Straggler Mitigation in Edge Computing with Coded Compressed Sensing

Yangyang Tao
Department of Computer Science
Northern Kentucky University
Highland Heights, KY, USA
ytao13@stevens.edu

Junxiu Zhou
Department of Computer Science
Northern Kentucky University
Highland Heights, KY, USA
zhouj2@nku.edu

***Abstract*—In edge computing the performance of distributed edge computation tasks are adversely impacted by stragglers (i.e., slowest devices). Previous research addresses this problem using coding techniques to bypass the dependence on stragglers. In stead of completely discarding partially unfinished coded computations on stragglers, recent research incorporates those computations contributed by stragglers before the deadline. A faster computation recovery is achieved. One problem with this approach, however, is the recovery accuracy because it is based on lossy quantization over coded data. In this paper, we treat the partially unfinished coded computation as erroneous computations and formulate the computation recovery problem as a compressed sensing (CS) problem. With this rateless approximate code approach, we can recover the erroneous computations with a high accuracy rate when the ratio of stragglers is relatively low. Experimental results show that we reduce the error rate by average of 32% under various straggler ratios compare with the state of the art.**

***Index Terms*—Edge Computing, Parallel Machine Learning, Federated Learning, Compressed Sensing, Rateless Coding**

I. Introduction

Edge computing is a promising distributed computing paradigm for data analytic applications in 5G and future generation networks. In edge computing, computing devices (i.e., edge devices) are brought closer to sources of data (bring computation to data), e.g., Internet of Things (IoT), which results in reduced response time and bandwidth efficiency [1]. Due to device/network heterogeneity and byzantine failures, the response delay on different edge devices may vary significantly [2]. Some may be much slower than others and become stragglers (and hence bottlenecks) in distributed computation tasks.

To mitigate the adverse impacts of stragglers, one approach is to use coding techniques and disperse coded data at each edge device. For linear operations as seen in distributed machine learning and matrix multiplication, individual edge devices can execute the operations on coded data and send their local results to the server (a.k.a. the master controller). The latter will decode the locally computed results to recover the actual results. In case stragglers are not able to return their results timely, the master controller will collect data from other devices for the recovery. To this end, various coding techniques, such as the maximum distance separable (MDS) code [3] and the rateless code [4], have been investigated.

One outstanding problem with these approaches is that they completely discard the partial incomplete results collected from the stragglers [3], or introduce additional communication and computational costs to recover the complete results from the stragglers ([4]). To fully utilize partial incomplete results of the stragglers, Kim et. al [5] recently proposed to use binary coding approach to approximate the missing local results of the stragglers using both non-stragglers' results and the partial results of the stragglers. As compared to previous research Kim et. al [5] result in a better chance of recovery even if there are not sufficient devices in the coding scheme (e.g., less than k devices in (n, k)-coding schemes. The proposed scheme is also more robust under network topology dynamics)(e.g., due to device mobility).

Kim et. al [5], however, also exhibits several limitations as follows. First, this approach relies on lossy quantization over coded data, which results in decoding errors with the recovery. Second, it requires to exchange additional state information between edge devices and the master controller. Last but not the least, a brute force based approximation is needed to encode and decode the data. These problems negatively impacted the recovery accuracy, bandwidth and computation efficiency.

In this work, we take a different approach and propose a rateless approximate code method to address the straggler issue. Specifically, we treat the intermediate coded computation results from stragglers as erroneous computations. As long as the relatively number of stragglers in the coded edge computing system is small, the error vector over all edge nodes has the sparse property. The erroneous coded computation of the straggler can therefore be formulated as a compressed sensing (CS) problem. By carefully designing the generator matrix under certain constraints during the coding process, the orthogonal matching pursuit (OMP) algorithm can be used to solve the CS problem and the error vector can be exactly recovered. Subsequently, the complete computation results of a straggler can be recovered by subtracting the recovered error from the obtained erroneous computation. Experimental results show that we reduce the error rate by average of 32% under various straggler ratios (from 10% - 90%).

