Rateless Sum-Recovery Codes For Distributed Non-Linear Computations

Ankur Mallick
Carnegie Mellon University
Pittsburgh, PA, USA
amallic1@andrew.cmu.edu

Gauri Joshi
Carnegie Mellon University
Pittsburgh, PA, USA
gaurij@andrew.cmu.edu

***Abstract*—We address the problem of slowdown caused by straggling nodes in distributed non-linear computations. Many common non-linear computations can be written as a sum of inexpensive non-linear functions (e.g. Taylor series). Based on this observation, we propose a new class of rateless codes called rateless sum-recovery codes whose aim is to *recover the sum of source symbols*, without necessarily recovering individual symbols. Source symbols correspond to individual inexpensive functions and each encoded symbol is the sum of a subset of source symbols. Encoded symbols are computed in a distributed fashion and for a computation that can be written as a sum of m inexpensive functions, successful sum-recovery is possible with high probability as long as slightly more than m encoded symbols are received. Our code is rateless, systematic and has sparse parities. Moreover, encoded symbols are constructed by sampling *without* replacement at individual nodes, thereby making decoding superfluous if the encoded symbols from any node cover all source symbols. We validate our claims through a range of simulations and also discuss open questions for future works.**

A full version of this paper is accessible at [1].

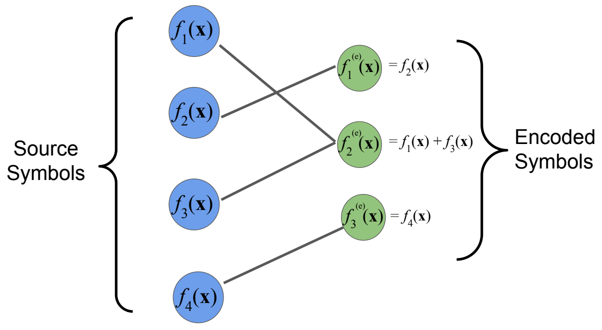
I. Introduction

Large-scale distributed computations are susceptible to delays caused by unreliable or slow computing nodes called *stragglers* [2]. In the past, adding redundancy via task replication was the prevalent approach for straggler mitigation [3], [4]. A recent line of work [5]–[11] (not an exhaustive list) has shown that adding redundancy to *linear* computations using erasure codes can provide superior straggler resilience. The canonical example of this is matrix-vector multiplication of the form $\mathbf{b} = \mathbf{A}\mathbf{x}$, $\mathbf{A} = [\mathbf{A}_1^T \ \mathbf{A}_2^T]^T$. If computed across three worker nodes this can be split into sub-computations $\mathbf{A}_1\mathbf{x}$ and $\mathbf{A}_2\mathbf{x}$, and a coded computation $(\mathbf{A}_1 + \mathbf{A}_2)\mathbf{x}$. Results from any two workers suffice to recover $\mathbf{A}\mathbf{x}$, and thus the system can tolerate one straggler. However, such linear coding schemes cannot be directly applied to the large class of *non-linear* computations in machine-learning [12] and scientific computing [13]. These computations often need to be performed in a distributed manner to obtain speedup, but they *cannot* be expressed as matrix-vector or matrix-matrix products.

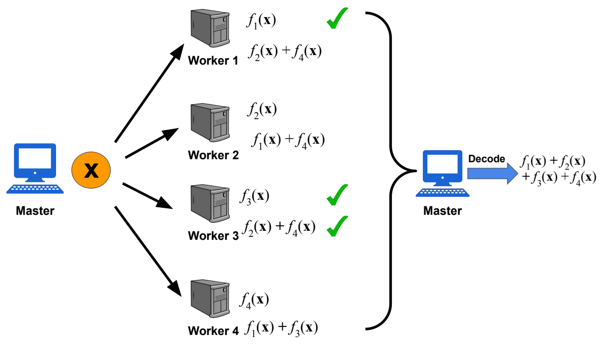
In this work, we consider distributed computation of expensive non-linear functions that can be decomposed into a sum of inexpensive non-linear functions, i.e., computations of the form $F(\mathbf{x}) = \sum_{i=1}^{m} f_i(\mathbf{x})$ where $f_1, f_2, \ldots, f_m$ are *non-linear* functions. *Any* differentiable non-linear function $F(\mathbf{x})$ can be approximated in this form with f_i's corresponding to the first m Taylor Series terms. Moreover, many popular machine learning computations like batch gradient descent [12], where the batch gradient is the sum of stochastic gradients, and kernel methods [14] where the kernel at a test point is the sum of kernels with respect to each training point, can also be written in this form. For large m, it may be desirable to compute the sum in a distributed fashion due to latency/memory constraints, thereby leading to the aforementioned problem of straggling.

