\mathbf{A}$ with a $n \times 1$ vector $\mathbf{x}$ is to distribute the computation equally over p worker nodes. We can split the rows of $\mathbf{A}$ into p submatrices $\mathbf{A}_1, \dots, \mathbf{A}_p$, with m/p rows each (assume that p divides m). Each submatrix is multiplied with $\mathbf{x}$ in parallel on one of the p worker nodes and the master aggregates the result into the $m \times 1$ vector $\mathbf{b}$. While this approach is efficient in the number of computations, the completion of the matrix-vector multiplication job can be bottlenecked by the slowest worker node.

2) The MDS-Coded Strategy: Recent works like [12], [13] have applied coding to overcome the problem of stragglers in the uncoded strategy. The strategy involves pre-multiplying $\mathbf{A}$ at the central node with a suitable encoding matrix $\mathbf{F}$ denoting the MDS codes. If a matrix is encoded using a (p, k) MDS code, the matrix $\mathbf{A}$ is split into k matrices $\mathbf{A}_1, \dots, \mathbf{A}_k$. The MDS code adds $p-k$ redundant matrices $\mathbf{A}_{k+1}, \dots, \mathbf{A}_p$ which are linear combinations of the matrices $\mathbf{A}_1, \dots, \mathbf{A}_k$. Worker node i computes the product of matrix $\mathbf{A}_i$ with vector $\mathbf{x}$. Thus the system is robust to $p - k$ stragglers. However this strategy adds a significant computation overhead. When none of the nodes are slow, the system performs mp/k row-vector products, whereas in the uncoded case it only performs m computations.

3) Dynamic Load Balancing: Dynamic Load Balancing (DLB) is an ideal strategy that enjoys all the benefits of parallelizing the matrix-vector multiplication, without any drawbacks. Here, the matrix-vector multiplication is treated as a job with m tasks, each task corresponding to one row-vector product. The central node assigns one task to each of the p workers. As soon as a worker finishes its task, it is assigned the next task. The matrix-vector multiplication is complete when exactly m tasks are collectively finished by the workers. Since none of the workers are idle at any point, the system is not bottlenecked by the slowest worker(s). The data (elements of the corresponding matrix row) required to complete the row-vector multiplication can be sent dynamically by the master node. This strategy may be impractical due to the constant communication between the master and the worker

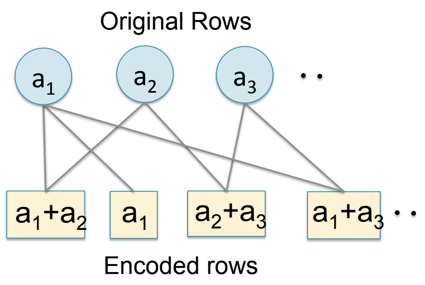


Fig. 5: Bipartite graph representation of the encoding of the rows $\mathbf{a}_1, \mathbf{a}_2, \dots \mathbf{a}_m$ of matrix $\mathbf{A}$. Each encoded row is the sum of d rows of $\mathbf{A}$ chosen uniformly at random, where d is drawn from the Robust Soliton degree distribution (1).

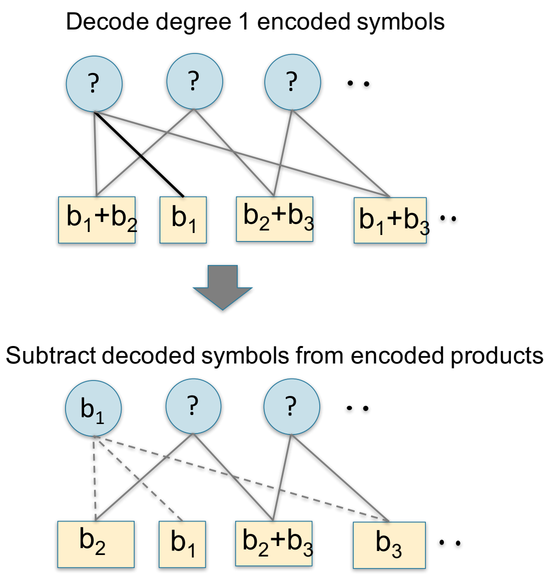

Decode degree 1 encoded symbols
?
?
?
b_1+b_2
b_1
b_2+b_3
b_1+b_3
Subtract decoded symbols from encoded products
b_1
?
?
b_2
b_1
b_2+b_3
b_3

Fig. 6: In each step of the iterative decoding process, a single degree one encoded symbol is decoded directly, and is subtracted from all sums in which it participates.

nodes, however, it serves as a good theoretical benchmark for comparing more practical strategies.

IV. Proposed Rateless Coding Strategy

In this section we propose the use of rateless codes to mitigate the effect of stragglers in computing the matrix-vector product $\mathbf{b} = \mathbf{A}\mathbf{x}$. Our algorithm achieves near-perfect load balancing and low latency, with negligible computation overhead. We describe our algorithm in two parts - first, we describe how LT codes [15], one of the first practically realizable rateless codes, can be applied to perform coded matrix vector multiplication, and then we describe a distributed implementation of this scheme using the master-worker framework described in Section III-A.

A. LT-Coded Matrix-vector Multiplication

LT codes, described by Luby in [15] are a class of rateless erasure codes that can be used to generate a potentially limitless number of encoded symbols from a set of m source symbols. The original source symbols can be recovered with a high probability from any M' encoded symbols using an iterative decoding algorithm based on belief propagation that is commonly called the *peeling decoder* [15]–[17]. Here M' is a random variable. In our analysis we deal with $m' = \mathbb{E}[M']$ which we formally define as the *decoding threshold* of our algorithm.