In summary, the proposed rateless approximate code design has the following contributions:

979-8-3503-7099-7/24/$31.00 ©2024 IEEE

(1) To our best knowledge, this is the first work that uses the CS technology to approximate the error caused by stragglers in a coded edge computing;

(2) Unlike most coding-based methods that ignore the erroneously coded computation results generated by stragglers, in this work, we utilize them to recover the error vector.

(3) The proposed approach can achieve a lower error rate and even exact recovery under CS constraints as compared to the state of the art. Unlike previous work, our error recovery process does not need to store state information, such as the position of the error parts in erroneous computation. As a result, the CS process can be efficiently solved with linear complexity in terms of the total number of stragglers and coding parameter.

(4) Theoretical analysis and experimental results show that under given constraints, the proposed method outperforms state-of-the-art work by an average of 32% in terms of recovery error reduction.

The rest of this paper is organized as follows. Section II overviews related work. Section III presents the system model and our design. We present experimental results to evaluate the proposed design in Section IV, which is followed by the conclusion in Section V.

II. Related Work

A. Coded Edge Computing

Edge computing is a distributed computing paradigm has recently drawn extensive attention. Due to heterogeneity of devices and network conditions, the computational power and response time of edge devices vary and some may be much slower than others for distributed computation tasks. To address the straggler problem, many techniques have been proposed in the literature. One promising approach is to apply coding techniques and disperse coded data to edge devices. Linear operations can be directly computed on coded data by edge devices. The master controller collects local coded results and decode to recover the actual results. To address the problem of stragglers, various techniques have been proposed in the literature.

In ref. [3], Lee et. al proposed to use MDS code to resolve the straggler issue. This approach codes a training data matrix for distributed learning using the MDS code and hence the coded computation from any subset of the edge nodes can be used to recover the original computation. However, this approach ignored the computation obtained from stragglers. Mallick and Joshi [4] resorted to rateless code to encode the training data matrix. As the result, the original computation can be recovered with a slight overhead of coded computation from edge node.

Recently, Kim et al. [5] proposed a binary coding approach for distributed edge computing. This approach encodes the original data with a two-step coding process. First, it maps the data into binary coding through quantization; then, the binary coded data is encoded through a binary generator matrix. With this binary coding process, the edge nodes are able to finish their computation process after receiving a number of rows of the coded data from the master controller. Then the master collects the coded computation results from edge nodes and estimates the original computation by employing the maximum likelihood method. In this work the results from stragglers will be collected at the master controller and treated as the computation with errors, which will be used to estimate the expected results from the stragglers. Inspired by [5], this paper uses the CS technology to recover the erroneous coded computing on stragglers. Based on the recovered error vector, we will eliminate the errors from the coded computation result. The proposed approach overcomes several limitations of [5] with a reduced recovery complexity and a higher recovery accuracy under given constraints.

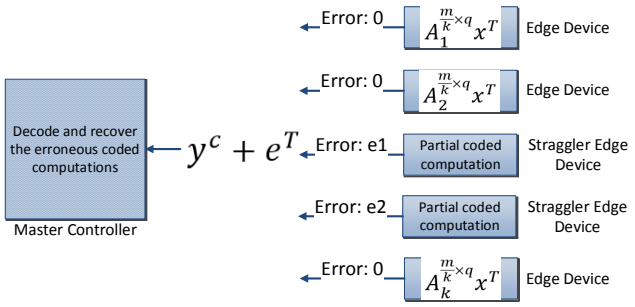


Fig. 1. Edge computation system model with coded large-scale matrix vector multiplication.

B. Compressed Sensing

Compressed sensing (CS) is a fundamental methodology in signal processing discipline, which has gained extensive interests in various areas [6]. Under the sparsity and coherence conditions, CS can solve an under-determined linear system and efficiently acquire and reconstruct a signal. The under-determined linear system can be represented by $\mathbf{b} = \boldsymbol{A}\mathbf{x}^T$. Matrix $\boldsymbol{A}$ has more columns than rows, so this linear system has no unique solution $\mathbf{x}$. Of particular interest is to find $\mathbf{x}$ which is a sparse representation of $\mathbf{b}$, via ℓ_1 norm minimization. Since the representation is nondeterministic, if there exists a constant ϵ and the linear equation meets restriction $||\mathbf{b} - \boldsymbol{A}\mathbf{x}^T|| < \epsilon$, the solution is called the sparse approximation of $\mathbf{b}$. In this representation, $\mathbf{b}$ is no long required to be in $span(A)$. There are numerous well-known algorithms (basis pursuit, matching pursuit and Stage wise greedy approach) [7] that can be used to solve this approximation problem.