Among the few works that have looked at non-linear coded computations, [15] only considers polynomial functions while [16] only approximately recovers the original computation and does not give any error bounds. The closest in spirit to our work is the line of work on gradient coding [17]–[21] which seek to recover the batch gradient from a set of coded stochastic gradients. However, they either need knowledge of the worst case number of stragglers for exact recovery which is often unavailable in practice, or provide approximate recovery where the error increases with magnitude of F which is undesirable for large magnitude F. Alternately, [22] performs sum-recovery by randomly sampling and computing disjoint partial sums of f_i's at workers with the master waiting for at least one copy of each partial sum. This can still be bottlenecked by straggling since workers only perform a *single computation* and thus the computation of F can be delayed if one or more f_i's are only sampled by slow nodes. In general, if the computations performed at individual workers do not suffice to recover the sum then straggling can be a bottleneck since fast workers may be idle while the master waits for the slow workers to complete their tasks. On the other hand, in [23] we mitigate straggling by using *rateless codes* to generate a (potentially infinite) stream of partial sums at workers such that *any* $m + o(m)$ computations from across all workers suffices to recover F. This balances the load by allowing fast workers to perform more work. However, directly applying existing rateless codes like LT codes [24] requires decoding each f_i before computing $\sum_{i=1}^{m} f_i(\mathbf{x})$ which may be superfluous if the goal is sum-recovery.

Our Solution. In this work, we introduce a *new class* of rateless codes called Rateless Sum-Recovery (SR) codes designed for distributed computing of the sum of non-linear functions under straggling (see Fig. 1). Being rateless, SR codes preserve the load balancing property of LT codes i.e. worker computations are proportional to their speeds and the sum can be exactly recovered with high probability from *any*


Source Symbols
$f_1(\mathbf{x})$
$f_2(\mathbf{x})$
$f_3(\mathbf{x})$
$f_4(\mathbf{x})$
$f_1^{(e)}(\mathbf{x}) = f_2(\mathbf{x})$
$f_2^{(e)}(\mathbf{x}) = f_1(\mathbf{x}) + f_3(\mathbf{x})$
$f_3^{(e)}(\mathbf{x}) = f_4(\mathbf{x})$
Encoded Symbols

(a) Encoding


Master
x
Worker 1 $f_1(\mathbf{x})$ ✓ $f_2(\mathbf{x}) + f_4(\mathbf{x})$
Worker 2 $f_2(\mathbf{x})$ $f_1(\mathbf{x}) + f_4(\mathbf{x})$
Worker 3 $f_3(\mathbf{x})$ ✓ $f_2(\mathbf{x}) + f_4(\mathbf{x})$ ✓
Worker 4 $f_4(\mathbf{x})$ $f_1(\mathbf{x}) + f_3(\mathbf{x})$
Master
Decode
$f_1(\mathbf{x}) + f_2(\mathbf{x}) + f_3(\mathbf{x}) + f_4(\mathbf{x})$

(b) Computation and Decoding

Figure 1: An illustration of SR-coded computing for a system of 4 workers tasked with computing $F(\mathbf{x}) = \sum_{i-1}^{4} f_i(\mathbf{x})$. Encoded symbols are sampled without replacement at each worker. Tick marks indicate completed tasks which suffice to recover $F(\mathbf{x})$. Observe that Worker 2 and 4 (stragglers) do not complete any tasks but the sum can be recovered.

$m + o(m)$ encoded symbols without prior knowledge of the number of stragglers. Additionally, unlike LT codes, in SR codes the encoded symbols at the same worker are constructed by sampling source symbols *without replacement* (sampling across workers is independent). Thus if the encoded symbols received from a worker cover all source symbols then the sum can be computed directly by just adding these encoded symbols. Lastly SR codes are systematic and the first encoded symbol at each worker corresponds to a single distinct f_i. Thus if the first computation of each worker is available then the sum can again be directly computed by addition. Thus, while decoding cost is unavoidable if using LT codes, there are several scenarios with SR codes where sum recovery is possible without decoding.