Definition 3 (Decoding Threshold (m'))**.** *We define decoding threshold m' as the* expected *number of encoded symbols required to decode a set of m source symbols under the proposed rateless coding strategy..*

To apply LT codes to the task of encoding the matrix-vector product $\mathbf{b} = \mathbf{A}\mathbf{x}$, we treat the rows of the matrix $\mathbf{A}$ as source symbols. Each encoded symbol is the sum of a randomly chosen subset of source symbols. Thus rows of the encoded matrix are given by $\mathbf{a}_{\mathbf{e},j} = \sum_{i \in S} \mathbf{a}_i$ where S is a random subset of rows of $\mathbf{A}$.

To choose S, we first choose the degree of the encoded symbol according to a degree distribution. The degree d of an encoded symbol is defined as the number of source symbols that contribute to the encoded symbol. If there are m source symbols (m rows), d is chosen according to the Robust Soliton degree distribution [15] wherein the probability of choosing $d = i$ is proportional to

$$\rho(d) = \begin{cases} \frac{R}{im} + \frac{1}{m} & \text{for } i = 1 \\ \frac{R}{im} + \frac{1}{m(m-1)} & \text{for } i = 2, \dots, m/R - 1 \\ \frac{R \ln(R/\delta)}{m} + \frac{1}{m(m-1)} & \text{for } i = m/R \\ \frac{1}{m(m-1)} & \text{for } i = m/R + 1, \dots, m \end{cases} \tag{1}$$

$R = c \log(m/\delta)\sqrt{m}$ for some $c > 0$ (c is a design parameter) and δ is the failure probability of the decoding algorithm and is thus chosen to lie in $[0, 1]$. Some guidelines for choosing c, and δ can be found in [40]. The exact probability of choosing $d = i$ can be found by normalizing ρ to sum to 1.

Once the degree d is chosen, encoding is performed by choosing d source symbols uniformly at random (this determines S) and adding them to generate an encoded symbol. The encoding process is illustrated in Fig. 5.

Once the rows of the encoded matrix $\mathbf{A}_\mathbf{e}$ are generated, we can compute the encoded matrix vector product $\mathbf{b}_\mathbf{e} = \mathbf{A}_\mathbf{e}\mathbf{x}$. To decode the desired matrix vector product $\mathbf{b} = \mathbf{A}\mathbf{x}$ from a subset of m' symbols of $\mathbf{b}_\mathbf{e}$ we use the the iterative peeling decoder described in [15]–[17]. If $\mathbf{b} = [b_1, b_2, \dots b_m]$, the decoder may receive symbols $b_1 + b_2 + b_3$, $b_2 + b_4$, b_3, b_4, and so on. Decoding is performed in an iterative fashion. In each iteration, the decoder finds a degree one encoded symbol, covers the corresponding source symbol, and subtracts the symbol from all other encoded symbols connected to that source symbols. This decoding process is illustrated in Fig. 6. The Robust Soliton degree distribution (1) used for encoding the rows of $\mathbf{A}$ ensures that with high probability there are

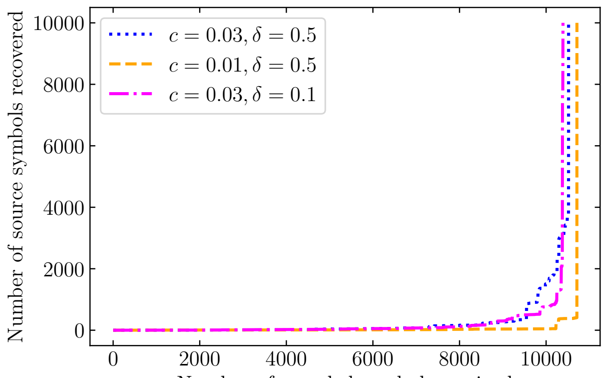


Fig. 7: The number of decoded symbols is almost constant until $m = 10,000$ encoded symbols are received after which it increases rapidly.

degree one symbols at each iteration, thus leading to fast and low-complexity decoding.

Fig. 7, shows simulation results for the number of symbols decoded successfully for each encoded symbol received. For this we perform LT-Coded multiplication of a randomly generated $10,000 \times 10,000$ matrix with a $10,000 \times 1$ vector. The matrix $\mathbf{A}$ is encoded using an LT code with parameters c and δ chosen according to the guidelines of [40]. We generate a single row of the encoded matrix $\mathbf{A_e}$ at a time which is then multiplied with the $10,000 \times 1$ size vector $\mathbf{x}$ to give a single element of the encoded matrix vector product $\mathbf{b_e}$. The process is repeated until we have enough symbols for successfully decoding the entire $10,000 \times 1$ size vector $\mathbf{b}$ using the peeling decoder. The plots of Fig. 7 correspond to different choices of c and δ. In each case we observe an avalanche behavior wherein very few symbols are decoded up to a point (approximately upto $10,000$ encoded symbols received) after which the decoding proceeds very rapidly to completion.

The theoretical encoding and decoding properties of LT codes are summarized in the following lemmas:

Lemma 1 (Theorem 13 in [15])**.** *For any constant $\delta > 0$, the average degree of an encoded symbol is $\mathcal{O}(\log(m/\delta))$ where m is the number of source symbols.*

Corollary 1. *Each encoding symbol can be generated using $\mathcal{O}(\log m)$ symbol operations on average.*

Lemma 2 (Theorem 17 in [15])**.** *For any constant $\delta > 0$ and for a source block with m source symbols, the LT decoder can recover all the source symbols from a set of $m_e = m + \mathcal{O}(\sqrt{m}) \log^2(m/\delta)$ with probability at least $1 - \delta$.*

Corollary 2. *The decoding threshold m' as defined in Definition 3 is given by $m' = m(1 + \epsilon)$ where $\epsilon \to 0$ as $m \to \infty$*

Corollary 3. *Since the average degree of an encoded symbol is $\mathcal{O}(\log(m/\delta))$ the decoding requires $\mathcal{O}(m \log m)$ symbol operations on average.*

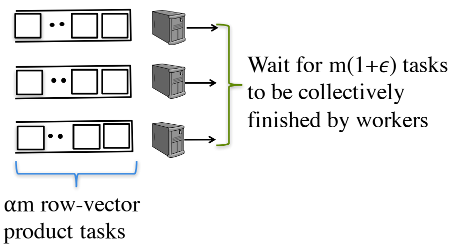

Wait for m(1+ϵ) tasks to be collectively finished by workers
αm row-vector product tasks

Fig. 8: Queueing model for the rateless LT-coded strategy. For large α and m, its latency is equivalent to the dynamic load balancing scheme shown in Fig. 2.