III. System Model and Our Design

A. System Model

We take a classic distributed coded edge computating scenario into consideration as shown in Fig. 1, i.e., a star edge computing network architecture with a single master and n edge nodes. To simplify the discussion, we consider the computation task to be a general problem of coded large-scale

matrix vector multiplication ($\boldsymbol{y}^c = \boldsymbol{G}\boldsymbol{A}\boldsymbol{x}^T$). Distributed large matrix vector multiplication is the fundamental component of many scientific analysis and data analytic applications, such as solving partial differential linear programming systems [8], neural network process [9], graph computation [10] and distributed optimization [11]. For the main task of large-scale matrix-vector multiplication, the matrix is coded and distributed to edge nodes along with the vector by the master. Multiplication is conducted on edge nodes over coded matrix before the result is aggregated from edge nodes and sent to master for decoding. The master node works as the fusion node to distribute the coded partial training data matrix $\mathbf{G}\boldsymbol{A}$ and weight vector $\mathbf{x}^T$ to edge nodes. Each edge node computes the partial matrix vector multiplication and returns the result to master. Considering a systematic rateless coding approach (the original training data matrix will be retained in the coded training data matrix), the generator matrix $\boldsymbol{G}$ for the systematic rateless coding is of form $\mathbf{G} = [\frac{\mathbf{I}}{\mathbf{P}}]$. $\mathbf{P}$ is the parity part which determines redundancy blocks of the coded matrix and $\mathbf{I}$ is the identity matrix. The original computation is $\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}^T$. The ultimate goal is to recover this original computation through the decoding process. Decoding starts when the total of received partial coded computation from edge nodes reaches the required amount (for rateless coding, the required amount of coded computation results is slightly large than the original computation). The decoding can be represented by $\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}^T = \boldsymbol{G}^{-1}\boldsymbol{y}^c$. However, some of the coded computation may contain errors due to the instability of edge nodes for the deadline sensitive applications. So the received coded computation can be represented by $\mathbf{y}^{'} = \boldsymbol{y}^c + \mathbf{e}^T$, $\mathbf{e}^T$ is the error vector. The entry of error vector is 0 if the edge node finishes the coded computation within the given deadline; otherwise, the entry denotes the error due to the unfinished coded computation. The number of none-zero entries of the error vector is determined by the number of stragglers. Different from traditional MDS code that discard the erroneous intermediate results and continue to wait for the rest of edge nodes to return the results, the proposed rateless approximate coding design utilizes these partially completed or erroneous intermediate results.

B. Rateless Approximate Code

The proposed rateless approximate code design for a distributed large matrix vector multiplication problem is illustrated in this section. Given a training data matrix $\boldsymbol{A}^{m\times q}$ ($m < q$, m and q are the row and column size of $\boldsymbol{A}$) and the weight vector $\boldsymbol{x}^q$, we first divide the matrix into k row splits as $[\boldsymbol{A}_j^{\frac{m}{k}\times q}], j \in [1,k]$. $\boldsymbol{G}^{n\times k} = [\frac{\mathbf{I}_k}{\mathbf{P}^{n-k\times k}}]$ is the generator matrix over a field F_λ for the coding design, $\lambda = 2$. Therefore, the encoded matrix data is $\boldsymbol{A}^{'} = \boldsymbol{G}^{n\times k}[\boldsymbol{A}_j], j \in [1,k]$, $\boldsymbol{A}_j$ has dimension $\frac{m}{k} \times q$. The master node distributes the encoded matrix data and the weight vector to n edge nodes. Each of the edge nodes is responsible for the partial computation of $[\boldsymbol{A}_i^{'}]^{\frac{m}{k}\times q}\boldsymbol{x}^T, i \in [1,n]$, over encoded matrix data it received. The overall coded computation received from edge nodes can be represented as $\boldsymbol{y}^{'} = \boldsymbol{A}^{'}\boldsymbol{x}^T + \mathbf{e}^T$, where $\boldsymbol{y}^{'}$ has a dimension of $m \times 1$. The encoding and computation process is shown as below:

$$\begin{aligned}\mathbf{y} &= \boldsymbol{A}\mathbf{x}^T\\ \mathbf{y}^c &= \mathbf{A}^{'}\mathbf{x}^T\\ \mathbf{y}^{'} &= \mathbf{y}^c + \mathbf{e}^T\end{aligned}$$