II. Preliminaries

A. Problem Setup

We consider computations of the form

$$F(\mathbf{x}) = \sum_{i=1}^{m} f_i(\mathbf{x}), \tag{1}$$

where if f_i's are non-linear then $F(\mathbf{x})$ is also non-linear. Such computations arise in machine learning during inference in ensemble models like Random Forests [25] and Deep Ensembles [26] where f_i is the i^{th} decision tree or neural network in the ensemble. Such computations can also be found in statistics [27] and physics [13] where f_i is the i^{th} term in the Taylor series approximation of a non-linear function. For this work, we consider that the individual f_i's are inexpensive to compute but the number of functions m is large, due to which computing (1) at a single node is expensive which motivates the need for distributed computation. While we assume that the computation of (1) is distributed over m worker nodes, this can be easily generalized to $p < m$ worker nodes by considering each group of m/p terms in (1) to be a single function. Assuming m workers gives cleaner analytical results and also matches emerging distributed computing frameworks like AWS Lambda [28], [29] where each worker typically computes a single function.

B. Baselines

Our proposed rateless coded scheme for distributed computation of (1) will be compared against the following baselines:

1) **All-at-One (One).** The entire computation is performed at a single node. The node receives $\mathbf{x}$ as input, computes $f_i(\mathbf{x})$, $i = 1, \ldots, m$ and then computes the sum $F(\mathbf{x})$.
2) **Uncoded (Unc).** The computation is distributed across m worker nodes (worker i computes $f_i(\mathbf{x})$). A central node (master) communicates $\mathbf{x}$ to workers, collects results of worker computations, and computes (1). The master needs to wait for *all* workers to complete their tasks.
3) **Fractional Repetition Codes (Rep) [17].** The computation is distributed across m workers but this time tasks are *replicated*. The group of workers $d(j-1)+1, \ldots, dj$ each compute the *partial sum* $\sum_{i=d(j-1)+1}^{dj} f_i(\mathbf{x})$ for $j = 1, \ldots, m/d$. For eg. if $d = 2$, workers 1 and 2 compute $f_1(\mathbf{x}) + f_2(\mathbf{x})$, workers 3 and 4 compute $f_3(\mathbf{x}) + f_4(\mathbf{x})$ etc. $F(\mathbf{x})$ can be computed once the master has collected all distinct partial sums. Thus the master only needs to wait for the *fastest* worker from each group.
4) **Batch Coupon Collector (BCC) [22].** The sum in (1) is again split into m/d disjoint partial sums as above (each partial sum consists of d distinct f_i's) but now each worker randomly samples and computes a single partial sum independent of the others (analogous to the coupon collector problem). The master needs to wait until it has received at least one copy of each partial sum.
5) **LT Coding (LT) [23].** LT codes [24] are applied to generate encoded symbols as sums of random subsets of source symbols (individual f_i's). A total of $m_e (m_e > m)$ encoded symbols are assigned to m workers such that each worker is assigned m_e/m symbols. The master sends $\mathbf{x}$ to the workers which computes encoded symbols. Decoding is possible once the master receives $m + o(m)$ encoded symbols from across *all* workers. Decoding involves applying the iterative peeling decoder [24] to the received encoded symbols to *recover each* $f_i(\mathbf{x})$, $i = 1, \ldots, m$ *and then computing their sum to obtain* $F(\mathbf{x})$.

C. Evaluation Criterion

We will use latency (defined below) as a metric to compare the aforementioned baselines and our approach.

Definition 1 (Latency (T))**.** *Latency T of a computing scheme is the time taken to collect a set of partial sums of the functions $f_i(\mathbf{x})$ from which the sum in* (1) *can be exactly recovered.*

III. Sum-Recovery Codes

We will now describe the encoding and decoding steps for our Rateless Sum-Recovery (SR) codes designed for distributed computing of (1). We will then highlight the advantages of our scheme over baselines. Individual $f_i(\mathbf{x})$'s are source symbols while encoded symbols are linear combinations of $f_i(\mathbf{x})$'s.

A. Encoding

We assume that information required to compute all f_i's (parameters, coefficients, etc.) can be accessed by *each* worker, even though each worker does not need to compute every f_i. Computation begins when a new input $\mathbf{x}$ is received for which $F(\mathbf{x})$ is required. Input $\mathbf{x}$ is communicated to all workers and each worker node computes encoded symbols sequentially (Fig. 1a) and transmits them to the central node (master).

