EURASIP Journal on Wireless Communications and Networking

RESEARCH **Open Access**

Private and rateless adaptive coded matrix-vector multiplication

Rawad Bitar[1*], Yuxuan Xing[2], Yasaman Keshtkarjahromi[3], Venkat Dasari[4], Salim El Rouayheb[5] and Hulya Seferoglu[2]

*Correspondence: rawad.bitar@tum.de
[1] Department of Electrical and Computer Engineering, Technical University of Munich, 80802 Munich, Germany
Full list of author information is available at the end of the article

The preliminary results of this paper are presented in part at the SPIE Defense + Commercial Sensing, Baltimore, MD, April 2019 [1].

Abstract
Edge computing is emerging as a new paradigm to allow processing data near the edge of the network, where the data is typically generated and collected. This enables critical computations at the edge in applications such as Internet of Things (IoT), in which an increasing number of devices (sensors, cameras, health monitoring devices, etc.) collect data that needs to be processed through computationally intensive algorithms with stringent reliability, security and latency constraints. Our key tool is the theory of coded computation, which advocates mixing data in computationally intensive tasks by employing erasure codes and offloading these tasks to other devices for computation. Coded computation is recently gaining interest, thanks to its higher reliability, smaller delay, and lower communication costs. In this paper, we develop a private and rateless adaptive coded computation (PRAC) algorithm for distributed matrix-vector multiplication by taking into account (1) the privacy requirements of IoT applications and devices, and (2) the heterogeneous and time-varying resources of edge devices. We show that PRAC outperforms known secure coded computing methods when resources are heterogeneous. We provide theoretical guarantees on the performance of PRAC and its comparison to baselines. Moreover, we confirm our theoretical results through simulations and implementations on Android-based smartphones.

Keywords: Distributed coded computing, Secret sharing, Rateless private codes, Heterogeneous computing clusters

1 Introduction

Edge computing is emerging as a new paradigm to allow processing data near the edge of the network, where the data is typically generated and collected. This enables computation at the edge in applications such as Internet of Things (IoT), in which an increasing number of devices (sensors, cameras, health monitoring devices, etc.) collect data that needs to be processed through computationally intensive algorithms with stringent reliability, security and latency constraints.

One of the promising solutions to handle computationally intensive tasks is computation offloading, which advocates offloading tasks to remote servers or cloud. Yet, offloading tasks to remote servers or cloud could be luxury that cannot be afforded by most of the edge applications, where connectivity to remote servers can be lost or compromised, which makes edge computing crucial.

Edge computing advocates that computationally intensive tasks in a device (master) could be offloaded to other edge or end devices (workers) in close proximity. However, offloading tasks to other devices leaves the IoT and the applications it is supporting at the complete mercy of an attacker. Furthermore, exploiting the potential of edge computing is challenging mainly due to the heterogeneous and time-varying nature of the devices at the edge. Thus, our goal is to develop a private, dynamic, adaptive, and heterogeneity-aware cooperative computation framework that provides both privacy and computation efficiency guarantees. Note that the application of this work can be extended to cloud computing at remote data-centers. However, we focus on edge computing as heterogeneity and time-varying resources are more prevalent at the edge as compared to data-centers.

We assume the following setup throughout the paper. A master device referred to as master wishes to offload intensive computations to n helper devices referred to as workers. The workers cannot be trusted with the privacy of the master's data but will run the required computations. The workers have different computation and network resources that change with time which creates a time-varying and heterogeneous computing environment. Workers that are slow or unresponsive are termed as stragglers. The goal of the master is to assign tasks that are proportional to the overall speed of the workers and tolerate the presence of straggling workers. This will be done by assigning a new task to a worker when it is expected to have finished its previous task.

Our key tool is the theory of coded computation, which advocates mixing data in computationally intensive tasks by employing erasure codes and offloading these tasks to other devices for computation [2–14]. The following canonical example demonstrates the effectiveness of coded computation.

Example 1 Consider the setup where a master device wishes to offload a task to 3 workers. The master has a large data matrix A and wants to compute matrix vector product $A\mathbf{x}$. The master device divides the matrix A row-wise equally into two smaller matrices A_1 and A_2, which are then encoded using a (3, 2) Maximum Distance Separable (MDS) code[1] to give $B_1 = A_1$, $B_2 = A_2$ and $B_3 = A_1 + A_2$, and sends each to a different worker. Also, the master device sends $\mathbf{x}$ to workers and ask them to compute $B_i\mathbf{x}$, $i \in \{1, 2, 3\}$. When the master receives the computed values (i.e., $B_i\mathbf{x}$) from at least two out of three workers, it can decode its desired task, which is the computation of $A\mathbf{x}$. The power of coded computations is that it makes $B_3 = A_1 + A_2$ act as a "joker" redundant task that can replace any of the other two tasks if they end up straggling or failing. □

The above example demonstrates the benefit of coding for edge computing. However, the very nature of task offloading from a master to worker devices makes the computation framework vulnerable to attacks. One of the attacks, which is also the focus of this work, is *eavesdropper adversary*, where one or more of workers can behave as an

[1] An (n, k) MDS code divides the master's data into k chunks and encodes it into n chunks $(n > k)$ such that any k chunks out of n are sufficient to recover the original data.

Table 1 Example PRAC operation in heterogeneous and time-varying setup

Time	Worker 1	Worker 2	Worker 3
1	R_1	$\mathbf{A_1} + \mathbf{A_3} + R_1$	$\mathbf{A_3} + R_1$
2		R_2	
3	$\mathbf{A_2} + \mathbf{A_3} + R_2$		
4			$\mathbf{A_2} + R_2$

The master distributes tasks to the workers in parallel. When a worker finishes its current task, the master assigns it a new task. The table depicts the behavior of the algorithm as a function of the times instances when a worker finishes its task at hand

eavesdropper and can spy on the coded data sent to these devices for computations.[2] For example, $B_3 = A_1 + A_2$ in Example 1 can be processed and spied by worker 3. Even though $A_1 + A_2$ is coded, the attacker can infer some information from this coded task. Privacy against eavesdropper attacks is extremely important in edge computing [15–17]. Thus, it is crucial to develop a private coded computation mechanism against eavesdropper adversary who can gain access to offloaded tasks.

In this paper, we develop a private and rateless adaptive coded computation (PRAC) mechanism. PRAC is (1) private as it is secure against eavesdropper adversary, (2) rateless, because it uses fountain codes [18–20] instead of Maximum Distance Separable (MDS) codes[3] [21, 22], and (3) adaptive as the master device offloads tasks to workers by taking into account their heterogeneous and time-varying resources. Next, we illustrate the main idea of PRAC through an illustrative example.

Example 2 We consider the same setup in Example 1, where a master device offloads a task to 3 workers. The master has a large data matrix A and wants to compute matrix vector product $A\mathbf{x}$. The master device divides matrix A row-wise into 3 sub-matrices A_1, A_2, A_3; and encodes these matrices using a fountain code[4] [18–20]. An example set of coded packets is A_2, A_3, $A_1 + A_3$, and $A_2 + A_3$. However, prior to sending a coded packet to a worker, the master generates a random key matrix R with the same dimensions as A_i and with entries drawn uniformly from the same alphabet as the entries of A. The key matrix is added to the coded packets to provide privacy as shown in Table 1. Since PRAC is adaptive, it requires the master to divide the tasks into smaller sub-tasks and send them sequentially (over time) to the workers. In particular, at the start of time slot 1, a key matrix R_1 is created and combined with $A_1 + A_3$ and A_3, and transmitted to workers 2 and 3, respectively. R_1 is also transmitted to worker 1 in order to obtain $R_1\mathbf{x}$ that will help the master in the decoding process. The computation of $(A_1 + A_3 + R_1)\mathbf{x}$ is completed at the end of time slot 1. Thus, at that time slot the master generates a new matrix, R_2, and sends it to worker 2. At the end of time slot 2, worker 1 finishes its computation, therefore the master adds R_2 to $A_2 + A_3$ and sends it to worker 1. A similar process is repeated at the end of time slot 3. Now the master waits for worker 2 to

[2] Note that this work focuses specifically on *eavesdropper adversary* although there are other types of attacks; for example *Byzantine adversary*, which is out of scope of this work.

[3] Our security scheme can be applied to any linear code. However, we choose fountain code since they are a better fit for edge computing characterized by heterogeneous and time-varying behavior.

[4] fountain codes are desirable here for two properties: (1) they provide a fluid abstraction of the coded packets so the master can always decode with high probability as long as it collects enough packets; (2) They have low decoding complexity.

return $R_2\mathbf{x}$ and for any other worker to return its uncompleted task in order to decode $A\mathbf{x}$. Thanks to using key matrices R_1 and R_2, and assuming that workers do not collude, privacy is guaranteed. On a high level, privacy is guaranteed because the observation of the workers is statistically independent from the data A. $\square$

This example shows that PRAC can take advantage of coding for computation, and provide privacy.

Organization The structure of the rest of this paper is as follows. We summarize our contributions on a high level in Sect. 2. We give a brief overview of the related work in Sect. 3. We present the system model in Sect. 4. Section 5 presents the design of private and rateless adaptive coded computation (PRAC). We characterize and analyze PRAC in Sect. 6. We present evaluation results in Sect. 7. Section 8 concludes the paper.

2 Results and discussion

We design PRAC for heterogeneous and time-varying private coded computing with colluding workers. In particular, PRAC codes sub-tasks using fountain codes, and determines how many coded packets and keys each worker should compute dynamically over time. We provide theoretical analysis of PRAC and show that it (1) guarantees privacy conditions, (2) uses minimum number of keys to satisfy privacy requirements, and (3) maintains the desired rateless property of non-private fountain codes. Furthermore, we provide a closed form task completion delay analysis of PRAC. Finally, we evaluate the performance of PRAC via simulations as well as in a test bed consisting of real Android-based smartphones as compared to baselines.