The ultimate goal is to recover the original computation $\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}^{\boldsymbol{T}}$ based on coded partial results returned by edge nodes. In order to decode the coded computation, rateless code design requires the coded partial computations are from at least $k+\delta$ edge nodes, where δ is a small number. Under this scheme, the first step is to recover the error vector $\mathbf{e}^T$ in the coded computation $\mathbf{y}^{'}$ before decoding. Given the number of stragglers $s << k+\delta$, the error vector is actually a sparse vector because the number of none zero entries of the error vector is determined by s. So $\mathbf{e}^T$ is a sparse vector. In order to construct a CS problem for $\mathbf{e}^T$, we introduce the parity check matrix of $\mathbf{G}$ as $\mathbf{H}^{n-k\times n} = [-\mathbf{P}^{n-k\times k}|\mathbf{I}_{n-k}]$. The detailed construction of the CS problem is shown below:

$$\begin{aligned}\tilde{\mathbf{y}}^{'} &= \mathbf{H}\mathbf{y}^{'}\\ &= \mathbf{H}(\mathbf{y}^c + \mathbf{e}^T)\\ &= \mathbf{H}\mathbf{y}^c + \mathbf{H}\mathbf{e}^T\\ &= \mathbf{H}\mathbf{G}\boldsymbol{A}\mathbf{x}^T + \mathbf{H}\mathbf{e}^T\end{aligned}$$

$$\mathbf{H}\mathbf{G}\mathbf{A} = \mathbf{0}^{(n-k)\times 1} \tag{1}$$

$$\tilde{\mathbf{y}}^{'} = \mathbf{H}\mathbf{e}^T \tag{2}$$

Eq. (1) is from the property of the parity check matrix and Eq. (2) is the compact form of the CS. In this CS problem, we are not concerned about which part of the $\mathbf{e}^T$ is none zero since $\mathbf{e}^T$ is sparse. We can recover an optimal $\mathbf{e}^*$ as the error vector. It is easy to show that this CS problem could be conveniently solved with the Orthogonal Matching Pursuit (OMP) algorithm. After we get the error vector, the coded computation can be recovered by $\mathbf{y}^c = \mathbf{y}^{'} - \mathbf{e}^*$. Finally, the decode process $\boldsymbol{G}^{-1}\boldsymbol{y}^c$ recovers the original computation of $\boldsymbol{A}\mathbf{x}^{\mathbf{T}}$.

C. Generator Matrix Construction

In this section, we give the prerequisites for the CS problem and propose an algorithm for constructing the generator matrix under the given prerequisites. In the CS problem constructed in previous section, we treat the parity check matrix $\boldsymbol{H}$ as the sensing matrix and $\mathbf{e}^T$ as the sparse coefficients. The relationship between coherence properties of $\boldsymbol{H}$ and the number of stragglers is defined in Theorem III.1.

Theorem III.1. *Given the error vector e^T, the number of stragglers s, μ as the coherence of $\boldsymbol{H}$ and $\boldsymbol{H} = [-\mathbf{P}|\mathbf{I}]$, we have the below inequality:*