The first encoded symbol computed at worker i is the systematic symbol $f_i(\mathbf{x})$. This ensures that each worker computes a *distinct* systematic symbol so that if there is little or no straggling and all the m systematic symbols are received at the master, then the sum in (1) can be directly computed.

Following their systematic symbol, workers start computing non-systematic encoded symbols, independent of the other workers. Each encoded symbol is communicated to the master before the worker computes the next encoded symbol. The j^{th} non-systematic symbol computed by worker i is given by

$$f_{ij}^{(e)}(\mathbf{x}) = \sum_{l \in \mathcal{S}_{ij}} f_l(\mathbf{x}) \tag{2}$$

where each S_{ij} is a set of d source symbols (excluding the systematic symbol) chosen uniformly at random *without replacement* i.e. $S_{ij} \cap S_{ij'} = \phi,\ \forall\ j, j'$, where ϕ is the null set. However, as workers choose their symbols independently there may be overlap between symbols chosen by different workers.

The degree d, that is, the number of source symbols in each encoded symbol, is a hyperparameter. Large d increases computation load at workers but a small d may increase the number of encoded symbols needed by the master to cover all source symbols (all source symbols need to be covered for (1) to be recoverable). Theorem 2 below shows that $d = \Omega(\log m)$ symbols are *necessary* to cover all source symbols while our simulations show that $d = \mathcal{O}(\log m)$ symbols are *sufficient* to recover (1) thus ensuring low encoding cost for our scheme.

In matrix form, the m_e encoded symbols can be represented by the $m_e \times m$ generator matrix $\mathbf{G_e} = [\mathbf{P}_1^\top, \ldots, \mathbf{P}_m^\top]^\top$ where $\mathbf{P}_i$ is the generator matrix at worker i. Thus the first row of each $\mathbf{P}_i$ (corresponding to systematic symbols) has 1 in position i and 0 at all other positions while row j $(j > 1)$ of each $\mathbf{P}_i$ (corresponding to parity symbols) has 1 at indices given by S_{ij}. The vector of encoded symbols is given by $\mathbf{f_e} = \mathbf{G_e}\mathbf{f}$ where $\mathbf{f} = [f_1(\mathbf{x}) \ldots f_m(\mathbf{x})]^T$.

B. Decoding

The master collects the encoded symbols until it has enough to be able to recover $F(\mathbf{x})$ (Fig. 1b). If all m systematic symbols are received or if all $1 + (m-1)/d$ encoded symbols (1 systematic symbol and $(m-1)/d$ parity symbols) from a particular worker are received then they can directly be added to compute $F(\mathbf{x})$. Otherwise, let $\mathbf{G}$ denote the $M' \times m$ consisting of the M' rows of the encoding matrix $\mathbf{G_e}$ that correspond to the encoded symbols received from across all workers, including systematic and parity symbols, and let $\mathbf{1}$ be the $m-$dimensional vector of all 1's. Decoding is possible if

$$\mathbf{G}^\top \mathbf{v} = \mathbf{1} \tag{3}$$

has a unique solution $\mathbf{v}_*$. If $\mathbf{f}'$ is the vector of received (encoded) symbols, then $\mathbf{v}_*^\top \mathbf{f}' = \mathbf{v}_*^\top \mathbf{G}\mathbf{f} = \mathbf{1}^\top \mathbf{f} = F(\mathbf{x})$,

A unique solution to (3) exists if rank$(\mathbf{G}^\top) =$ rank$([\mathbf{G}^\top|\mathbf{1}])$ [30], since this ensures that $\mathbf{1}$ lies in the column-span of $\mathbf{G}^\top$ ($[\mathbf{G}^\top|\mathbf{1}]$ is obtained by appending $\mathbf{1}$ to $\mathbf{G}^\top$ along its columns). In our analysis and simulations (see Section IV), we find that if d, the number of source symbols in each encoded symbol, satisfies $d = \mathcal{O}(\log m)$ and M', the number of received encoded symbols, satisfies $M' = m + o(m)$ then a unique solution of (3) exists with high probability. Thus, worker nodes compute encoded symbols in proportion to their speed (fast workers compute more symbols), and as along as we receive $m + o(m)$ encoded symbols *across all workers*, we can recover the sum with high probability. This rateless behavior differentiates our code from gradient codes where $\mathbf{G}$ is designed for a pre-determined number of straggling workers and specific requirements are imposed on the amount of computations required from different workers for sum recovery to be possible due to which the performance of the code is adversely affected if the actual straggling deviates from its expected behavior.