Recently there has been significant interest in applying coding theoretic techniques to speed up machine learning algorithms, as detailed in the next section. While most of the literature has focused on mitigation of slow workers, several recent works consider security on top of it, e.g., [10, 23–26]. In a recent work, the authors of [27] show that, under certain model assumptions, there are regimes, in terms of data splitting and number of workers used, where offloading tasks to the workers can be faster than doing the computations locally for large dimensional data. In our paper, we assume that we are operating in these large-scale regimes where offloading is needed. In this context, we view the contribution of this paper as posing the problem of bringing and adapting coded computation tools to applications on the edge such as IoT. As such, our contribution includes two parts: (1) In IoT, the resource availability experiences high fluctuation over time. Adapting rateless codes, e.g., [28] to privacy is not immediate due to the need of MDS codes to satisfy the privacy constraints (cf. “Appendix 2”). This paper shows that even though MDS code is needed for privacy, the encoding of the data itself can be arbitrary. (2) We provide implementation on android devices (phones and tablets) to substantiate the suitability of our proposed algorithm to edge computing.

3 Related work

Mobile cloud computing is a rapidly growing field with the aim of providing better experience of quality and extensive computing resources to mobile devices [29, 30]. The main solution to mobile computing is to offload tasks to the cloud or to neighboring devices by exploiting connectivity of the devices. With task offloading come several challenges such as heterogeneity of the devices, time varying communication channels and energy efficiency, see e.g., [31–34]. We refer interested reader to [2] and references within for a detailed literature on edge computing and mobile cloud computing.

The problem of stragglers in distributed systems is initially studied by the distributed computing community, see e.g., [35–38]. Research interest in using coding theoretical techniques for straggler mitigation in distributed content download and distributed computing is rapidly growing. The early body of work focused on content download, see e.g., [39–43]. Using codes for straggler mitigation in distributed computing started in [12] where the authors proposed the use of MDS codes for distributed linear machine learning algorithms in homogeneous workers setting.

Following the work of [12], coding schemes for straggler mitigation in distributed matrix-matrix multiplication, coded computing and machine learning algorithms are introduced and the fundamental limits between the computation load and the communication cost are studied, see e.g., [8, 44] and references within for matrix-matrix multiplication, see [4, 7, 10–13, 24, 28, 45–53] for machine learning algorithms and [5, 6, 9, 54] and references within for other topics.

Codes for privacy and straggler mitigation in distributed computing are first introduced in [3, 26] where the authors consider a homogeneous setting and focus on matrix-vector multiplication. The problem of private distributed matrix-matrix multiplication and private polynomial computation with straggler tolerance is studied [23, 55–59]. In the private matrix-matrix multiplication setting, the master wants to simultaneously maintain the privacy of both matrices which is a generalization of the matrix-vector multiplication setting. The former works are designed for the homogeneous static setting in which the master has a prior knowledge on the computation capacities of the workers and pre-assigns the sub-tasks equally to them. In addition, the master sets a threshold on the number of stragglers that it can tolerate throughout the whole process. In contrast, PRAC is designed for the heterogeneous dynamic setting in which workers have different computation capacities that can change over time. PRAC assigns the sub-tasks to the workers in an adaptive manner based on the estimated computation capacity of each worker. Furthermore, PRAC can tolerate a varying number of stragglers as it uses an underlying rateless code, which gives the master a higher flexibility in adaptively assigning the sub-tasks to the workers. Those properties of PRAC allow a better use of the workers over the whole process. On the other hand, PRAC is restricted to matrix-vector multiplication. Although coded computation is designed for linear operations, there is a recent effort to apply coded computation for nonlinear operations. For example, [25] applied coded computation to logistic regression, and the framework of Gradient coding started in [10] generalizes to any gradient-descent algorithm. Our work is complementary with these works. For example, our work can be directly used as complementary to [25] to provide privacy and adaptive task offloading to logistic regression.

Secure multi-party communication (SMPC) [60] can be related to our work as follows. The setting of secure multi-party computing schemes assumes the presence of several parties (masters in our terminology) who want to compute a function of all the data owned by the different parties without revealing any information about the individual data of each party. This setting is a generalized version of the master/worker setting that we consider. More precisely, an SMPC scheme reduces to our Master/worker setting if we assume that only one party owns data and the others have no data to include in the function to be computed. SMPC schemes use threshold secret sharing schemes, therefore they restrict the master to a fixed number of stragglers. Thus, showing that PRAC outperforms Staircase codes (which are the best known family of threshold secret sharing schemes) implies that PRAC outperform the use of SMPC schemes that are reduced to this setting.

Works on privacy-preserving machine learning algorithms are also related to our work. However, the privacy constraint in this line of work is computational privacy and the proposed solutions do not take stragglers into account, see e.g., [61–63].

Private information retrieval [64] is also related to distributed coded computing as noted in [65]. A scheme designed for private matrix-vector multiplication can be used as a PIR scheme where the data of interest is replicated among the servers. Therefore, under some manipulation of the stored data, this scheme can be seen as a PIR scheme with flexible rate.

We restrict the scope of this paper to eavesdropping attacks, which are important on their own merit. We do not consider security against Byzantine attacks, where the malicious (adversarial) workers send corrupted data to the master in order to corrupt the whole computation process. Privacy and security can be achieved by using Maximum Distance Separable (MDS)-like codes which restrict the master to a fixed maximum number of stragglers [23, 58]. Our solution on the other hand addresses the privacy problem in an adaptive coded computation setup without such a restriction. In this setup, security cannot be addressed by expanding the results of [23, 58]. In fact, we developed a secure adaptive coded computation mechanism in our recent paper [66] against Byzantine attacks. The mechanism in [66] allows the master to detect with high probability the presence of malicious workers, yet it does not ensure privacy of the data. The private and secure adaptive coded computation obtained by combining this paper and [66] is out of scope of this paper.

4 System model

Setup We consider a master/workers setup at the edge of the network, where the master device M offloads its computationally intensive tasks to workers w_i, $i \in [n]$, via device-to-device (D2D) links such as Wi-Fi Direct and/or Bluetooth. The master device divides a task into smaller sub-tasks, and offloads them to workers that process these sub-tasks in parallel.

Task Model We focus on the computation of linear functions, i.e., matrix-vector multiplication. We suppose the master wants to compute the matrix vector product $A\mathbf{x}$, where $A \in \mathbb{F}_q^{m\times\ell}$ can be thought of as the data matrix and $\mathbf{x} \in \mathbb{F}_q^{\ell}$ can be thought of as an attribute vector. We assume that the entries of A and $\mathbf{x}$ are drawn independently and uniformly at random[5] from $\mathbb{F}_q$. The motivation stems from machine learning applications where computing linear functions is a building block of several iterative algorithms [67, 68]. For instance, the main computation of a gradient descent algorithm with squared error loss function is

[5] We abuse notation and denote both the random matrix representing the data and its realization by A. We do the same for $\mathbf{x}$.

$$\mathbf{x}^{+} = \mathbf{x} - \alpha A^{T}(A\mathbf{x} - \mathbf{y}), \tag{1}$$

where $\mathbf{x}$ is the value of the attribute vector at a given iteration, $\mathbf{x}^{+}$ is the updated value of $\mathbf{x}$ at this iteration and the learning rate α is a parameter of the algorithm. Equation (1) consists of computing two linear functions $A\mathbf{x}$ and $A^{T}\mathbf{w} \triangleq A^{T}(A\mathbf{x} - \mathbf{y})$.

Worker and attack model The workers incur random delays while executing the task assigned to them by the master device. The workers have different computation and communication specifications resulting in a heterogeneous environment which includes workers that are significantly slower than others, known as stragglers. Moreover, the workers cannot be trusted with the master's data. We consider an *eavesdropper adversary* in this paper, where one or more of workers are compromised by an adversary who wants to spy on the coded data sent to these devices for computations. In a comprehensive solution for IoT networks, we assume that standard security protocols (such as datagram transport layer security—DTLS) will be in place, in addition to our proposed scheme, in order to protect the wireless links from eavesdropping from nodes other than the intended workers in the master-worker links. We assume that up to z, $z < n$, workers can collude, i.e., z workers can share the data they received from the master in order to obtain information about A. The parameter z can be chosen based on the desired privacy level; a larger z means a higher privacy level and vice versa. One would want to set z to the largest possible value for maximum, $z = n - 1$ security purposes. However, this has the drawback of increasing the complexity and the runtime of the algorithm. In our setup we assume that z is a fixed and given system parameter.

Coding and secret keys The matrix A can be divided into b row blocks (we assume that b divides m, otherwise all-zero rows can be added to the matrix to satisfy this property) denoted by A_i, $i = 1, \dots, b$. The master applies fountain coding [18–20] across row blocks to create information packets $\nu_j \triangleq \sum_{i=1}^{m} c_{i,j}A_i$, $j = 1, 2, \dots$, where the $c_{i,j} \in \{0, 1\}$. An information packet is a matrix of dimension $m/b \times \ell$, i.e., $\nu_j \in \mathbb{F}_q^{m/b \times \ell}$. Note that information packets can be encoded using any linear code. Fountain codes enable a fluid encoding of the information and allow the master to obtain $A\mathbf{x}$ as long as enough packets are collected from the workers, irrespective of their origin, which makes them a better fit for the heterogeneous and time-varying setting considered in this paper [2]. In order to maintain privacy of the data, the master device generates random matrices R_i of dimension $m/b \times \ell$ called *keys*. The entries of the R_i matrices are drawn uniformly at random from the same field as the entries of A. Each information packet ν_j is *padded* with a linear combination of z keys $f_j(R_{i,1}, \dots, R_{i,z})$ to create a secure packet $s_j \in \mathbb{F}_q^{m/b \times \ell}$ defined as $s_j \triangleq \nu_j + f_j(R_{i,1}, \dots, R_{i,z})$. We show in "Appendix 2" that encoding the random keys using an MDS code is necessary to guarantee the perfect privacy of the data. Therefore, even though information packets are encoded using Fountain codes, the linear combinations of the random matrices are created using MDS codes.