$$||e||_0 = s\frac{m}{k} < \frac{1}{2}(\mu^{-1} + 1) \tag{3}$$

where

$$\mu = \max_{i \neq j} \frac{| < (p_i, p_j) > |}{||p_i||_2 ||p_j||_2} \tag{4}$$

Proof. The number of error entries in error vector e^T are determined by the number of stragglers s. For a deadline sensitive application, the error only occurs when the edge nodes fail to finish the computation on time. Then the returned computation contains errors at unfinished locations of the computation. For each edge node, it needs to process $\frac{m}{k}$ rows of $\mathbf{A}$. So e^T has a sparsity of $s\frac{m}{k}$. According to the definition of rateless code, we have $s\frac{m}{k} << (k+\delta)\frac{m}{k} \approx m$. Thus e^T is a sparse vector. Recall the definition of CS problem, we have the inequality in Eq. (3). Since $\boldsymbol{H}$ = $[-\mathbf{P}|\mathbf{I}]$, the coherence of columns related to the parity matrix $\mathbf{P}$ of H. Then the coherence μ can be represented in Eq. (4). The bound of μ is proved to be $[\frac{k}{(n-k)(n-1)}, 1]$ [12]. □

Under Theorem III.1, we construct the generator matrix G accordingly to ensure the coherence of parity check matrix H of G fulfills the desired constraints of Eq. (3). The algorithm is given in Algorithm 1.

Algorithm 1 *Generator Matrix Construction*

procedure
Initialization:
    Initialize $\boldsymbol{P}$ parity part from sub Gaussian distribution
    Initialize s, m, k
    Initialize μ from Eq. (4) coherence of $\boldsymbol{P}$
loop:
    while Eq. (3) is False **do**
        Draw value for column p_i and p_j
        from sub Gaussian distribution
        μ := Eq. (4)
    end while
end loop
    Return $\mathbf{G}$ = $[\frac{\mathbf{I}}{\mathbf{P}}]$, $\mathbf{H}$ = $[-\mathbf{P}|\mathbf{I}]$
end procedure

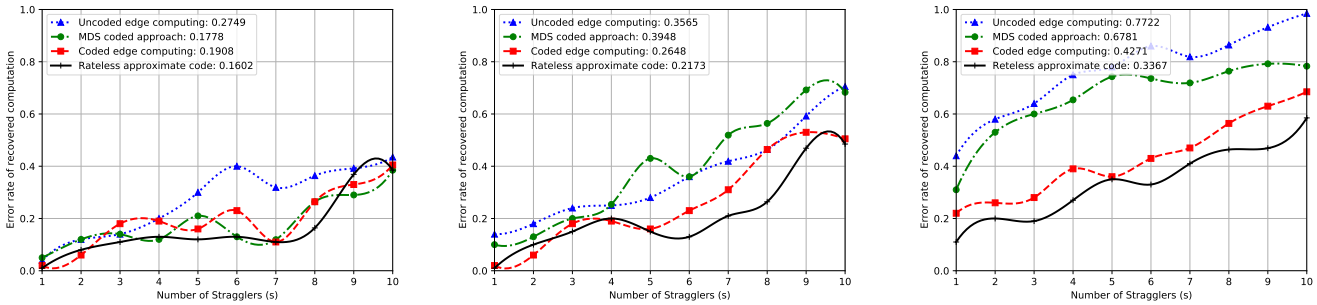
IV. Experimental Results

To illustrate the effectiveness of the proposed rateless approximate coding method, we conducted extensive experiments on the test-bed environment CloudLab [13]. We used a total of 20 nodes in CloudLab with 1 as the master and 19 as the edge nodes. The configuration is identical for all the edge nodes in terms of CPU, memory and storage. However, we manually introduced heterogeneity to the edge nodes by defining a *straggler probability*. The straggler probability indicates the probability that the specific edge node cannot finish the task within the deadline. It simulates the instability of the edge nodes in real environments. The implementation is based on the Message Passing Interface for python (MPI4py) library [14]. The benefit of MPI4py is program once and run on every edge node which reduced the effort to program the logic for each node. We used the synchronous communication scheme of MPI4py to program the tasks for our experiment. We have also implemented a MDS coded scheme [3] and coded edge computing [5] for comparison purpose.