C. Advantages

The main aim of distributing an expensive computation like (1) is to reduce computation load at individual nodes, and obtain speedup due to parallelism while mitigating the effect of straggling nodes. In this regard our SR codes offer the following key benefits:

1) **Ratelessness:** With SR codes, recovering the sum in (1) is possible when either a) all $1 + (m-1)/d$ encoded symbols (1 systematic symbol and $(m-1)/d$ parity symbols) from a worker are received or b) $m + o(m)$ encoded symbols from across all workers are received. This balances the load across workers (fast workers perform more computations) leading to better straggler mitigation than fixed rate gradient coding approaches [17]–[22]. Moreover in case (a) when all encoded symbols from a worker are received, (1) can be directly recovered (by adding the symbols) without any decoding cost. On the other hand, with LT Codes [23], the decoding cost is always at least $\mathcal{O}(m \log m)$.

2) **Systematic Construction:** Each worker first computes and sends a *distinct* $f_i(\mathbf{x})$ before computing any linear combinations of f_i's. This makes the code systematic and ensures that there is little or no additional encoding/decoding cost if all f_i's are received directly as in cases where there is little straggling.
3) **Low Encoding Cost:** Each encoded symbol is a sparse linear combination of source symbols (we show in Section IV that $d \sim \mathcal{O}(\log m)$ is sufficient for sum-recovery). The sparse parity of the encoded symbols ensures *low computation load* at worker nodes since each encoded symbol is the sum of only a small number of source symbols. On the other hand, using a dense code like a maximum distance separable (MDS) code [5] would lead to a much higher computation load since each encoded symbol is a linear combination of *all* source symbols.

To the best of our knowledge, SR codes are the first class of codes which provide these advantages. Other variants of rateless codes can provide some but not all of the advantages. For e.g. Systematic LT Codes [31] have *dense parities* which leads to a high encoding cost, while Repairable Fountain Codes [32] sample encoded symbols with replacement due to which the decoding cost is unavoidable even if $m + o(m)$ encoded symbols are received from the same worker.

IV. Theoretical Analysis

In this section we derive an upper bound on the expected value of T_{SR}, the latency of the SR-Coded computing scheme. T_{SR} is the time required for the sum (1) to be recoverable from the received encoded symbols i.e. the time required to collect enough encoded symbols at the master such that (3) has a unique solution. Theorem 1 gives an upper bound on $\mathbb{E}[T_{\text{SR}}]$ when the expected number of encoded symbols collected at the master is $\mathbb{E}[M'] = m + o(m)$ symbols. Theorem 2 shows that $d = \Omega(\log m)$ are needed for $m + o(m)$ encoded symbols to cover *all* m source symbols with high probability. Finally, we show through simulations, if $d = \mathcal{O}(\log m)$ and $m + o(m)$ encoded symbols are collected, then a unique solution of (3) exists with high probability. The proof of all theoretical results and additional simulations are included in the Appendix of [1].

Delay Model. Our analysis of the latency of SR codes is based on the delay model of [23] wherein Worker i performs B_j computations in time Y_j where

$$Y_j = X_j + \tau B_j, \quad \text{for all } j = 1, \ldots, m \tag{4}$$

X_j is a random variable that includes initial random delays due to network latency setup time etc., while τ is a constant which represents the time taken by any worker to perform a single computation of the form $f_i(\mathbf{x})$. Observe that in SR-coded computing each parity symbol consists of d such computations. The model of an initial random delay followed by constant computation times is consistent with real-world observations [2]. When $X_j \sim \exp(\mu)$, the time taken by worker j to compute n_j encoded symbols of degree d is distributed as $\Pr(Y_j \leq t) = 1 - \exp(-\mu(t - \tau d n_j))$ illustrating that after the initial delay, worker latency is proportional to the number of computations.

Theorem 1 (Latency of SR Codes)**.** *For $m \to \infty$, and assuming that $\mathbb{E}[M'] = m + o(m)$ is the expected number of symbols needed for successful decoding, the expected latency for SR-coded computing of* (1) *with m workers and $X_i \sim \exp(\mu)$ for all workers $i = 1, \ldots, m$, is bounded as.*

$$\mathbb{E}[T_{SR}] \leq (2 + o(1))\tau d + \frac{1}{\mu} \tag{5}$$

where d is the degree of the SR code, or the number of source symbols added to obtain each encoded symbol.