The master device sends $\mathbf{x}$ to all workers, then it sends the keys and the s_j's to the workers according to our PRAC scheme described later. Each worker multiplies the received packet by $\mathbf{x}$ and sends the result back to the master. Since the encoding is rateless, the master keeps sending packets to the workers until it can decode $A\mathbf{x}$. The master then sends a stop message to all the workers.

Privacy conditions Our primary requirement is that any collection of z (or less) workers will not be able to obtain any information about A, in an information theoretic sense.

In particular, let P_i, $i = 1 \dots, n$, denote the collection of packets sent to worker w_i. For any set $\mathcal{B} \subseteq \{1, \dots, n\}$, let $P_{\mathcal{B}} \triangleq \{P_i, i \in \mathcal{B}\}$ denote the collection of packets given to worker w_i for all $i \in \mathcal{B}$. The privacy requirement[6] can be expressed as

$$H(A|P_{\mathcal{Z}}) = H(A), \quad \forall \mathcal{Z} \subseteq \{1, \dots, n\} \text{ s.t. } |\mathcal{Z}| \leq z. \tag{2}$$

$H(A)$ denotes the entropy, or uncertainty, about A and $H(A|P_{\mathcal{Z}})$ denotes the uncertainty about A after observing $P_{\mathcal{Z}}$.

Delay model We focus on the delays incurred by distributing the tasks to the workers and overlook the time spent on encoding and decoding the tasks[7] at the master. Each packet transmitted from the master to a worker w_i, $i = 1, 2, \dots, n$, experiences the following delays: (1) transmission delay for sending the packet from the master to the worker, (2) computation delay for computing the multiplication of the packet by the vector $\mathbf{x}$, and (3) transmission delay for sending the computed packet from the worker w_i back to the master. We denote by $\beta_{t,i}$ the computation time of the tth packet at worker w_i and RTT_i denotes the average round-trip time spent to send and receive a packet from worker w_i.

5 Design of PRAC

5.1 Overview

We present the detailed explanation of PRAC. Let $p_{t,i} \in \mathbb{F}_q^{m/b \times \ell}$ be the tth packet sent to worker w_i. This packet can be either a key, $p_{t,i} = R_{t,i}$, or a secure packet $p_{t,i} = s_{t,i} = v_{t,i} + f_j(R_{t,i}, \dots, R_{t,z})$. For each value of t, the master sends z keys denoted by $R_{t,1}, \dots, R_{t,z}$ to z different workers and up to $n - z$ secure packets $s_{t,1}, \dots, s_{t,n-z}$ to the remaining workers. The master needs the results of $b + \epsilon$ information packets, i.e., $v_{t,i}\mathbf{x}$, to decode the final result $A\mathbf{x}$, where ϵ is the overhead required by fountain coding.[8] To obtain the results of $b + \epsilon$ information packets, the master needs the results of $b + \epsilon$ secure packets, $s_{t,i}\mathbf{x} = (v_{i,j} + f_j(R_{t,i}, \dots, R_{t,z}))\mathbf{x}$, together with all the corresponding.[9] $R_{t,i}\mathbf{x}, i = 1, \dots, z$. Therefore, only the results of the $s_{t,i}\mathbf{x}$ for which all the computed keys $R_{t,i}\mathbf{x}, i = 1, \dots, z$, are received by the master can account for the total of $b + \epsilon$ information packets.

5.2 Dynamic rate adaptation

The dynamic rate adaptation part of PRAC is based on [2]. In particular, the master offloads coded packets gradually to workers and receives two acknowledgements (ACK) from each worker for each transmitted packet; one confirming the receipt of the packet

[6] In some cases the vector $\mathbf{x}$ may contain information about A and therefore must not be revealed to the workers. We explain in "Appendix 1" how to generalize our scheme to account for such cases.

[7] It is worth noting that fountain codes enjoy a linear time encoding and decoding complexity. Other codes such Reed-Solomon codes require inverting a $k \times k$ matrix and have decoding complexity of at least $\mathcal{O}(k \log k)$ if the generator matrix is a Vandermonde matrix and $\mathcal{O}(k^2)$ in general.

[8] We do not account for the probability of decoding failure because we assume that the master receives enough packets to decode with probability one. The overhead of packets required by fountain coding for the master to decode is typically as low as 5% [20], i.e., $\epsilon = 0.05b$.

[9] Recall that $f_j(R_{t,1}, \dots, R_{t,z})$ is a linear function, thus it is easy to extract $(R_{t,i})\mathbf{x}, i = 1, \dots, z$, from $(f_j(R_{t,1}, \dots, R_{t,z}))\mathbf{x}$.

by the worker, and the second one (piggybacked to the computed packet) showing that the packet is computed by the worker. Then, based on the frequency of the received ACKs, the master decides to transmit more/less coded packets to that worker. In particular, each packet $p_{t,i}$ is transmitted to each worker w_i before or right after the computed packet $p_{t-1,i}\mathbf{x}$ is received at the master. For this purpose, the average per packet computing time $\mathbb{E}[\beta_{t,i}]$ is calculated for each worker w_i dynamically based on the previously received ACKs. Each packet $p_{t,i}$ is transmitted after waiting $\mathbb{E}[\beta_{t,i}]$ from the time $p_{t-1,i}$ is sent or right after packet $p_{t-1,i}\mathbf{x}$ is received at the master, thus reducing the idle time at the workers. This policy is shown to approach the optimal task completion delay and maximizes the workers' efficiency and is shown to improve task completion time significantly compared with the literature [2].

5.3 Coding

We explain the coding scheme used in PRAC. We start with an example to build an intuition and illustrate the scheme before going into details.

Example 3 Assume there are $n = 4$ workers out of which any $z = 2$ can collude. Let A and $\mathbf{x}$ be the data owned by the master and the vector to be multiplied by A, respectively. The master sends $\mathbf{x}$ to all the workers. For the sake of simplicity, assume A can be divided into $b = 6$ row blocks, i.e., $A = \begin{bmatrix} A_1^T & A_2^T & \dots & A_6^T \end{bmatrix}^T$. The master encodes the A_i's using fountain code. We denote by *round* the event when the master sends a new packet to a worker. For example, we say that worker 1 is at round 3 if it has received 3 packets so far. For every round t, the master generates $z = 2$ random matrices $R_{t,1},\ R_{t,2}$ (with the same size as A_1) and encodes them using an $(n, z) = (4, 2)$ systematic maximum distance separable (MDS) code by multiplying $R_{t,1},\ R_{t,2}$ by a generator matrix G as follows

$$G \begin{bmatrix} R_{t,1} \\ R_{t,2} \end{bmatrix} \triangleq \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \\ 1 & 2 \end{bmatrix} \begin{bmatrix} R_{t,1} \\ R_{t,2} \end{bmatrix}. \tag{3}$$

This results in the encoded matrices of $R_{t,1}$, $R_{t,2}$, $R_{t,1} + R_{t,2}$, and $R_{t,1} + 2R_{t,2}$. Now let us assume that workers can be stragglers. At the beginning, the master initializes all the workers at round 1. Afterwards, when a worker w_i finishes its task, the master checks how many packets this worker has received so far and how many other workers are at this round. If this worker w_i is the first or second to be at round t, the master generates $R_{t,1}$ or $R_{t,2}$, respectively, and sends it to w_i. Otherwise, if w_i is the jth worker ($j > 2$) to be at round t, the master multiplies $\begin{bmatrix} R_{t,1} & R_{t,2} \end{bmatrix}^T$ by the jth row of G, adds it to a generated fountain coded packet, and sends it to w_i. The master keeps sending packets to the workers until it can decode $A\mathbf{x}$. We illustrate the idea in Table 2.

We now explain the details of PRAC in the presence of z colluding workers.

1 *Initialization* The master divides A into b row blocks $A_1, \dots, A_b$ and sends the vector $\mathbf{x}$ to the workers. Let $G \in \mathbb{F}_q^{n\times z}$, $q > n$, be the generator matrix of an (n, z) systematic MDS code. For example one may use systematic Reed-Solomon codes that use Van-

Table 2 Depiction of PRAC in the presence of stragglers

Time	Worker 1	Worker 2	Worker 3	Worker 4
1	$R_{1,1}$	$R_{1,2}$	$\mathbf{A_4} + R_{1,1} + R_{1,2}$	$\mathbf{A_3} + \mathbf{A_4} + \mathbf{A_6} + R_{1,1} + 2R_{1,2}$
2				$R_{2,1}$
3	$R_{2,2}$			
4		$\mathbf{A_3} + R_{2,1} + R_{2,2}$	$\mathbf{A_4} + \mathbf{A_5} + R_{2,1} + 2R_{2,2}$	
5		$R_{3,1}$		
6	$\mathbf{A_2} + R_{3,1} + R_{3,2}$			$R_{3,2}$
7		$R_{4,1}$	$\mathbf{A_1} + R_{3,1} + 2R_{3,2}$	
8	$R_{4,2}$			$\mathbf{A_2} + \mathbf{A_3} + R_{4,1} + R_{4,2}$

The master keeps generating packets using fountain codes until it can decode $A\mathbf{x}$. The master estimates the average task completion time of each worker. When a worker is expected to finish the task at hand, the master sends a new task to that worker. This mechanism avoids idle time at the workers. Thus, at every time instance in which a worker is expected to finish its task, the master creates a new packet and sends it to that worker. Each new packet sent to a worker must be secured with a new random key. The master can decode $A_1\mathbf{x}, \ldots, A_6\mathbf{x}$ after receiving all the packets not having $R_{4,1}$ or $R_{4,2}$ in them

dermonde matrix as generator matrix, see for example [69]. The master generates z random matrices $R_{1,1}, \ldots, R_{1,z}$ and encodes them using G. The generation of truly random numbers can be done as in [70, 71], where the master harnesses entropy from internal and external sources to guarantee true randomness. Each coded key can be denoted by $\mathbf{g}_i\mathcal{R}_1$ where $\mathbf{g}_i$ is the ith row of G and $\mathcal{R}_1 \triangleq \begin{bmatrix} R_{1,1}^T & \ldots & R_{1,z}^T \end{bmatrix}^T$. The master sends the z keys $R_{1,1}, \ldots, R_{1,z}$ to the first z workers, generates $n-z$ fountain coded packets of the A_i's, adds to each packet an encoded random key $\mathbf{g}_i\mathcal{R}_1$, $i = z+1, \ldots n$, and sends them to the remaining $n-z$ workers.