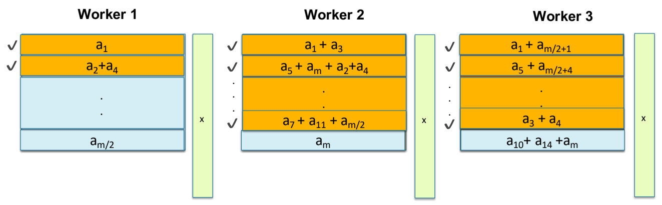


Fig. 9: In the rateless LT-coded strategy we need to wait for any $m' \simeq m(1 + \epsilon)$ row-vector products to be computed collectively by worker nodes. Orange shading and check marks indicate completed row-vector products at the time of successful decoding of $\mathbf{b} = \mathbf{A}\mathbf{x}$. Partial work done by all nodes (indicated by check marks) is used for decoding.

B. *Distributed Implementation*

We now describe the implementation of LT-Coded matrix vector multiplication in a distributed master-worker framework. The encoding and decoding tasks are performed at the master while the workers multiply rows of the encoded matrix with the vector $\mathbf{x}$ i.e. they compute products of the form $\mathbf{a}_{\mathbf{e},j}\mathbf{x}$ and communicate the result to the master.

The encoding step can be treated as a pre-processing step in that it is only performed initially. Thus the $m \times n$ matrix $\mathbf{A}$ is encoded by treating the rows $\mathbf{a}_1, \mathbf{a}_2, \ldots \mathbf{a}_m$ of the matrix as source symbols to generate $m_e = \alpha m$ rows of the encoded matrix $\mathbf{A_e}$ as described in Section IV-A. Since the general motivation for performing distributed matrix vector multiplication is that $\mathbf{A}$ is too large to be stored in memory at a single machine, the encoding step can be performed by accessing rows of $\mathbf{A}$ stored in a distributed file system [41]–[43] and computing their sum to generate a row of $\mathbf{A_e}$. While the cost of communicating matrix rows over a distributed file system may be high, we note that this is a one-time cost since the matrix is encoded only once.

The rows of the encoded matrix are then distributed among the worker nodes. Each worker node is assigned an equal number of rows of $\mathbf{A_e}$ as illustrated in Fig. 1. The $\alpha m/p$ rows assigned to each node are stored in its local memory. This can be accomplished using distributed memory abstractions like Resilient Distributed Datasets [20]. Subsequently, whenever we wish to multiply $\mathbf{A}$ with a vector $\mathbf{x}$, the master communicates $\mathbf{x}$ to the workers. Each worker multiplies $\mathbf{x}$ with each row of $\mathbf{A_e}$ stored in its memory and returns the product (a scalar) to the master. Alternately, to minimize communication,

Strategy	Latency	Computations	Complexity
LT-coded	$\frac{\tau m(1+\epsilon)}{p} + \frac{1}{\mu}$	$m(1+\epsilon)$	$O(m \log m)$
Uncoded	$\frac{\tau m}{p} + \frac{1}{\mu} \log p$	m	$O(m)$
MDS	$\frac{\tau m}{k} + \frac{1}{\mu} \log \frac{p}{p-k}$	mp/k	$O(m^3 + m^2)$
DLB	$\frac{\tau m}{p} + \frac{1}{\mu}$	m	$O(m)$

TABLE I: Comparison of different strategies to multiply an $m \times n$ size matrix $\mathbf{A}$ with vector $\mathbf{x}$ using p worker nodes. The latency values are approximate, and computation values are for the case when none of the nodes slowdown.

the worker may only send progress updates to the master node indicating the number of row-product computation tasks it has completed, and send the products only upon request by the master.

The master node receives coded row-vector products from the worker nodes. Since the master is aware of the mapping, such as in Fig. 5, from source symbols to encoded symbols, it can use this knowledge to recover the desired matrix vector product $\mathbf{b} = \mathbf{A}\mathbf{x}$ from a subset of the elements of $\mathbf{b_e} = \mathbf{A_e}\mathbf{x}$ using the iterative peeling decoder. For a matrix with m rows, the decoding process will require $m' = m(1+\epsilon)$ row-vector products, in expectation, where ϵ is a small overhead, such that $\epsilon \to 0$ as $m \to \infty$, that depends on the parameters of the Robust Soliton distribution defined in (1).

Once the master decodes all the elements in the product vector $\mathbf{b} = \mathbf{A}\mathbf{x}$, it sends a *done* signal to all workers nodes to stop their local computation. The workers will continue sending row-vector products to the master, until the master sends the *done* signal. A worker node may complete all the $\alpha m/p$ row-vector products assigned to it before receiving the *done* signal. This worker will then remain idle, while the master collects more row-vector products from other workers until it is able to decode $\mathbf{b}$. Figures 8 and 9 illustrate the queueing model, and a 3-node example of the rateless coding strategy respectively.

While we have used LT Codes [15] for our analysis and simulations due to their ease of implementation and fast decoding, we note that any rateless random linear network code which generates αm ($\alpha > 1$) encoded symbols that are linear independent with high probability can be used for encoding the m rows of matrix $\mathbf{A}$. A larger value of α implies higher redundancy, and more tolerance to straggling nodes. In particular one can consider more advanced rateless codes such as Raptor Codes [16], [17] and Online Codes [44]. Raptor Codes have better encoding and decoding properties than LT codes and we are exploring using them instead of LT codes in our ongoing systems implementation.