The performance of the proposed rateless approximate code is evaluated with three other approaches: 1) uncoded edge computing, 2) MDS code based edge computing and 3) coded edge computing [5]. The uncoded edge computing approach doesn't have any augment techniques to handle the straggler issue. MDS code approach first divides the original data into k row splits and then encodes those row splits into n coded data shards. $n-k$ of the data shards are redundancy. Here the coding parameter (n, k) is the same for coded edge computing and the proposed rateless approximate code approach. The size of the quantization bit of coded edge computing is chosen according to the max value of synthesized training data matrix. For the three coded approaches, the generator matrix is binary matrix contains only 0 and 1. The design follows a systematic coding scheme which contains a complete copy of original data in coded data shards. For the proposed rateless approximate code, the generator matrix is generated with Algorithm 1.

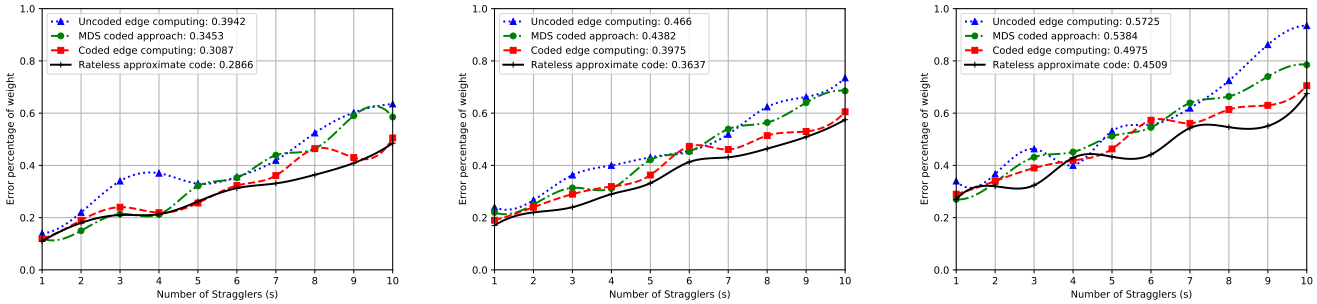
The tasks we chose for the experiments are large distributed matrix vector multiplication and federated multi-task learning (FMTL) [15] in a deadline sensitive mode. There is a preset deadline for both tasks. When the computation time reaches the deadline, the edge nodes are required to return current finished computation to master. For large distributed matrix vector multiplication, a synthesized training data matrix and a weight vector are given as $\boldsymbol{A}$ and $\boldsymbol{x}$ respectively. The (training) data matrix is sampled from a given distribution. In the experiments, the dimension of A is set as 100000×10 and the length of weight vector is selected as 10. Since we have total 19 edge nodes, the number of row splits k is in the range of $[2, 18]$. The purpose is that there should be at least 2 row splits and at least 1 redundancy. $n = 19$ is the total number of edge nodes. This coding parameters (n, k) holds for all three coding approaches in our experiment. The number of stragglers is the critical factor for the experiments in our design. We vary the number of stragglers from 1 to 10 in our experiments to evaluate the error rate of the recovered computation from the edge nodes. The error rate is calculated as the ratio of the total incorrectly recovered computation to the total computation. The results are shown in Fig. 2.

As shown in Fig. 2, the straggler probability is 0.6, which indicates among the given number of straggler edge nodes, there are 60% of chance that it cannot finish the computation within the deadline. We experimented on three values of coding parameter k: 10, 15, 18. The rest of the configurations are all identical. For each straggler number, the error rate (percentage of error) of recovered computation is averaged from multiple runs of experiments. As shown in Fig. 2a (k=10), with the increase of the number of stragglers, the error rate of uncoded edge computing is the highest. This is because the uncoded approach is the most vulnerable to the impact of stragglers. The MDS coded approach and the coded edge computing approach have similar error rates. However,



(a) Error rate with coding parameter k = 10 (b) Error rate with coding parameter k = 15 (c) Error rate with coding parameter k = 18

Fig. 2. Distributed matrix vector multiplication with given coding parameter k = 10, 15, and 18 and straggler probability = 0.6. The rest parameters are given by default.