2 *Encoding and adaptivity* When the master wants to send a new packet to a worker (noting that a packet $p_{t,i}$ is transmitted to worker w_i before, or right after, the computed packet $p_{t-1,i}\mathbf{x}$ is received at the master according to the strategy described in Sect. 5.2), it checks at which round this worker is, i.e., how many packets this worker has received so far, and checks how many other workers are at least at this round. Assume worker w_i is at round t and $j-1$ other workers are at least at this round. If $j \leq z$, the master generates and sends $R_{t,j}$ to the worker. However, if $j > z$ the master generates a fountain coded packet of the A_i's (e.g., $A_1 + A_2$), adds to it $\mathbf{g}_j\mathcal{R}_t$ and sends the packet $(A_1 + A_2 + \mathbf{g}_j\mathcal{R}_t)$ to the worker. Each worker computes the multiplication of the received packet by the vector $\mathbf{x}$ and sends the result to the master.

3 *Decoding and speed* Let τ_i denote the number of packets sent to worker i. We define τ_{max} as the largest value of τ_i for which the master has received all the $R_{t,i}\mathbf{x}$ for all $i = 1, \ldots, z$ and $t = 1, \ldots, \tau_{max}$. The master can therefore subtract $R_{t,i}$, $t = 1, \ldots, \tau_{max}$ and $i = 1, \ldots, z$, from all received secure information packets, and thus can decode the A_i's using the fountain code decoding process. The number of secure packets that can be used to decode the A_i's is dictated by the $(z+1)$st fastest worker, i.e., the master can only use the results of secure information packets computed at a given round if at least $z+1$ workers have completed that round. Let u_t denote the number of workers which completed round t, then the master can use $\max\{u_t - z, 0\}$ information packets from round t in the decoding process. In order to decode $A\mathbf{x}$, the master needs $\sum_{t=1}^{\tau_{max}} u_t = b + \epsilon$ information packets in total. For example, if the z fastest workers have completed round 100 and the $(z+1)$st fastest worker has completed round 20, the master can only use the packets belonging to the first 20 rounds. The reason is that the master needs all the keys corresponding to a given round in order to use the secure information packet for decoding. In Lemma 2

we prove that this scheme is optimal, i.e., in private coded computing the master cannot use the packets computed at rounds finished by less than $z+1$ workers irrespective of the coding scheme.

6 Performance analysis of PRAC

6.1 Privacy

In this section, we provide theoretical analysis of PRAC by particularly focusing on its privacy properties.

Theorem 1 *PRAC is a rateless real-time adaptive coded computing scheme that allows a master device to run distributed linear computation on private data A via n workers while satisfying the privacy constraint given in* (2) *for a given $z < n$.*

Proof Since the random keys are generated independently at each round, it is sufficient to study the privacy of the data on one round and the privacy generalizes to the whole algorithm. We show that for any subset $Z \subset \{1, \dots, n\}, |Z| = z$, the collection of packets $p_Z \triangleq \{p_{t,i}, i \in Z\}$ sent at round t reveals no information about the data A as given in (2), i.e., $H(A) = H(A|p_Z)$. Let K denote the random variable representing all the keys generated at round t, then it is enough to show that $H(K|A, p_Z) = 0$ as detailed in "Appendix 2." Therefore, we need to show that given A as side information, any z workers can decode the random keys $R_{t,1}, \dots, R_{t,z}$. Without loss of generality assume the workers are ordered from fastest to slowest, i.e., worker w_1 is the fastest at the considered round t. Since the master sends z random keys to the fastest z workers, then $p_{t,i} = R_{t,i}, i = 1, \dots, z$. The remaining $n-z$ packets are secure information packets sent to the remaining $n-z$ workers, i.e., $p_{t,i} = s_{t,i} = \nu_{t,i} + f(R_{t,1}, \dots, R_{t,z})$, where $\nu_{t,i}$ is a linear combination of row blocks of A and $f(R_{t,1}, \dots, R_{t,z})$ is a linear combination of the random keys generated at round t. Given the data A as side information, any collection of z packets can be expressed as z codewords of the (n, z) MDS code encoding the random keys. Thus, given the matrix A, any collection of z packets is enough to decode all the keys and $H(K|A, p_Z) = 0$ which concludes the proof. □

Remark 1 PRAC requires the master to wait for the $(z+1)$st fastest worker in order to be able to decode $A\mathbf{x}$. We show in Lemma 2 that this limitation is a byproduct of all private coded computing schemes.

Remark 2 PRAC uses the minimum number of keys required to guarantee the privacy constraints. At each round PRAC uses exactly z random keys which is the minimum amount of required keys (c.f. Equation (12) in "Appendix 2").

Lemma 2 *Any private coded computing scheme for distributed linear computation limits the master to the speed of the* $(z+1)$*st fastest worker.*

Proof The proof of Lemma 2 is provided in "Appendix 3." □

6.2 Task completion delay

In this section, we characterize the task completion delay of PRAC and compare it with Staircase codes [3], which are secure against eavesdropping attacks in a coded computation setup with homogeneous resources. First, we start with task completion delay characterization of PRAC.

Theorem 3 *Let b be the number of row blocks in A, let $\beta_{t,i}$ denote the computation time of the tth packet at worker w_i and let RTT_i denote the average round-trip time spent to send and receive a packet from worker i. The task completion time of PRAC is approximated as*

$$T_{\mathrm{PRAC}} \approx \max_{i\in\{1,\dots,n\}}\{\mathrm{RTT}_i\} + \frac{b+\epsilon}{\sum_{i=z+1}^{n} 1/\mathbb{E}[\beta_{t,i}]}, \tag{4}$$

$$\approx \frac{b+\epsilon}{\sum_{i=z+1}^{n} 1/\mathbb{E}[\beta_{t,i}]}, \tag{5}$$

where w_i's are ordered indices of the workers from fastest to slowest, i.e., $w_1 = \arg\min_i \mathbb{E}[\beta_{t,i}]$.

Proof The proof of Theorem 3 is provided in "Appendix 4." □

Now that we characterized the task completion delay of PRAC, we can compare it with the state-of-the-art. Secure coded computing schemes that exist in the literature usually use static task allocation, where tasks are assigned to workers a priori. The most recent work in the area is Staircase codes, which is shown to outperform all existing schemes that use threshold secret sharing [3]. However, Staircase codes are static; they allocate fixed amount of tasks to workers a priori. Thus, Staircase codes cannot leverage the heterogeneity of the system, neither can it adapt to a system that is changing in time. On the other hand, our solution PRAC adaptively offloads tasks to workers by taking into account the heterogeneity and time-varying nature of resources at workers. Therefore, we restrict our focus on comparing PRAC to Staircase codes.

Staircase codes assigns a task of size $b/(k-z)$ row blocks to each worker.[10] Let T_i be the time spent at worker i to compute the whole assigned task. Denote by $T_{(i)}$ the ith order statistic of the T_i's and by $T_{\mathrm{SC}}(n,k,z)$ the task completion time, i.e., time the master waits until it can decode $A\mathbf{x}$, when using Staircase codes. In order to decode $A\mathbf{x}$ the master needs to receive a fraction equal to $(k-z)/(d-z)$ of the task assigned to each worker from any d workers where $k \le d \le n$. The task completion time of the master is expressed as [3]

$$T_{\mathrm{SC}}(n,k,z) = \min_{d\in\{k,\dots,n\}}\left\{\frac{k-z}{d-z}T_{(d)}\right\}. \tag{6}$$

[10] Note that in addition to n and z, all threshold secret sharing based schemes require a parameter k, $z < k < n$, which is the minimum number of non stragglers that the master has to wait for before decoding $A\mathbf{x}$.

Theorem 4 *The gap between the completion time of PRAC and coded computation using staircase codes is lower bounded by:*

$$\mathbb{E}[T_{\mathrm{SC}}] - \mathbb{E}[T_{\mathrm{PRAC}}] \geq \frac{bx - \epsilon y}{y(x+y)}, \tag{7}$$

where $x = \dfrac{n-d^*}{E[\beta_{t,n}]}$, $y = \dfrac{d^* - z}{E[\beta_{t,d^*}]}$ *and* d^* *is the value of* d *that minimizes Eq. (6).*

Proof We provide the proof of Theorem 4 in [72]. □

Theorem 4 shows that the lower bound on the gap between secure coded computation using staircase codes and PRAC is in the order of number of row blocks of A. Hence, the gap between secure coded computation using Staircase codes and PRAC is linearly increasing with the number of row blocks of A. Note that, ϵ, the required overhead by fountain coding used in PRAC, becomes negligible as b increases.