V. Theoretical Analysis

In this section we discuss our main theoretical results, which are summarized in Table 1. We first describe our model for computation delays at each worker node, and then compare the expected latency and computations of the proposed rateless LT coding strategy with the benchmarks described in Section III.

A. Delay Model and Order Statistics Primer

We assume that worker i requires time Y_i to perform C_i computations where

$$Y_i = X_i + \tau C_i, \quad \text{for all } i = 1, \dots, p \tag{2}$$

and $X_i \sim \exp(\mu)$, is an exponential random variable with rate μ. Thus, the delay involves the sum of two components – an exponential random variable X_i that models the network latency, initial setup time, and other random components, and a constant shift that is linear in the number of computations. If a worker completes C_i row-vector products this shift is $C_i\tau$, where τ is the time taken to perform each computation. The distribution of the per-worker runtime can be shown to be

$$\mathbb{P}(Y_i \le t) = 1 - \exp(-\mu(t - \tau C)) \tag{3}$$

While this follows the shifted exponential delay models used in [12], [14] and [38], the key difference is that the shift is parameterized by the number of computations at each worker. We believe this is a more realistic model since it clearly captures the effect of increasing the amount of computations on the delay – if a worker is assigned more computations, there is larger delay. And unlike previous works, the decay rate μ of the exponential part of the delay does not change with the number of computations performed by that worker.

We now state some standard results [45] on order statistics of exponential random variables to aid the understanding of the latency analysis presented below. If $X_1, X_2, \dots X_p$ are exponential random variables with rate μ, their k^{th} order statistic is denoted by $X_{k:p}$. Thus, $X_{1:p} = \min(X_1, X_2, \dots X_p)$, and $X_{p:p} = \max(X_1, X_2, \dots X_p)$. The expected value of $X_{k:p}$ is given by

$$\mathbb{E}[X_{k:p}] = \frac{1}{\mu}\left(\frac{1}{p} + \cdots + \frac{1}{p-k+1}\right), \tag{4}$$

$$= \frac{H_p - H_{p-k}}{\mu}, \tag{5}$$

where H_p is the p^{th} Harmonic number

$$H_p \triangleq \begin{cases} \sum_{i=1}^{p} \frac{1}{i} & \text{for } p = 1, 2, \dots \\ 0 & \text{for } p = 0 \end{cases} \tag{6}$$

For large p, $H_p = \log p + \gamma$, where γ is the Euler-Mascheroni constant and thus we can use the approximation $H_p \simeq \log p$ for large p.

B. Latency Analysis

In what follows we derive expressions for the expected latency of each of the coded computing schemes studied so far. We use T_{uncoded}, T_{MDS}, T_{LT}, to represent the latency of the uncoded, MDS-coded, and LT-coded schemes respectively.

Theorem 1 (Latency of the Uncoded Strategy)**.** *The expected latency for the uncoded strategy with p worker nodes is*

$$\mathbb{E}[T_{\text{uncoded}}] = \frac{\tau m}{p} + \frac{1}{\mu} H_p \simeq \frac{\tau m}{p} + \frac{1}{\mu}\log p \tag{7}$$

where H_p is the p^{th} Harmonic number.

Proof. The latency T_{uncoded} in the uncoded case is the time until all p workers finish m/p tasks each, which is

$$T_{\text{uncoded}} = \max(Y_1, Y_2, \ldots, Y_p), \tag{8}$$
$$= \max(X_1 + \tau C_1, X_2 + \tau C_2, \ldots, X_p + \tau C_p), \tag{9}$$

where (8) is obtained by using the delay model defined in (2). Since each worker performs m/p row-vector products, we substitute $C_1 = C_2 = \ldots = C_p = m/p$ and take expectation on both sides to obtain

$$\mathbb{E}[T_{\text{uncoded}}] = \frac{\tau m}{p} + \mathbb{E}[\max(X_1, X_2, \ldots, X_p)], \tag{10}$$
$$= \frac{\tau m}{p} + \frac{1}{\mu} H_p, \tag{11}$$
$$\simeq \frac{\tau m}{p} + \frac{1}{\mu} \log p, \tag{12}$$

where (11) and (12) follow from (5) and (6). □

Theorem 2 (Latency of the MDS-Coded Strategy)**.** *The expected latency for the MDS-coded case is given by*

$$\mathbb{E}[T_{MDS}] = \frac{\tau m}{k} + \frac{1}{\mu}(H_p - H_{p-k}) \simeq \frac{\tau m}{p} + \frac{1}{\mu} \log \frac{p}{p-k}. \tag{13}$$

Proof. The latency in the MDS-coded case is $T_{\text{MDS}} = Y_{k:p}$, where $Y_{k:p}$ is the k^{th} order statistic of the individual worker latencies $Y_1, Y_2, \ldots, Y_p$ since we only wait for the fastest k workers to finish the task assigned to them. In this case, each of the fastest k workers performs $\frac{m}{k}$ computations and thus the expected overall latency is given by

$$\mathbb{E}[T_{\text{MDS}}] = \mathbb{E}[X_{k:p}] + \tau \frac{m}{k}, \tag{14}$$
$$= \frac{\tau m}{k} + \frac{1}{\mu}(H_p - H_{p-k}), \tag{15}$$
$$\simeq \frac{\tau m}{k} + \frac{1}{\mu} \log \frac{p}{p-k}. \tag{16}$$

where (15) and (16) follow from the exponential order statistics results in (5) and (6). □

Thus in the MDS-coded case there is a tradeoff between the two terms in the above expression. A lower value of k increases the number of redundant computations thus increasing $\tau m/k$ but it decreases the effect of straggling workers which is quantified by the decrease in $\log(p/(p-k))$. Thus our delay model effectively captures the trade-off associated with increasing k in the MDS-coded case. This is also illustrated by our simulation results in the next section.