Remark 1. This matches the upper bound for the latency of the LT-coded scheme in [23] if d is $\Omega(\log(m))$ i.e. of the same order as the degree of LT codes. The need for sparse encoded symbols (low encoding cost) can be clearly seen from the fact that the right-hand side of (5) is proportional to d. Thus, the expected latency can be significantly higher for dense codes.

Remark 2. In practice $\mathbb{E}[M']$, the expected number of symbols required for sum-recovery is much smaller than the expected number of symbols required for LT decoding [24] because the number of encoded symbols required for a solution of (3) to exist, is typically smaller than the number of encoded symbols required for the iterative peeling decoder of [24] to work. This is illustrated by our simulations below where SR codes outperform LT codes in a range of settings. Also, SR codes do not require decoding i.e. have zero decoding cost/delay if all systematic symbols are received, or if the entire set of encoded symbols from a worker node is received.

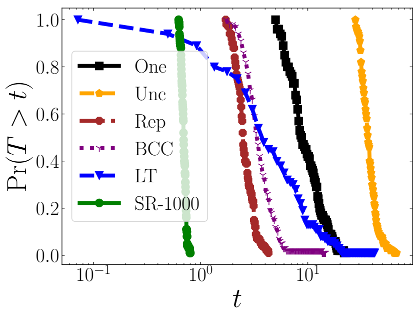
Theorem 2. *If each parity symbol is generated as described with d nonzero entries, then $d = \Omega(\log m)$ is needed for any set of $m + o(m)$ covers all source symbols with high probability.*

This is analogous to the Coupon-Collector (CC) problem and gives a *necessary* condition on the degree d, for sum-recovery with $m + o(m)$ SR-coded symbols. Simulations below show that $m + o(m)$ encoded symbols, with degree $d = \mathcal{O}(\log m)$, are also *sufficient* for sum-recovery with high probability.

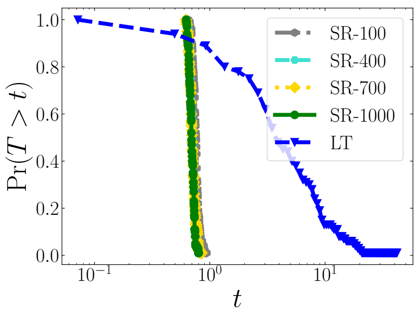
Simulations. We simulate baselines and SR-coded schemes under our delay model (4) with $m = 1000$, $\tau = 0.005$, $X_i \sim \exp(0.2)$, and $d = 10 \sim \mathcal{O}(\log m)$ (same average degree as LT codes). Simulations with Pareto X_i's in the Appendix of [1] show that SR codes are not limited to exponential X_i's.

Results in Fig. 2a clearly illustrate the superiority of SR Codes (lighter tail latency) over *all* baselines including prior works [17], [22], [23]. In rare cases the collected encoded symbols with LT Codes [23] might have a low average degree which gives a small probability of lower latency with LT coding (low degree encoded symbols are computed faster by workers). But this is clearly a very low probability event.

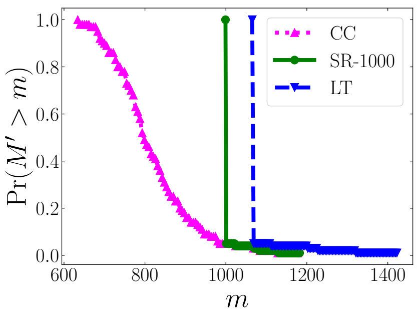
In Fig. 2a we use SR-1000 to denote that each worker has access to all source symbols (f_i's) when generating encoded symbols. Since this may be infeasible due to cost/privacy constraints, we also restrict worker access to limited source symbols (SR-k corresponds to each worker having access to $k \leq m$ random f_i's), and in Fig. 2b we see that even with $k = 100$, SR codes *significantly outperform* LT codes.



(a) SR Codes v/s baselines



(b) SR codes with limited source symbols



(c) Thresholds on Sum-recovery

Figure 2: (a) SR Codes have *significantly* lighter tail probability of latency than *all* baselines ($X_i \sim \exp(0.2)$). (b) Despite limiting the number of source symbols that each worker has access to (SR-k corresponds to each worker having access to $k \leq m$ source symbols), SR Codes continue to outperform LT Codes. (c) SR codes typically require fewer encoded symbols for sum-recovery than LT codes require for decoding. However, the number of encoded symbols required for sum-recovery is higher than the number of encoded symbols required to cover all source symbols akin to the Coupon-Collector (CC) problem. All tail probabilities are calculated using 100 Monte-Carlo simulations.