Thus, PRAC outperforms secure coded computation using Staircase codes in heterogeneous systems. The more heterogeneous the workers are, the more improvement is obtained by using PRAC. However, Staircase codes can slightly outperform PRAC in the case where the slowest $n-z$ workers are homogeneous, i.e., have similar compute service times T_i. In this case both algorithms are restricted to the slowest $n-z$ workers (see Lemma 2), but PRAC incurs an ϵ overhead of tasks (due to using fountain codes) which is not needed for Staircase codes. In particular, from (5) and (6), when the $n-z$ slowest workers are homogeneous, the task completion time of PRAC and Staircase codes are equal to $\frac{b+\epsilon}{n-z}\mathbb{E}[\beta_{t,n}]$ and $\frac{b}{n-z}\mathbb{E}[\beta_{t,n}]$, respectively.

7 Experimental performance evaluation

7.1 Simulations

In this section, we present simulations run on MATLAB, and compare PRAC with the following baselines: (1) Staircase codes [3], (2) C3P [2] (which is not secure as it is not designed to be secure), and (3) Genie C3P (GC3P) that extends C3P by assuming a knowledge of the identity of the eavesdroppers and ignoring them. We note that GC3P serves as a lower bound on private coded computing schemes for heterogeneous systems[11] for the following reason: for a given number of z colluding workers the ideal coded computing scheme knows which workers are eavesdroppers and ignores them to use the remaining workers without need of randomness. If the identity of the colluding workers is unknown, coded computing schemes require randomness and become limited to the $(z+1)$st fastest worker (Lemma 2). GC3P and other coded computing schemes have similar performance if the z colluding workers are the fastest workers. If the z colluding workers are the slowest, then GC3P outperforms any coded computing scheme. Note that our solution PRAC considers the scenario of unknown eavesdroppers. Comparing PRAC with G3CP shows how good PRAC is as compared to the best possible solution for heterogeneous systems. In terms of comparing PRAC to solutions designed for the homogeneous setting, we restrict our attention to Staircase codes which

[11] If the system is homogeneous Staircase codes outperform GC3P, because pre-allocating tasks to the workers avoids the overhead needed by fountain codes.

are a class of secret sharing schemes that enjoys a flexibility in the number of workers needed to decode the matrix-vector multiplication. Staircase codes are shown to outperform any coded computing scheme that requires a threshold on the number of stragglers [3].

In our simulations, we model the computation time of each worker w_i by an independent shifted exponential random variable with rate λ_i and shift c_i, i.e., $F(T_i = t) = 1 - \exp(-\lambda_i(t - c_i))$. We take $c_i = 1/\lambda_i$ and consider three different scenarios for choosing the values of λ_i's for the workers as follows:

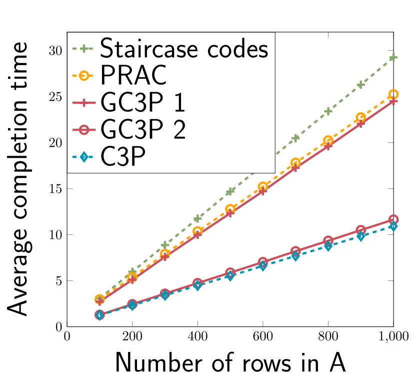
- *Scenario 1* we assign $\lambda_i = 3$ for half of the workers, then we assign $\lambda_i = 1$ for one quarter of the workers and assign $\lambda_i = 9$ for the remaining workers.
- *Scenario 2* we assign $\lambda_i = 1$ for one third of the workers, the second third have $\lambda_i = 3$ and the remaining workers have $\lambda_i = 9$.
- *Scenario 3* we draw the λ_i's independently and uniformly at random from the interval [0.5, 9].

When running Staircase codes, we choose the parameter k that minimizes the task completion time for the desired n and z. We do so by simulating Staircase codes for all possible values of k, $z \leq k \leq n$, and choose the one with the minimum completion time.

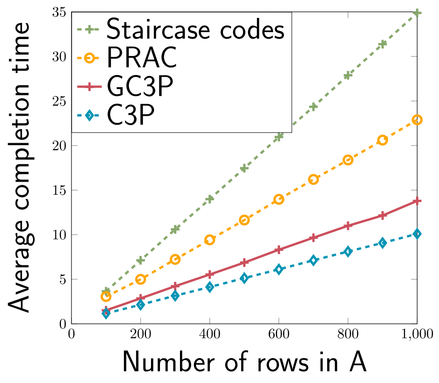
We take $b = m$, i.e., each row block is simply a row of A. The size of each element of A and vector $\mathbf{x}$ are assumed to be 1 Byte (or 8 bits). Therefore, the size of each transmitted packet $p_{t,i}$ is $8 * \ell$ bits. For the simulation results, we assume that the matrix A is a square matrix, i.e., $l = m$. We take $m = 1000$, unless explicitly stated otherwise. C_i denotes the average channel capacity of each worker w_i and is selected uniformly from the interval [10, 20] Mbps. The rate of sending a packet to worker w_i is sampled from a Poisson distribution with mean C_i.

In Fig. 1 we show the effect of the number of rows m on the completion time at the master. We fix the number of workers to 50 and the number of colluding workers to 13 and plot the completion time for PRAC, C3P, GC3P and Staircase codes. Notice that PRAC and Staircase codes have close completion time in scenario 1 (Fig. 3a) and this completion time is far from that of C3P. The reason is that in this scenario we pick exactly 13 workers to be fast ($\lambda_i = 9$) and the others to be significantly slower. Since PRAC assigns keys to the fastest z workers, the completion time is dictated by the slow workers. To compare PRAC with Staircase codes notice that the majority of the remaining workers have $\lambda_i = 3$ therefore pre-allocating equal tasks to the workers is close to adaptively allocating the tasks.

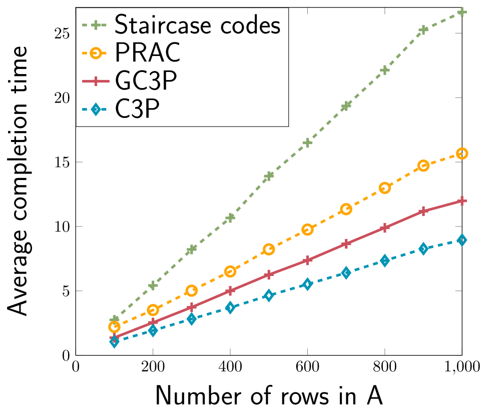
In terms of lower bound on PRAC, observe that when the fastest workers are assumed to be adversarial, GC3P and PRAC have very similar task completion time. However, when the slowest workers are assumed to be adversarial the completion of GC3P is very close to C3P and far from PRAC. This observation is in accordance with Lemma 2. In scenarios 2 and 3 we pick the adversarial workers uniformly at random and observe that the completion time of PRAC becomes closer to GC3P when the workers are more heterogeneous. For instance, in scenario 3, GC3P and PRAC have closer performance when the workers' computing times are chosen uniformly at random from the interval [0.5, 9].



a Scenario 1 with the fastest 13 workers as eavesdropper for GC3P 1 and the slowest workers as eavesdropper for GC3P 2.



b Scenario 2 with 13 workers picked at random to be eavesdroppers.

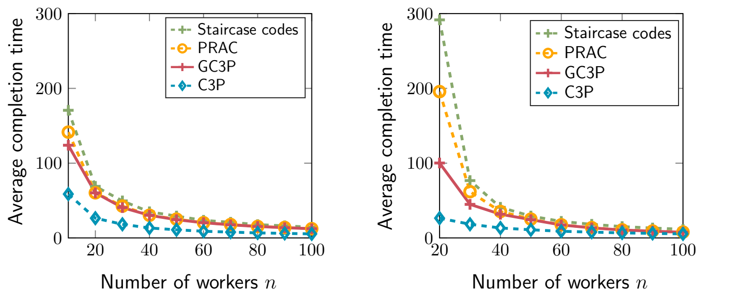


c Scenario 3 with 13 workers picked at random to be eavesdroppers.

Fig. 1 Comparison between PRAC and the baselines Staircase codes, GC3P, and C3P in different scenarios with $n = 50$ workers and $z = 13$ colluding eavesdroppers for different values of the number of rows m. For each value of m we run 100 experiments and average the results. When the eavesdropper are chosen to be the fastest workers, PRAC has very similar performance to GC3P. When the eavesdroppers are picked randomly, the performance of PRAC becomes closer to this of GC3P when the non adversarial workers are more heterogeneous

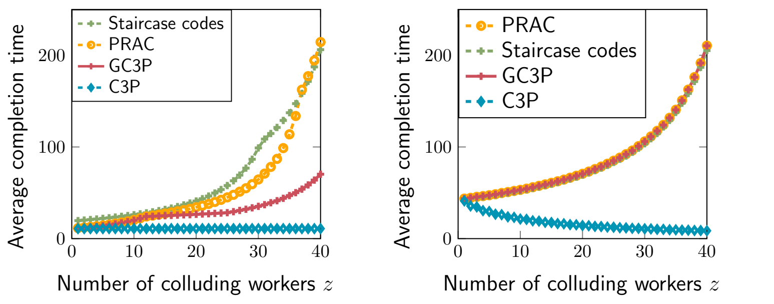
In Fig. 2, we plot the task completion time as a function of the number of workers n for a fixed number of rows $m = 1000$ and λ_i's assigned according to scenario 1. In Fig. 2a, we change the number of workers from 10 to 100 and keep the ratio $z/n = 1/4$ fixed. We notice that with the increase of n the completion time of PRAC becomes closer to GC3P. In Fig. 2b, we change the number of workers from 20 to 100 and keep $z = 13$ fixed. We notice that with the increase of n, the effect of the eavesdropper is amortized and the completion time of PRAC becomes closer to C3P. In this setting, PRAC always outperforms Staircase codes.