Theorem 3 (Latency of the Rateless Coding Strategy)**.** *For large m_e i.e. $\alpha = m_e/m \to \infty$, the expected latency for the LT-coded case has the following upper and lower bounds.*

$$\mathbb{E}[T_{LT}] \leq \frac{\tau m'}{p} + \frac{1}{\mu} + \tau, \tag{17}$$
$$\mathbb{E}[T_{LT}] \geq \frac{\tau m'}{p} + \frac{1}{p\mu}, \tag{18}$$

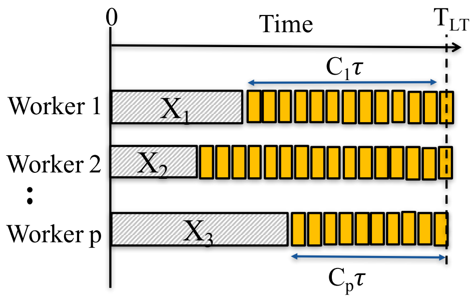


Fig. 10: Worker i has an random exponential initial delay X_i, after which it completes row-vector product tasks (denoted by the small rectangles), taking τ seconds per task. The latency T_{LT} is the time to complete m' tasks in total.

where $m' = m(1+\epsilon)$ is the expected number of symbols necessary for successful decoding.

Proof. As per our model, the time taken by worker i to perform C_i computations is given by

$$Y_i = X_i + \tau C_i, \quad \text{for } i = 1, \ldots, p. \tag{19}$$

The latency T_{LT} is the earliest time when $\sum_{i=1}^{p} C_i = m'$, as illustrated in Fig. 10. We note that, in this case it is not necessary that each worker has completed at least 1 computation. Specifically, if $T_{\text{LT}} - X_i \leq \tau$ for any i then it means that worker i has not performed even a single computation in the time that the system as a whole has completed m' computations (owing to the large initial delay X_i). Therefore we define

$$\mathcal{W}_{\text{LT}} := \{i : T_{\text{LT}} - X_i \geq \tau\} \tag{20}$$

Here $\mathcal{W}_{\text{LT}}$ is the set of workers for which $C_i > 0$. Thus

$$T_{\text{LT}} = \max_{i \in \mathcal{W}_{\text{LT}}} Y_i, \tag{21}$$
$$= \max_{i \in \mathcal{W}_{\text{LT}}} (X_i + \tau C_i), \tag{22}$$
$$\geq \min_{i \in [1,\ldots p]} X_i + \tau \max_{i \in \mathcal{W}_{\text{LT}}} C_i, \tag{23}$$
$$\geq \min_{i \in [1,\ldots p]} X_i + \tau \frac{m'}{p}, \tag{24}$$

where to obtain (23), we replace each X_i in (22) by $\min_{i \in [1,\ldots p]} X_i$ and then we can bring it outside the maximum. To obtain (24), we observe that in order for the p workers to collectively finish m' computations, the maximum number of computations completed by a worker has to be at least m/p. Taking expectation on both sides we get

$$\mathbb{E}[T_{\text{LT}}] \geq \mathbb{E}[\min(X_1, X_2, \ldots X_p)] + \frac{\tau m'}{p}, \tag{25}$$
$$= \frac{1}{p\mu} + \frac{\tau m'}{p}. \tag{26}$$

where the lower bound in (26) follows from the result (5) on order statistics of exponential random variables.

To derive the upper bound, we note that

$$T_{\text{LT}} \leq X_i + \tau(C_i + 1), \quad \text{for all } i = 1, \ldots, p \tag{27}$$

This is because at time T_{LT} each of the workers $1, \dots, p$, have completed $C_1, \dots, C_p$ row-vector product tasks respectively, but they may have partially completed the next task. The 1 added to each C_i accounts for this edge effect, which is also illustrated in Fig. 10.

Summing over all i on both sides, we get

$$\sum_{i=1}^{p} T_{\text{LT}} \leq \sum_{i=1}^{p} X_i + \sum_{i=1}^{p} \tau(C_i + 1) \tag{28}$$

$$pT_{\text{LT}} \leq \sum_{i=1}^{p} X_i + \tau(m' + p) \tag{29}$$

Taking expectation on both sides and rearranging we obtain the upper bound,

$$p\mathbb{E}[T_{\text{LT}}] \leq \frac{p}{\mu} + \tau(m' + p), \tag{30}$$

$$\mathbb{E}[T_{\text{LT}}] \leq \frac{\tau m'}{p} + \frac{1}{\mu} + \tau. \tag{31}$$

□

Corollary 4. *The expected latency of the Dynamic Load-Balancing (DLB) scheme described in Section III-C is identical to the LT-coded scheme with $m' = m$ and hence follows the above upper and lower bounds of Theorem 3 with $m' = m$. Thus for large m, when $m' \to m$, the expected latency of rateless coding converges to that of dynamic load balancing.*

Remark 1. *For finite values of m_e, if each worker is assigned m_e/p computations, it is possible that not all values of C_i that satisfy $\sum_{i=1}^{p} C_i = m'$ are permissible since we also need to enforce the constraint $C_i \leq m_e/p$. Hence we need large redundancy $(\alpha = m_e/m \to \infty)$ for the results of Theorem 3 to be strictly true. However in Section VI we show through simulations that even for fairly small amounts of redundancy $(\alpha = m_e/m = 2)$ the latency distribution for the LT and DLB schemes is almost identical.*

C. Computations and Decoding Complexity

Table I shows a comparison of the different strategies in terms of the total number of row-vector product computations performed by the workers until $\mathbf{b}$ can be decoded. There are no redundant computations in the uncoded and dynamic load balancing schemes, and thus $C = m$. In the MDS coding scheme, each nodes is assigned m/k computations, and thus if there is no node slowdown or straggling, the total number of computations is mp/k, which is a significant computation overhead. In the rateless LT coding scheme, the iterative decoding process succeeds upon collectively receiving $m(1 + \epsilon)$ symbols from the nodes, where $\epsilon \to 0$ as $m \to \infty$. Thus, rateless coding is asymptotically optimal in the number of computations.