(a) Error rate with coding parameter k = 10 (b) Error rate with coding parameter k = 15 (c) Error rate with coding parameter k = 18

Fig. 3. FMTL with given coding parameter k = 10, 15, and 18 and straggler probability = 0.7. The rest parameters are given by default.

the error rate of the MDS coded approach starts to surge when the number of stragglers exceeds 9. The reason is that the redundancy of the coding scheme is $n - k = 9$. When the number of stragglers exceeds 9, the MDS coded approach has 100% chance to receive the computation results from stragglers. The coded edge computing has three quantization processes during the computation. So it suffers from the loss of accuracy in the decoding process. The error rate of coded edge computing is also higher than the proposed approach. Our rateless approximate approach has the lowest average error rate because it can recover exactly the errors in the coded computation. However, when the amount of stragglers becomes large, our approach also suffers from the surge of error. This is because the number of stragglers approaches the restriction of the maximum coherence of parity check matrix H.

We also compared the error rate between the 4 models with 5 different straggler probabilities (10% , 30% , 50% , 70% ,and 90%). The results are shown in Table I. We fixed the coding parameter as k = 10 and straggler number as s = 9. The error rate is the average from multiple runs of experiments. The highlighted error rate is the optimal one among the 4 models for the given probability. For all of the 4 models, the error rates increase as the straggler probability increases. But the overall average error rate is not as high as Fig. 2 for a high straggler probability. The reason is that the number of stragglers is fixed to 9 which equals to the number of redundancy in the coding scheme. For our proposed rateless approximate code approach, the error rate is lower than other 3 models in almost every straggler probability scenario (the only higher case is when the straggler rate is 30%, the reason is due to the coding parameter for th other two approach is exactly the number of the redundancy $n - k$, however, in real environment this is not always the case). Therefore, our proposed approach has the best performance regarding the error rate of the recovered computation.

TABLE I. Average error rate of recovered computation for 4 models regarding different straggler probability. The coding parameter k = 10 and straggler number s = 9.

Model/Straggler Probability	10%	30%	50%	70%	90%
Uncoded edge computing	0.19	0.24	0.28	0.36	0.48
MDS coded edge computing [3]	0.12	0.15	0.21	0.23	0.28
Coded edge computing [5]	0.13	**0.14**	0.24	0.22	0.24
Rateless approximate code	**0.08**	0.16	**0.18**	**0.20**	**0.19**

Next, in order to evaluate the proposed solution with a real edge learning algorithm, we incorporate the MDS code

approach, coded edge computing and rateless approximate code to FMTL and conducted the experiment on it. FMTL is proposed to solve the non-linear distributed machine learning problem. It captures the non-linear relationship among each local training. The FMTL problem can be represent as follow:

$$\min_{\boldsymbol{X},\boldsymbol{\Omega}} \{\sum_{i=1}^{n} \frac{1}{ts} \sum_{t=1}^{ts} \ell_i(-\alpha_t) + \frac{1}{2\lambda} \boldsymbol{\alpha}^T \boldsymbol{K} \boldsymbol{\alpha}\} \tag{5}$$

In the formula, $\boldsymbol{X}$ is the weight matrix and each column corresponds to the weight vector of edge node i. $\boldsymbol{\Omega}$ is the model weight relationship calculated from the weight vector of each edge node. ts is the total number of training samples on each edge node. ℓ_i is the loss function according to the given machine learning model. α_t is the dual variable of the ℓ_i. $\boldsymbol{K}$ is the kernel matrix. λ is a constant. The ultimate goal is to update the weight vector of each edge node. FMTL first solves the minimization problem of Eq. (5). Then the weight is updated by $\boldsymbol{X}$, α_t, and $\boldsymbol{A}$.

FMTL runs in a federated format. However, the MDS code approach, coded edge computing approach, and proposed rateless approximate code approach all contain a coded data distribution process. In the implementation, we conduct this process preferentially and distributed the coded weight vector to each edge node. The edge node then conducts the computation for the weight of local model with a given convex loss function and uploads the local weight to the master node. The master node will decode the computation accordingly and update the relationship $\boldsymbol{\Omega}$. For this experiment, we define the error rate as the average weight difference between the 4 models and the vanilla FMTL without stragglers. The weight difference is the percentage of differed element in the weight matrix. The results are presented in Fig. 3 which shows the error percentage of the weight matrix.