In Fig. 3, we plot the task completion time as a function of the number of colluding workers. In Fig. 3a, we choose the computing time at the workers according to scenario 1. We change z from 1 to 40 and observe that the completion time of PRAC



a Task completion time as a function of the number of workers with $z = n/4$. **b** Task completion time as a function of the number of workers with $z = 13$.

Fig. 2 Comparison between PRAC, Staircase codes and GC3P in scenario 1 for different values of the number workers and number of colluding workers. We fix the number of rows to $m = 1000$. For each value of the *x*-axis we run 100 experiments and average the results. We observe that the difference between the completion time of PRAC and this of GC3P is large for small values of $n - z$ and decreases with the increase of $n - z$



a Task completion time as a function of the number of colluding workers for $n = 50$. Computing time of the workers are chosen according to scenario 1. **b** Task completion time for $n = 50$ workers and variable z. Computing times of the workers are chosen such that the $n - z$ slowest workers are homogeneous.

Fig. 3 Comparison between PRAC and Staircase codes average completion time as a function of number of colluding workers *z*. We fix the number of rows to $m = 1000$. Both codes are affected by the increase of number of colluding helpers because their runtime is restricted to the slowest $n - z$ workers. We observe that PRAC outperforms Staircase codes except when the $n - z$ slowest workers are homogeneous

deviates from that of GC3P with the increase of z. More importantly, we observe two inflection points of the average completion time of PRAC at $z = 13$ and $z = 37$. Those inflection points are due to the fact that we have 12 fast workers ($\lambda = 9$) and 25 workers with medium speed ($\lambda = 3$) in the system. For $z > 36$, the completion time of Staircase codes becomes less than that of PRAC because the 14 slowest workers are homogeneous. Therefore, pre-allocating the tasks is better than using fountain codes and paying for the overhead of computations. To confirm that Staircase codes always outperforms PRAC when the slowest $n - z$ workers are homogeneous, we run a simulation in which we divide the workers into three clusters. The first cluster consists of $\lfloor z/2 \rfloor$ fast workers ($\lambda = 9$), the second consists of $\lfloor z/2 \rfloor + 1$ workers that are regular ($\lambda = 3$) and the remaining $n - z$ workers are slow ($\lambda = 1$). In Fig. 3b we fix n to 50 and change z from 1 to 40. We observe that Staircase codes always outperform PRAC in this setting. In contrast to non secure C3P, Staircase codes and PRAC are always restricted to the slowest $n - z$ workers and cannot leverage the increase of the number of fast workers. For GC3P, we assume that the fastest workers are eavesdroppers. We note that as expected from Lemma 2, when the fastest workers are assumed to be eavesdroppers the performance of GC3P and PRAC becomes very close.

7.2 Experiments

Setup The master device is a Nexus 5 Android-based smartphone running 6.0.1. The worker devices are Nexus 6Ps running Android 8.1.0. The master device connects to worker devices via Wi-Fi Direct links and the master is the group owner of Wi-Fi Direct group. The implementation of all the algorithms is done using an Android application written in Java. The master and the workers form a star topology where the master device is the center. The devices are placed approximately 10 inches away from each other. The devices communicate via TCP sockets. TCP sockets are wrapped by data output and input stream to send and receive different data types such as integer, double and bytes. A simple communication protocol is built between the master and the workers. The master firstly sends the size of the data followed by the actual data for multiplication. The worker receives the data, process it and sends the acknowledgement and the results back to the master. The master retrieves the results and estimates the processing delay. To allow parallel processing, the master device utilize multi-threading and deals with each worker by independent threads. We denote this type of thread as "worker thread." A main thread at the master controls all the worker threads. For PRAC, the main thread records the round of each worker. All the results from each worker thread are reported to the main thread as well.

The master device is required to complete one matrix multiplication ($y = Ax$) where A is of dimensions $60 \times 10{,}000$ and x is a $10{,}000 \times 1$ vector. We also take $m = b$ i.e., each packet is a row of A. In our implementations the workers are dedicated to the master and do not run background applications or other computing tasks. Therefore, we introduced an artificial delay at the workers following an exponential distribution. The introduced delays serves to emulate real scenarios in which workers would be running other applications in the background. We manipulate the delays in the experiment to analyze the performance of PRAC and other baselines algorithms in the presence of stragglers and validate our theoretical findings. A worker device sends the result to the master after it is done calculating and the introduced delay has passed. Furthermore, we assume that $z = 1$ i.e., there is one unknown worker that is adversarial among all the workers. The experiments are conducted in a lab environment where there are other Wi-Fi networks operating in the background.

Baselines Our PRAC algorithm is compared to three baseline algorithms: (1) Staircase codes that preallocate the tasks based on n, the number of workers, k, the minimum number of workers required to reconstruct the information, and z, the number of colluding workers; (2) GC3P in which we assume the adversarial worker is known and excluded during the task allocation; (3) Non secure C3P in which the security problem is ignored and the master device will utilize every resource without randomness. In this setup we run C3P on $n - z$ workers.

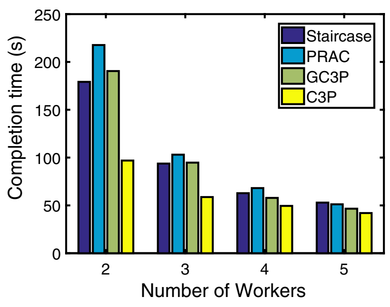


Fig. 4 Completion time as function of the number of workers in homogeneous setup

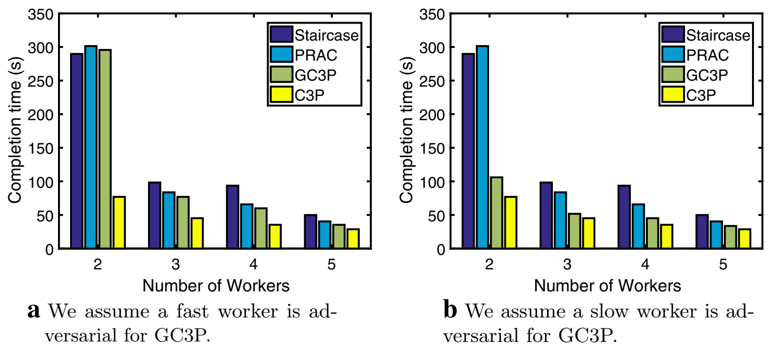
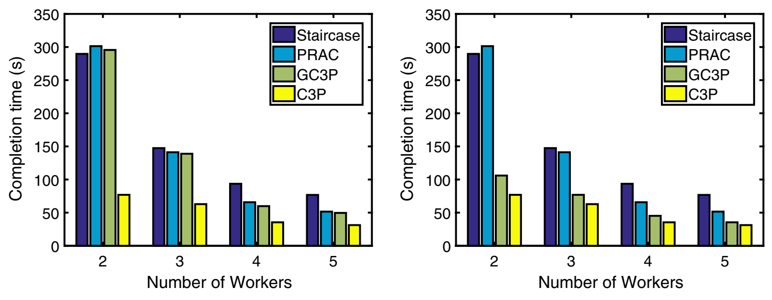


Fig. 5 Completion time as function of the number of workers in heterogeneous setup

Results Figure 4 presents the task completion time with increasing number of workers for the homogeneous setup, i.e., when all the workers have similar computing times. Computing delay for each packet follows an exponential distribution with mean $\mu = 1/\lambda = 3$ s in all workers. C3P performs the best in terms of completion time, but C3P does not provide any privacy guarantees. PRAC outperforms Staircase codes when the number of workers is 5. The reason is that PRAC performs better than Staircase codes in heterogeneous setup, and when the number of workers increases, the system becomes a bit more heterogeneous. GC3P significantly outperforms PRAC in terms of completion time. Yet, it requires a prior knowledge of which worker is adversarial, which is often not available in real world scenarios.

Now, we focus on heterogeneous setup. We group the workers into two groups; fast workers (per task delay follows exponential delay with mean 2 s) and slow workers (per task delay follows exponential distribution with mean 5 s). Figure 5 presents the completion time as a function of number of workers. In this setup, for the n-worker scenario, there are $\lceil \frac{n}{2} \rceil$ fast and $\lfloor \frac{n}{2} \rfloor$ slow workers. The difference between the setups of Fig. 5a, b is that we remove a fast worker (as adversarial) for GC3P in the former, whereas in the latter, we assume that the eavesdropper is a slow worker. As illustrated in Fig. 5, for the 2-worker case, due to the 5% overhead introduced by fountain codes, PRAC performs worse than Staircase code. However, PRAC outperforms Staircase codes in terms of completion time for 3, 4, and 5 worker cases. This is due to the fact that PRAC can utilize results calculated by slow workers more effectively when the number of workers is



a We assume a fast worker is adversarial for GC3P.

b We assume a slow worker is adversarial for GC3P.

Fig. 6 Completion time as function of the number of workers in heterogeneous setup

large. On the other hand, the results computed by slow workers are often discarded in Staircase codes, which is a waste of computation resources. If a fast worker is removed as adversarial for GC3P, the difference between the performance of GC3P and PRAC becomes smaller. This result is intuitive as, in PRAC, the master has to wait for the $(z+1)$st fastest worker to decode $A\mathbf{x}$, which is also the case for GC3P in this setting.