Next, let us compare the decoding complexity of the strategies, which are also given in Table I. The uncoded and DLB strategies again have an optimal decoding complexity $O(m)$, with $O(1)$ operations per symbol. For MDS codes, the decoding involves inverting an $m \times m$ matrix which takes $O(m^3)$ operations, and multiplying it with an $m \times 1$ vector which takes $O(m^2)$ operations, resulting in an overall decoding complexity $O(m^3 + m^2)$. Rateless LT codes create each encoded row with $O(\log m)$ rows of $\mathbf{A}$ and use an iterative decoding process which needs $O(\log m)$ operations per symbol, which results in a decoding complexity $O(m \log m)$.

VI. Simulation Results

In this section we present simulations for comparing the latency and computations of the various distributed computing schemes discussed so far - the uncoded scheme, the MDS-coded scheme, the LT-coded scheme, and the Dynamic Load Balancing (DLB) scheme. In our simulations the number of rows $m = 10,000$, and number of worker nodes $p = 10$. We assume that the total time T_i taken by worker i to complete its assigned task is given by the delay model (2) with rate parameter $\mu = 5.0$ and the time to compute one row-vector product $\tau = \mu/1000 = 0.005$.

The uncoded scheme involves assigning $m/p = 1000$ computations to each worker and waiting for all the workers to finish all the computations assigned to them.

For the MDS-coded scheme we use a (p, k) MDS code with $k = p - 2 = 8$. Thus each processor is assigned $\frac{m}{k} = 1250$ computations and we wait for the fastest $k = 8$ processors to complete all the tasks assigned to them.

For the LT coded scheme we consider two settings. In the first setting we use an LT code with parameters $R = 0.03, \delta = 0.5, \alpha = 1.25$. This choice of α ensures that $\alpha m = 12,500$ and thus the number of redundant computations is the same as that for the MDS case. In the second setting we increase α to 2.0 due to which $\alpha m = 20,000$ to observe if it leads to any further improvement in performance. We note that the redundant storage in the second case is the same as the case where each submatrix is replicated across 2 nodes which is often done in real distributed computing systems like [19], [20]. In the LT coded scheme we wait for the fastest m' computations where m' is the minimum number of rows required for the decoding algorithm to be successful.

For the dynamic load balancing scheme we just wait for a total of m computations to be performed across all workers. The results of our experiments are shown in figures 11 to 13 All plots are generated using 500 Monte-Carlo simulations.

As can be seen from Figures 11 to 13, the LT-coded approach with $\alpha = 2.0$ outperforms both the uncoded and MDS-coded scheme in terms of latency. Moreover with a high probability, it also performs fewer computations than the MDS coded scheme. Remarkably, the latency performance of the LT-coded scheme with $\alpha = 2.0$ is almost indistinguishable from the ideal Dynamic Load-Balancing (DLB) scheme. Thus with the same amount of redundant storage as present in the replication-based schemes of existing distributed computing systems, we are able to outperform previous coded computing approaches and show near ideal performance.

VII. Concluding Remarks

Due to the massive size of matrices arising in modern data-driven applications, computations such as matrix-vector multi-

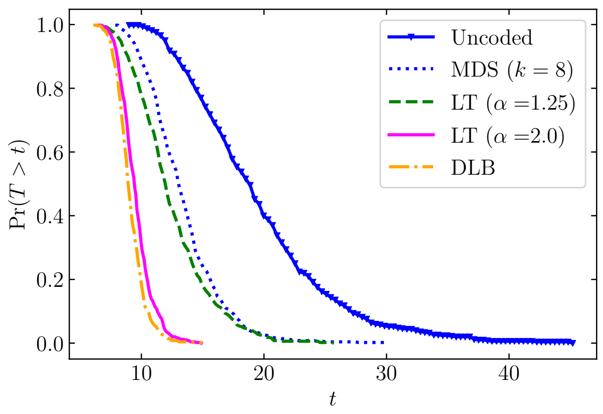


Fig. 11: Comparison of the tail distribution of the latency for different coding schemes. As expected, the uncoded scheme has the heaviest tail. The $(10, 8)$ MDS code and LT codes with $\alpha = 1.25 = 10/8$ have similar tail latency, but on increasing α to 2.0, the latency tail achieved by LT codes approaches the ideal Dynamic Load Balancing (DLB) scheme.

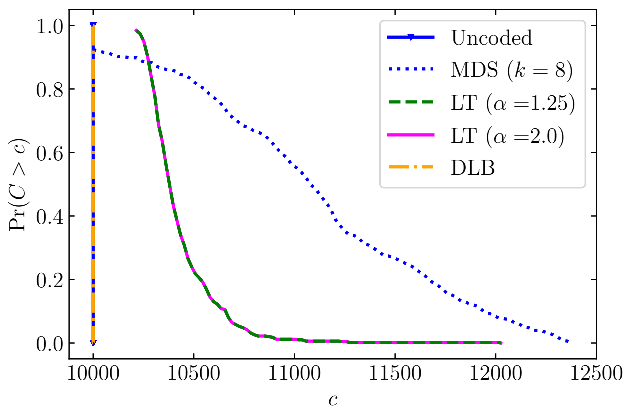


Fig. 12: In the uncoded and DLB schemes, the workers collectively perform exactly $m = 10,000$ row-vector product computations until $\mathbf{b} = \mathbf{Ax}$ can be decoded. MDS and LT coding schemes need redundant computations in order to reduce the latency tail (as shown in Fig. 11), but LT coded schemes (both $\alpha = 1.25$ and 2.0) perform significantly fewer redundant computations.

plication need to be parallelized across multiple nodes. In this paper we propose an erasure coding strategy based on *rateless fountain codes* to overcome bottlenecks caused by slow or straggling nodes. For a matrix with m rows, our strategy requires the nodes to collectively finish slightly more than m row-vector products, and thus it can seamlessly adapt to varying node speeds and achieve near-perfect load balancing. Moreover, it has a small overhead of redundant computations (asymptotically zero), and low decoding complexity. Thus, it strikes a better latency-computation trade-off than the uncoded and fixed-rate erasure coding strategies.