As shown in Fig. 3, the overall error rate is higher as compared with the previous task. As in FMTL, the weights are calculated from the weight relationship $\boldsymbol{\Omega}$ of each edge node and the weight relationship $\boldsymbol{\Omega}$ is computed from all the weight vectors. So the error will propagate to more positions during the computation. We used a similar configuration of coding parameter k (10, 15, and 18) and straggler probability 0.7 in this experiment. Other configuration are all identical for all the 4 models. The uncoded approach is the vanilla FMTL but with stragglers. It has the highest error with the increase of the number of stragglers. The performance of the MDS coded approach is similar to the uncoded approach as it only has the decoding process and doesn't consider the recovery of error. For the coded edge computing approach and the proposed rateless approximate coding approach, the error rates are lower in general since both of them recover the error part in the computation. The proposed rateless approximate coding approach has the best performance among all. However, due to the constraints in Theorem III.1, when the number of stragglers is large, our proposed rateless approximate coding approach will also suffer from a high error rate though it still behaves better than other three models.

V. Conclusion

This paper focuses on the straggler problem in the edge computing with deadline sensitive applications. How to reduce the impact from the stragglers is a key challenge that hinder the development of edge computing. One track of solution which is coded edge computing tries to use coding techniques (MDS coding, rateless coding, and binary coding) to resolve the problem. Inspired by previous works, this paper proposes rateless approximate coding, which combines the rateless coding with CS to reconstruct the errors in unfinished computations from stragglers. It first construct the CS through the generator matrix design with given constraints, then solve the CS problem to recover the error. The recovered error is subtracted from coded computation to recover the original computation. Through the experiment on two typical tasks, we prove the proposed rateless approximate coding achieves the lowest recovery error compare with other state of arts.

References

[1] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE internet of things journal*, vol. 3, no. 5, pp. 637–646, 2016.

[2] Y. Shi, K. Yang, T. Jiang, J. Zhang, and K. B. Letaief, "Communication-efficient edge ai: Algorithms and systems," *arXiv preprint arXiv:2002.09668*, 2020.

[3] K. Lee, M. Lam, R. Pedarsani, D. Papailiopoulos, and K. Ramchandran, "Speeding up distributed machine learning using codes," *IEEE Transactions on Information Theory*, vol. 64, no. 3, pp. 1514–1529, 2017.

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

[5] K. T. Kim, C. Joe-Wong, and M. Chiang, "Coded edge computing," in *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*. IEEE, 2020, pp. 237–246.

[6] Y. C. Eldar and G. Kutyniok, *Compressed sensing: theory and applications*. Cambridge university press, 2012.

[7] J. A. Tropp, "Greed is good: Algorithmic results for sparse approximation," *IEEE Transactions on Information theory*, vol. 50, no. 10, pp. 2231–2242, 2004.

[8] W. F. Ames, *Numerical methods for partial differential equations*. Academic press, 2014.

[9] W. Dally, "High-performance hardware for machine learning," *NIPS Tutorial*, vol. 2, 2015.

[10] L. Page, S. Brin, R. Motwani, and T. Winograd, "The pagerank citation ranking: Bringing order to the web." Stanford InfoLab, Tech. Rep., 1999.

[11] Y. Saad, *Iterative methods for sparse linear systems*. SIAM, 2003.

[12] L. Welch, "Lower bounds on the maximum cross correlation of signals (corresp.)," *IEEE Transactions on Information theory*, vol. 20, no. 3, pp. 397–399, 1974.

[13] D. Duplyakin, R. Ricci, A. Maricq, G. Wong, J. Duerig, E. Eide, L. Stoller, M. Hibler, D. Johnson, K. Webb *et al.*, "The design and operation of cloudlab," in *USENIX Annual Technical Conference (USENIX ATC 19)*. USENIX, 2019, pp. 1–14.

[14] L. D. Dalcin, R. R. Paz, P. A. Kler, and A. Cosimo, "Parallel distributed computing using python," *Advances in Water Resources*, vol. 34, no. 9, pp. 1124–1139, 2011.

[15] V. Smith, C.-K. Chiang, M. Sanjabi, and A. S. Talwalkar, "Federated multi-task learning," in *Advances in Neural Information Processing Systems*, 2017, pp. 4424–4434.