In Fig. 6, we consider the same setup with the exception that for the n-worker scenario, there are $\lceil \frac{n}{2} \rceil$ slow and $\lfloor \frac{n}{2} \rfloor$ fast workers. Staircase codes perform more closely to PRAC in the 3-worker case as compared to Fig. 5 since the setup of Fig. 6 assumes that the n-z=2 slowest workers are homogeneous, whereas in Fig. 5 the $n-z=2$ slowest workers are heterogeneous. Yet, for 5-worker case, PRAC outperforms Staircase codes when comparing to Fig. 5 since PRAC is adaptive to time-varying resources while Staircase codes assigns tasks a priori in a static manner.

Note that in all experiments when $n-z$ slowest workers are homogeneous Staircase codes outperform GC3P and PRAC. This happens because pre-allocating the tasks to the workers avoids the overhead of sub-tasks required by fountain codes and utilizes all the workers to their fullest capacity.

8 Conclusion

The focus of this paper is to develop a secure edge computing mechanism to mitigate the computational bottleneck of IoT devices by allowing these devices to help each other in their computations, with possible help from the cloud if available. Our key tool is the theory of coded computation, which advocates mixing data in computationally intensive tasks by employing erasure codes and offloading these tasks to other devices for computation. Focusing on eavesdropping attacks, we designed a private and rateless adaptive coded computation (PRAC) mechanism considering (1) the privacy requirements of IoT applications and devices, and (2) the heterogeneous and time-varying resources of edge devices. Our proposed PRAC model can provide adequate security and latency guarantees to support real-time computation at the edge. We showed through analysis, MATLAB simulations, and experiments on Android-based smartphones that PRAC outperforms known secure coded computing methods when resources are heterogeneous.

Abbreviations

A: $m \times \ell$ data matrix; A_i: *i*th row block of data matrix *A*; $\beta_{t,i}$: Computation time of the *t*th packet at w_i; ε: Overhead of fountain codes; M: Master; *m*: Number of rows in *A*; *n*: Number of workers; ν: Fountain coded packet of A_i's; PRAC: Private Rateless Adaptive Coded Computing; *R*: Random matrix; RTT_i: average round trip time to send and receive packet p_i; *s*: Secure fountain coded packet; T_i: Time to compute a packet at w_i; $T_{(d)}$: *d*th order statistic of T_i's; *T*: Time spent by M to decode $A\mathbf{x}$; w_i: Worker *i*; $\mathbf{x}$: $\ell \times 1$ attribute vector; *z*: Number of colluding workers.

Author's contributions

This is a joint work between three groups of researchers located at Rutgers University (RB and SER), University of Illinois at Chicago (YX, YK and HS) and the Army Research Lab (VD). VD provided applications for private rateless codes which lead to this research project. RB and SER work on data privacy and have contributed to the privacy part, code construction and privacy analysis, of the proposed scheme. YK and HS proposed the adaptive algorithm to encode and distribute tasks to the workers, and provided the time analysis of such algorithm. YX implemented PRAC on Android devices, smartphones and tablets. RB and YK ran the MATLAB simulations. All authors read and approved the final manuscript.

Funding

Open Access funding enabled and organized by Projekt DEAL. This work was supported in parts by the Army Research Lab (ARL) under Grant W911NF-1820181, National Science Foundation (NSF) under Grants CNS-1801708 and CNS-1801630, and the National Institute of Standards and Technology (NIST) under Grant 70NANB17H188.

Availability of data and materials

Not applicable.

Competing interests

The authors declare that they have no competing interests.

Author details

[1] Department of Electrical and Computer Engineering, Technical University of Munich, 80802 Munich, Germany. [2] Department of Electrical and Computer Engineering, University of Illinois at Chicago, Chicago, IL 60607, USA. [3] Storage Research Group at the Seagate Technology, Shakopee, MN 55379, USA. [4] US Army Research Lab, Aberdeen Proving Ground, MD, USA. [5] Department of Electrical and Computer Engineering, Rutgers University, Piscataway, NJ 08854, USA.

Appendix 1: Hiding the vector x

In machine learning applications, the master runs iterative algorithms in which the vector $\mathbf{x}$ contains information about *A* and needs to be hidden from the workers. We describe how PRAC can be generalized to achieve privacy for both *A* and $\mathbf{x}$. The idea is to divide the *n* workers into two disjoint groups and ask each of them to privately multiply *A* by a vector that is statistically independent of $\mathbf{x}$. In addition, the master should be able to decode $A\mathbf{x}$ from the results of both multiplications. The scheme works as follows. The master divides the workers into two groups of cardinality n_1 and n_2 such that $n_1 + n_2 = n$ and chooses the security parameters $z_1 < n_1$ and $z_2 < n_2$. To hide $\mathbf{x}$, the master generates a random vector $\mathbf{u}$ of same size as $\mathbf{x}$ and sends $\mathbf{x} + \mathbf{u}$ to the first group and $\mathbf{u}$ to the second group. Afterwards, the master applies PRAC on both groups. According to our scheme, the master decodes $A(\mathbf{x} + \mathbf{u})$ and $A\mathbf{u}$ after receiving enough responses from the workers of each group. Hence, the master can decode $A\mathbf{x}$. Note that no information about $\mathbf{x}$ is revealed because it is one-time padded by $\mathbf{u}$. Note that here we assume workers from group 1 do not collude with workers from group 2. The same idea can be generalized to the case where workers from different groups can collude by creating more groups and encoding $\mathbf{x}$ using an appropriate secret sharing scheme. For instance, if the master divides the workers into 3 groups and workers from any 2 different groups can collude, the master encodes $\mathbf{x}$ into $\mathbf{u}_1$, $\mathbf{u}_2$ and $\mathbf{u}_1 + \mathbf{u}_2 + \mathbf{x}$ and sends each vector to a different group.

Appendix 2: Extension of proof of privacy (i.e., Theorem 1)

Since at each round we generate new random matrices, it is enough to study the privacy condition at one round. Consider a given round t of PRAC. Let P_i denote the random variable representing packet p_i sent to worker w_i. For any subset $Z \subset \{1, \dots, n\}, |Z| = z$, denote by P_Z the collection of packets indexed by Z, i.e., $P_Z = \{p_i; i \in Z\}$. We prove that the information theoretic privacy constraint $H(A \mid P_Z) = H(A)$, given in (2), is equivalent to $H(K \mid P_Z, A) = 0$. The proof is standard [73–75] but we reproduce it here for completeness. In what follows, the logarithms in the entropy function are taken base q, where q is a power of prime for which all matrices can be defined in a finite field $\mathbb{F}_q$. We can write,

$$H(A \mid P_Z) = H(A) - H(P_Z) + H(P_Z \mid A) \tag{8}$$

$$= H(A) - H(P_Z) + H(P_Z \mid A) - H(P_Z \mid A, K) \tag{9}$$

$$= H(A) - H(P_Z) + I(P_Z; K \mid A) \tag{10}$$

$$= H(A) - H(P_Z) + H(K \mid A) - H(K \mid P_Z, A) \tag{11}$$

$$= H(A) - H(P_Z) + H(K) - H(K \mid P_Z, A) \tag{12}$$

$$= H(A) - z + z - H(K \mid P_Z, A) \tag{13}$$

$$= H(A) - H(K \mid P_Z, A). \tag{14}$$

Equation (9) follows from the fact that given the data A and the keys $R_1, \dots, R_z$ all packets generated by the master can be decoded, in particular the packets P_Z received by any z workers can be decoded, i.e., $H(P_Z \mid A, K) = 0$. Equation (12) follows because the random matrices are chosen independently from the data matrix A and Eq. (13) follows because PRAC uses z independent random matrices that are chosen uniformly at random from the field $\mathbb{F}_q$. Therefore, since the entropy $H(.)$ is always positive, proving that $H(A|P_Z) = H(A)$ is equivalent to proving that $H(K \mid P_Z, A) = 0$. In other words, we need to prove that the random matrices can be decoded given the collection of packets sent to any z workers and the data matrix A. This is the main reason behind encoding the random matrices using an (n, z) MDS code. We formally prove that $H(K \mid P_Z, A) = 0$ in the proof of Theorem 1. Note from Eq. (12) that for any code to be information theoretically private, $H(K)$ cannot be less then $H(P_Z) = z$. This means that a secure code must use at least z independent random matrices.

Appendix 3: Proof of Lemma 2

We prove the lemma by contradiction. Assume that there exists a private coded computing scheme for distributed linear computation that is secure against z colluding workers and allows the master to decode $A\mathbf{x}$ using the help of the fastest z workers. Without loss of generality, assume that the workers are ordered from the fastest to the slowest, i.e., worker w_1 is the fastest and worker w_n is the slowest. The previous assumption implies that the results sent from the first z workers contain information about $A\mathbf{x}$, otherwise the master would have to wait at least for the $(z+1)$st fastest worker to decode $A\mathbf{x}$. By linearity of the multiplication $A\mathbf{x}$, decoding information about $A\mathbf{x}$ from the results of z workers implies decoding information about A from the packets sent to those z workers. Hence, there exists a set $Z \subset \{1, \ldots, n\}$ of z workers for which $H(A|P_Z) \neq 0$, where P_Z denotes the tasks allocated to a subset of z workers, hence violating the privacy constraint. Therefore, any private coded computing scheme for linear computation limits the master to the speed of the $(z+1)$st fastest worker in order to decode the wanted result.