We are currently working on implementing the proposed rateless coding scheme in a real distributed computing cluster

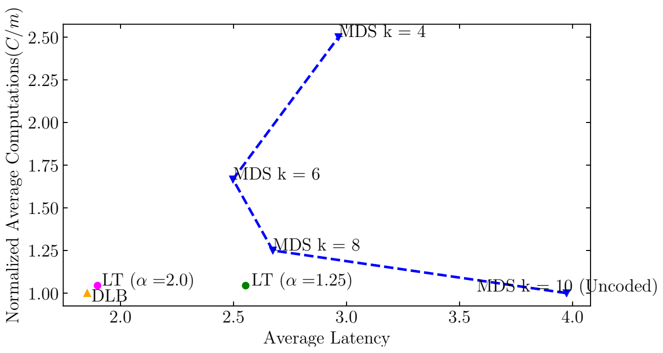


Fig. 13: Trade-off between average latency (T) and normalized average computations (C/m) for different parameter settings of the MDS and LT coded schemes shows that adding redundancy (decreasing k) in MDS codes eventually leads to an increase in both latency and number of computations. However, for LT codes adding redundancy (increasing α) reduces latency while keeping the number of computations constant, causing it to approach the ideal DLB scheme.

and will evaluate its performance for large-scale data processing applications such as neural network inference and PageRank. We also plan to investigate the theory behind LT codes and other improved versions of fountain codes such as Raptor-Q codes [17] and systematic fountain codes. More broadly, this work demonstrates that rateless codes are superior to fixed-rate coding strategies for the purpose of adapting to variability and heterogeneity in node speeds, as well as node failures. While we have only considered matrix-vector multiplication so far, we believe that the core rateless coding idea is applicable to a wide range of linear computations such as matrix-matrix multiplication, convolution and Fourier Transforms.

Acknowledgments

The authors are grateful to Pulkit Grover, Sanghamitra Dutta, Yaoqing Yang and Haewon Jeong for helpful discussions. Author Joshi also sincerely thanks Emina Soljanin, Alyson Fox, Fiona Knoll and Nadia Kazemi for fruitful initial discussions during the Women in Data Science and Mathematics (WiSDM) Research Collaboration Workshop held at Brown University in July 2017.

References

[1] William F Ames, *Numerical Methods for Partial Differential Equations*, Academic Press, 2014.

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

[3] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd, “The pagerank citation ranking: Bringing order to the web.,” Tech. Rep., Stanford InfoLab, 1999.

[4] Vipin Kumar, Ananth Grama, Anshul Gupta, and George Karypis, *Introduction to Parallel Computing: Design and Analysis of Algorithms*, vol. 400, Benjamin/Cummings Redwood City, 1994.

[5] Geoffrey C. Fox, Steve W. Otto, and Anthony JG. Hey, “Matrix algorithms on a hypercube i: Matrix multiplication,” *Parallel Computing*, vol. 4, no. 1, pp. 17–31, 1987.

[6] Jeffrey Dean and Luiz André Barroso, “The tail at scale,” *Communications of the ACM*, vol. 56, no. 2, pp. 74–80, 2013.

[7] Ganesh Ananthanarayanan, Srikanth Kandula, Albert G Greenberg, Ion Stoica, Yi Lu, Bikas Saha, and Edward Harris, "Reining in the outliers in map-reduce clusters using mantri.," in *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2010, vol. 10, p. 24.

[8] Ganesh Ananthanarayanan, Ali Ghodsi, Scott Shenker, and Ion Stoica, "Effective straggler mitigation: Attack of the clones.," in *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2013, vol. 13, pp. 185–198.

[9] Da Wang, Gauri Joshi, and Gregory Wornell, "Efficient task replication for fast response times in parallel computation," in *ACM SIGMETRICS Performance Evaluation Review*. ACM, 2014, vol. 42, pp. 599–600.

[10] Da Wang, Gauri Joshi, and Gregory Wornell, "Using straggler replication to reduce latency in large-scale parallel computing," *ACM SIGMETRICS Performance Evaluation Review*, vol. 43, no. 3, pp. 7–11, 2015.

[11] Kuang-Hua Huang et al., "Algorithm-based fault tolerance for matrix operations," *IEEE Transactions on Computers*, vol. 100, no. 6, pp. 518–528, 1984.

[12] Kangwook Lee, Maximilian Lam, Ramtin Pedarsani, Dimitris Papailiopoulos, and Kannan Ramchandran, "Speeding up distributed machine learning using codes," *IEEE Transactions on Information Theory*, 2017.

[13] Songze Li, Mohammad Ali Maddah-Ali, and A Salman Avestimehr, "A unified coding framework for distributed computing with straggling servers," in *IEEE Global Communications Conference (GLOBECOM) Workshops*. IEEE, 2016, pp. 1–6.

[14] Sanghamitra Dutta, Viveck Cadambe, and Pulkit Grover, "Short-dot: Computing large linear transforms distributedly using coded short dot products," in *Advances In Neural Information Processing Systems*, 2016, pp. 2100–2108.