Finally we observe in Fig. 2c that the tail of M', the number of symbols required for sum-recovery in SR codes is significantly lighter than that for successful decoding in LT Codes. This is the primary reason for lower latency with SR codes that directly target sum-recovery as opposed to LT codes that seek to recover each source symbol.

V. Discussion

The analysis and simulations in the previous sections provide encouraging initial results for the proposed rateless Sum-Recovery (SR) codes. The proposed codes address the relatively unexplored problem of computing a non-linear function in a distributed, straggler-resilient fashion. Framing the problem as one of sum-recovery allows us to cover a large class of non-linear computations (Taylor Series, Random Forests, batch gradients, etc.) while structuring the problem in a manner that is amenable to coded computing. While gradient codes [17]–[22] also perform sum recovery, they were designed for the specific application of gradient descent. Therefore, in addition to the advantages outlined in Section III-C of the proposed SR codes over gradient codes, our work also opens up a new set of applications for gradient codes to distributed non-linear computations. We will now discuss three open questions that we believe can guide future work in this space and may also be of general interest to coding theory researchers.

The first open question is providing a sufficient condition on the number of encoded symbols, M', needed for sum-recovery. Currently, Theorem 2 provides a necessary condition on the degree d such that $M' = m + o(m)$ encoded symbols can cover all source symbols. However this does not, in theory, suffice to guarantee that (3) has a solution which is needed for guaranteeing sum-recovery. For e.g. if $m = 2$ and the rows of $\mathbf{G}$ are $[1\ 0]^T$ and $[1\ 1]^T$, all source symbols are covered but (3) does not have a solution. Our simulations show that $M' = m+o(m)$ rows span $\mathbf{1}$ with high probability, but proving this theoretically remains an open challenge.

The second question, related to the first question, concerns deriving a condition on the rows of matrices like $\mathbf{G}$ such that they span $\mathbf{1}$. This may be of general mathematical interest. While prior works [7], [32] derive such results when the matrix rows are obtained by sampling with replacement, in our case, the problem is challenging because rows coming from the same worker, are sampled *without replacement*.

The final question, which may be the most important for practical deployments, concerns the decoding process. Observe that currently the decoding process requires us to solve the $m-$dimensional system of equations (3). Since $\mathbf{G}$ is sparse this can be solved, in the best case, with complexity $\mathcal{O}(m^2 \operatorname{polylog} m)$ which is still significantly higher than the $\mathcal{O}(m \log m)$ decoding complexity of LT Codes [24]. In this work, we see that our codes have lower computation latency (as per Definition 1) than LT codes in simulations as fewer encoded symbols are needed to solve (3) than to recover each source symbol using the iterative peeling decoder used in LT codes [24]. However modifying SR codes so that the cost of solving (3) matches the decoding cost of LT codes is, we believe, a key pre-requisite for large-scale practical deployment.

VI. Conclusion

We introduce a new class of codes called Rateless Sum-Recovery (SR) codes to address the problem of distributed computation of a sum of *non-linear* functions under straggling. The codes are tailored to the problem setting and provide the advantages of ratelessness, systematic construction, and low encoding cost which enables our scheme to provide superior stragggler mitigation as compared to a range of baselines across several scenarios. We believe the proposed coding scheme will provide a framework for coded computing schemes for sum-recovery in settings like serverless computing [28]. Therefore we also include a discussion on open questions and future directions which can significantly expand the scope of coded computing to real world non-linear distributed computing tasks like distributed machine learning training and inference.

References

[1] "Full Version," http://www.andrew.cmu.edu/user/gaurij/ITW2022.pdf.

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

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

[4] 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.

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

[6] 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.

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

[8] Sanghamitra Dutta, Viveck Cadambe, and Pulkit Grover, ""short-dot": Computing large linear transforms distributedly using coded short dot products," *IEEE Transactions on Information Theory*, vol. 65, no. 10, pp. 6171–6193, 2019.

[9] Sanghamitra Dutta, Mohammad Fahim, Farzin Haddadpour, Haewon Jeong, Viveck Cadambe, and Pulkit Grover, "On the optimal recovery threshold of coded matrix multiplication," *IEEE Transactions on Information Theory*, vol. 66, no. 1, pp. 278–301, 2019.