[15] Michael Luby, "LT Codes," in *Foundations of Computer Science, 2002. Proceedings. The 43rd Annual IEEE Symposium on*. IEEE, 2002, pp. 271–280.

[16] Amin Shokrollahi, "Raptor codes," *IEEE transactions on information theory*, vol. 52, no. 6, pp. 2551–2567, 2006.

[17] Amin Shokrollahi, Michael Luby, et al., "Raptor codes," *Foundations and trends in communications and information theory*, vol. 6, no. 3–4, pp. 213–322, 2011.

[18] Gauri Joshi, Joong Bum Rhim, John Sun, and Da Wang, "Fountain codes," in *Technical Report*, 2010, pp. 7–12.

[19] Jeffrey Dean and Sanjay Ghemawat, "Mapreduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[20] Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, and Ion Stoica, "Spark: Cluster computing with working sets.," *HotCloud*, vol. 10, no. 10-10, pp. 95, 2010.

[21] Longbo Huang, S. Pawar, Hao Zhang, and K. Ramchandran, "Codes can reduce queueing delay in data centers," in *IEEE International Symposium on Information Theory Proceedings (ISIT)*, July 2012, pp. 2766–2770.

[22] Gauri Joshi, Yanpei Liu, and Emina Soljanin, "Coding for fast content download," in *Allerton Conference on Communication, Control, and Computing*. IEEE, 2012, pp. 326–333.

[23] 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 of Communications*, vol. 32, no. 5, pp. 989–997, May 2014.

[24] Gauri Joshi, Emina Soljanin, and Gregory Wornell, "Queues with redundancy: Latency-cost analysis," *ACM SIGMETRICS Performance Evaluation Review*, vol. 43, no. 2, pp. 54–56, 2015.

[25] Nihar B. Shah, Kangwook Lee, and Kannan Ramchandran, "When do redundant requests reduce latency?," *IEEE Transactions on Communications*, vol. 64, no. 2, pp. 715–722, Feb 2016.

[26] Kangwook Lee, Nihar B. Shah, Longbo Huang, and Kannan Ramchandran, "The mds queue: Analysing the latency performance of erasure codes," *IEEE Transactions on Information Theory*, vol. 63, no. 5, pp. 2822–2842, May 2017.

[27] Gauri Joshi, Emina Soljanin, and Gregory Wornell, "Efficient redundancy techniques for latency reduction in cloud systems," *ACM Transactions on Modeling and Performance Evaluation of Computing Systems*, vol. 2, no. 12, may 2017.

[28] Geewon Suh, Kangwook Lee, and Changho Suh, "Matrix sparsification for coded matrix multiplication," in *Allerton Conference on Communication, Control, and Computing*. IEEE, 2017, pp. 1271–1278.

[29] Qian Yu, Mohammad Ali Maddah-Ali, and A Salman Avestimehr, "Coded fourier transform," *arXiv preprint arXiv:1710.06471*, 2017.

[30] Yaoqing Yang, Pulkit Grover, and Soummya Kar, "Coded distributed computing for inverse problems," in *Advances in Neural Information Processing Systems*, 2017, pp. 709–719.

[31] Albin Severinson, Alexandre Graell i Amat, and Eirik Rosnes, "Block-diagonal and lt codes for distributed computing with straggling servers," *arXiv preprint arXiv:1712.08230*, dec.

[32] Qian Yu, Mohammad Maddah-Ali, and Salman Avestimehr, "Polynomial codes: an optimal design for high-dimensional coded matrix multiplication," in *Advances in Neural Information Processing Systems*, 2017, pp. 4406–4416.

[33] Sanghamitra Dutta, Mohammad Fahim, Farzin Haddadpour, Haewon Jeong, Viveck Cadambe, and Pulkit Grover, "On the optimal recovery threshold of coded matrix multiplication," *arXiv preprint arXiv:1801.10292*, 2018.

[34] Sinong Wang, Jiashang Liu, and Ness Shroff, "Coded sparse matrix multiplication," *arXiv preprint arXiv:1802.03430*, 2018.

[35] Rashish Tandon, Qi Lei, Alexandros G Dimakis, and Nikos Karampatziakis, "Gradient coding: Avoiding stragglers in synchronous gradient descent," *stat*, vol. 1050, pp. 8, 2017.

[36] Wael Halbawi, Navid Azizan-Ruhi, Fariborz Salehi, and Babak Hassibi, "Improving distributed gradient descent using reed-solomon codes," *arXiv preprint arXiv:1706.05436*, 2017.

[37] Songze Li, Seyed Mohammadreza Mousavi Kalan, A Salman Avestimehr, and Mahdi Soltanolkotabi, "Near-optimal straggler mitigation for distributed gradient methods," *arXiv preprint arXiv:1710.09990*, 2017.

[38] Sanghamitra Dutta, Viveck Cadambe, and Pulkit Grover, "Coded convolution for parallel and distributed computing within a deadline," in *IEEE International Symposium on Information Theory (ISIT)*. IEEE, 2017, pp. 2403–2407.

[39] Can Karakus, Yifan Sun, and Suhas Diggavi, "Encoded distributed optimization," in *IEEE International Symposium on Information Theory (ISIT)*. IEEE, 2017, pp. 2890–2894.

[40] David JC MacKay, *Information theory, Inference and Learning Algorithms*, Cambridge university press, 2003.

[41] John H Howard et al., *An overview of the andrew file system*, Carnegie Mellon University, Information Technology Center, 1988.

[42] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung, *The Google file system*, vol. 37, ACM, 2003.

[43] Dhruba Borthakur et al., "Hdfs architecture guide," *Hadoop Apache Project*, vol. 53, 2008.

[44] Petar Maymounkov, "Online codes," Tech. Rep., Technical report, New York University, 2002.

[45] H. A. David and H. N. Nagaraja, *Order statistics*, John Wiley, Hoboken, N.J., 2003.