[10] Ankur Mallick, Malhar Chaudhari, Utsav Sheth, Ganesh Palanikumar, and Gauri Joshi, "Rateless codes for near-perfect load balancing in distributed matrix-vector multiplication," *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, vol. 3, no. 3, pp. 1–40, 2019.

[11] Ankur Mallick and Gauri Joshi, "Rateless codes for distributed computations with sparse compressed matrices," in *2019 IEEE International Symposium on Information Theory (ISIT)*. IEEE, 2019, pp. 2793–2797.

[12] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Andrew Senior, Paul Tucker, Ke Yang, Quoc V Le, et al., "Large scale distributed deep networks," in *Advances in neural information processing systems*, 2012, pp. 1223–1231.

[13] Alan Newell, *Nonlinear optics*, CRC Press, 2018.

[14] Thomas Hofmann, Bernhard Schölkopf, and Alexander J Smola, "Kernel methods in machine learning," *The annals of statistics*, pp. 1171–1220, 2008.

[15] Qian Yu, Songze Li, Netanel Raviv, Seyed Mohammadreza Mousavi Kalan, Mahdi Soltanolkotabi, and Salman A Avestimehr, "Lagrange coded computing: Optimal design for resiliency, security, and privacy," in *The 22nd International Conference on Artificial Intelligence and Statistics*. PMLR, 2019, pp. 1215–1225.

[16] Jack Kosaian, K. V. Rashmi, and Shivaram Venkataraman, "Parity models: A general framework for coding-based resilience in ML inference," *CoRR*, vol. abs/1905.00863, 2019.

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

[18] Zachary Charles, Dimitris Papailiopoulos, and Jordan Ellenberg, "Approximate gradient coding via sparse random graphs," *arXiv preprint arXiv:1711.06771*, 2017.

[19] Netanel Raviv, Rashish Tandon, Alex Dimakis, and Itzhak Tamo, "Gradient coding from cyclic mds codes and expander graphs," in *International Conference on Machine Learning*. PMLR, 2018, pp. 4305–4313.

[20] Hongyi Wang, Zachary Charles, and Dimitris Papailiopoulos, "Erasurehead: Distributed gradient descent without delays using approximate gradient coding," *arXiv preprint arXiv:1901.09671*, 2019.

[21] Sinong Wang, Jiashang Liu, and Ness Shroff, "Fundamental limits of approximate gradient coding," *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, vol. 3, no. 3, pp. 1–22, 2019.

[22] Songze Li, Seyed Mohammadreza Mousavi Kalan, A Salman Avestimehr, and Mahdi Soltanolkotabi, "Near-optimal straggler mitigation for distributed gradient methods," in *2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. IEEE, 2018, pp. 857–866.

[23] Ankur Mallick, Sophie Smith, and Gauri Joshi, "Rateless codes for distributed non-linear computations," in *2021 11th International Symposium on Topics in Coding (ISTC)*. IEEE, pp. 1–5.

[24] Michael Luby, "LT codes," in *Proceedings of the 43rd Annual IEEE Symposium on Foundations of Computer Science*, 2002, pp. 271–271.

[25] Leo Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[26] Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell, "Simple and scalable predictive uncertainty estimation using deep ensembles," *Advances in neural information processing systems*, vol. 30, 2017.

[27] Edward F Vonesh, Hao Wang, and Dibyen Majumdar, "Generalized least squares, taylor series linearization and fisher's scoring in multivariate nonlinear regression," *Journal of the American Statistical Association*, vol. 96, no. 453, pp. 282–291, 2001.

[28] Eric Jonas, Johann Schleier-Smith, Vikram Sreekanti, Chia-Che Tsai, Anurag Khandelwal, Qifan Pu, Vaishaal Shankar, Joao Carreira, Karl Krauth, Neeraja Yadwadkar, et al., "Cloud programming simplified: A berkeley view on serverless computing," *arXiv preprint arXiv:1902.03383*, 2019.

[29] Amazon, "AWS Lambda," https://aws.amazon.com/lambda/, 2014.

[30] Igor R Shafarevich and Alexey O Remizov, *Linear algebra and geometry*, Springer Science & Business Media, 2012.

[31] Amin Shokrollahi, "Raptor codes," *IEEE/ACM Transactions on Networking (TON)*, vol. 14, no. SI, pp. 2551–2567, 2006.

[32] Megasthenis Asteris and Alexandros G Dimakis, "Repairable fountain codes," *IEEE Journal on Selected Areas in Communications*, vol. 32, no. 5, pp. 1037–1047, 2014.