Appendix 4: Proof of Theorem 3

The total delay for receiving τ_i computed packets from worker w_i is equal to

$$T_i \approx \mathrm{RTT}_i + \tau_i \mathbb{E}[\beta_{t,i}] \approx \tau_i \mathbb{E}[\beta_{t,i}]$$

where RTT_i is the average transmission delay for sending one packet to worker w_i and receiving one computed packet from the worker, $\beta_{t,i}$ is the computation time spent on multiplying packet $p_{t,i}$ by $\mathbf{x}$ at worker w_i, and the average $\mathbb{E}[\beta_{t,i}]$ is taken over all τ_i packets. The reason is that PRAC is a dynamic algorithm that sends packets to each worker w_i with the interval of $\mathbb{E}[\beta_{t,i}]$ between each two consecutive packets and it utilizes the resources of workers fully [76]. The reason behind counting only one round-trip time (RTT) in T_i is that in PRAC, the packets are being transmitted to the workers while the previously transmitted packets are being computed at the worker. Therefore, in the overall delay only one RTT_i is required for sending the first packet $p_{1,i}$ to worker w_i and receiving the last computed packet $p_{\tau_i,i}\mathbf{x}$ at the master. To approximate the total delay, we assume that the transmission delay of one packet is negligible compared to the computing delay of all τ_i packets, which is a valid assumption in practice for IoT-devices at the edge.

On the other hand, in PRAC, the master stops sending packets to workers as soon as it collectively receives $b+\epsilon$ computed packets from the $n-z$ slowest workers (note that $b+\epsilon$ is the number of computed packets required for successful decoding, where ϵ is the overhead due to fountain Coding), i.e., $\sum_{i=z+1}^{n} \tau_i = b+\epsilon$. Note that the z fastest workers are assigned for computing the keys as described in the previous sections. Due to efficiently using the resources of workers by PRAC, all $n-z$ workers will finish computing τ_i packets approximately at the same time, i.e., $T_{\mathrm{PRAC}} \approx T_i \approx \tau_i \mathbb{E}[\beta_{t,i}]$, $i = z+1, \ldots, n$. By replacing τ_i with $\frac{T_{\mathrm{PRAC}}}{\mathbb{E}[\beta_{t,i}]}$ in $\sum_{i=z+1}^{n} \tau_i = b+\epsilon$, we can show that $T_{\mathrm{PRAC}} \approx \frac{b+\epsilon}{\sum_{i=z+1}^{n} 1/\mathbb{E}[\beta_{t,i}]}$. Note that the approximated value approaches the exact value by increasing b. The reason is that the workers' efficiency increases with increasing b

Received: 10 May 2020 Accepted: 15 December 2020
Published online: 22 January 2021

References

1. R. Bitar, Y. Xing, Y. Keshtkarjahromi, V. Dasari, S. El Rouayheb, H. Seferoglu, PRAC: private and rateless adaptive coded computation at the edge, in *SPIE Defense + Commercial Sensing*, vol. 11013 (2019). https://doi.org/10.1117/12.2519768
2. Y. Keshtkarjahromi, Y. Xing, H. Seferoglu, Dynamic heterogeneity-aware coded cooperative computation at the edge, in *IEEE 26th International Conference on Network Protocols (ICNP)* (2018)
3. R. Bitar, P. Parag, S. El Rouayheb, Minimizing latency for secure distributed computing, in *IEEE International Symposium on Information Theory (ISIT)*, pp. 2900–2904 (2017)
4. S. Li, M.A. Maddah-Ali, A.S. Avestimehr, A unified coding framework for distributed computing with straggling servers, in *Globecom Workshops (GC Wkshps)* (IEEE, 2016), pp. 1–6
5. S. Dutta, V. Cadambe, P. Grover, Coded convolution for parallel and distributed computing within a deadline. arXiv preprint arXiv:1705.03875 (2017)
6. Y. Yang, P. Grover, S. Kar, Computing linear transformations with unreliable components. IEEE Trans. Inf. Theory **63**, 3729–3756 (2017)
7. W. Halbawi, N. Azizan-Ruhi, F. Salehi, B. Hassibi, Improving distributed gradient descent using reed-solomon codes. arXiv preprint arXiv:1706.05436 (2017)
8. Q. Yu, M. Maddah-Ali, S. Avestimehr, Polynomial codes: an optimal design for high-dimensional coded matrix multiplication, in *Advances in Neural Information Processing Systems* (2017), pp. 4403–4413
9. S. Dutta, V. Cadambe, P. Grover, Short-dot: computing large linear transforms distributedly using coded short dot products, in *29th Annual Conference on Neural Information Processing Systems (NIPS)* (2016), pp. 2092–2100
10. R. Tandon, Q. Lei, A.G. Dimakis, N. Karampatziakis, Gradient coding: avoiding stragglers in distributed learning, in *International Conference on Machine Learning* (2017), pp. 3368–3376
11. S. Li, M.A. Maddah-Ali, A.S. Avestimehr, Fundamental tradeoff between computation and communication in distributed computing, in *IEEE International Symposium on Information Theory (ISIT)* (2016)
12. K. Lee, M. Lam, R. Pedarsani, D. Papailiopoulos, K. Ramchandran, Speeding up distributed machine learning using codes. IEEE Trans. Inf. Theory **64**(3), 1514–1529 (2018)
13. C. Karakus, Y. Sun, S. Diggavi, W. Yin, Straggler mitigation in distributed optimization through data encoding, in *Advances in Neural Information Processing Systems* (2017), pp. 5434–5442
14. M.F. Aktas, P. Peng, E. Soljanin, Effective straggler mitigation: which clones should attack and when? ACM SIGMETRICS Perform. Eval. Rev. **45**(2), 12–14 (2017)
15. S.N. Shirazi, A. Gouglidis, A. Farshad, D. Hutchison, The extended cloud: review and analysis of mobile edge computing and fog from a security and resilience perspective. IEEE J. Sel. Areas Commun. **35**(11), 2586–2595 (2017)
16. N. Abbas, Y. Zhang, A. Taherkordi, T. Skeie, Mobile edge computing: a survey. IEEE Internet Things J. **5**(1), 450–465 (2017)
17. R. Roman, J. Lopez, M. Mambo, Mobile edge computing, Fog et al.: a survey and analysis of security threats and challenges. Future Gener. Comput. Syst. **78**, 680–698 (2018)
18. M. Luby, Lt codes, in *The 43rd Annual IEEE Symposium on Foundations of Computer Science (FOCS)* (2002), pp. 271–280
19. A. Shokrollahi, Raptor codes. IEEE/ACM Trans. Netw. (TON) **14**(SI), 2551–2567 (2006)
20. D.J. MacKay, Fountain codes. IEE Proc. Commun. **152**(6), 1062–1068 (2005)
21. S. Lin, D. Costello, *Error-Correcting Codes* (Prentice-Hall Inc, Upper Saddle River, 1983)
22. F.J. MacWilliams, N.J.A. Sloane, *The Theory of Error-Correcting Codes* (Elsevier, Amsterdam, 1977)
23. Q. Yu, S. Li, N. Raviv, S.M.M. Kalan, M. Soltanolkotabi, S.A. Avestimehr, Lagrange coded computing: optimal design for resiliency, security, and privacy, in *22nd International Conference on Artificial Intelligence and Statistics*, PMLR (2019), pp. 1215–1225
24. M. Kim, H. Yang, J. Lee, Private coded computation for machine learning. arXiv preprint arXiv:1807.01170 (2018)
25. J. So, B. Guler, A.S. Avestimehr, P. Mohassel, CodedPrivateML: a fast and privacy-preserving framework for distributed machine learning. arXiv preprint arXiv:1902.00641 (2019)
26. R. Bitar, P. Parag, S. El Rouayheb, Minimizing latency for secure coded computing using secret sharing via Staircase codes. IEEE Trans. Commun. **68**, 4609–4619 (2020)
27. R.G. D'Oliveira, S.E. Rouayheb, D. Heinlein, D. Karpuk, Notes on communication and computation in secure distributed matrix multiplication, in *IEEE International Workshop on Privacy and Security for Information Systems (WPS)* (2020)
28. A. Mallick, M. Chaudhari, U. Sheth, G. Palanikumar, G. Joshi, Rateless codes for near-perfect load balancing in distributed matrix-vector multiplication. Proc. ACM Meas. Anal. Comput. Syst. **3**(3), 1–40 (2019)
29. H.T. Dinh, C. Lee, D. Niyato, P. Wang, A survey of mobile cloud computing: architecture, applications, and approaches. Wirel. Commun. Mob. Comput. **13**(18), 1587–1611 (2013)
30. N. Fernando, S.W. Loke, W. Rahayu, Mobile cloud computing: a survey. Future Gener. Comput. Syst. **29**(1), 84–106 (2013)
31. Z. Sanaei, S. Abolfazli, A. Gani, R. Buyya, Heterogeneity in mobile cloud computing: taxonomy and open challenges. IEEE Commun. Surv. Tutor. **16**(1), 369–392 (2014). https://doi.org/10.1109/SURV.2013.050113.00090
32. Y. Geng, W. Hu, Y. Yang, W. Gao, G. Cao, Energy-efficient computation offloading in cellular networks, in *IEEE International Conference on Network Protocols (ICNP)* (2015)
33. R.K. Lomotey, R. Deters, Architectural designs from mobile cloud computing to ubiquitous cloud computing-survey, in *IEEE World Congress on Services* (2014)
34. T. Penner, A. Johnson, B.V. Slyke, M. Guirguis, Q. Gu, Transient clouds: assignment and collaborative execution of tasks on mobile devices, in *IEEE Global Communications Conference* (2014)
35. J. Dean, S. Ghemawat, Mapreduce: simplified data processing on large clusters. Commun. ACM **51**(1), 107–113 (2008)
36. J. Dean, L.A. Barroso, The tail at scale. Commun. ACM **56**(2), 74–80 (2013)
37. B. Recht, C. Re, S. Wright, F. Niu, Hogwild: a lock-free approach to parallelizing stochastic gradient descent, in *Advances in Neural Information Processing Systems* (2011), pp. 693–701

38. J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, A. Senior, P. Tucker, K. Yang, Q.V. Le, et al., Large scale distributed deep networks, in *Advances in Neural Information Processing Systems* (2012), pp. 1223–1231
39. L. Huang, S. Pawar, H. Zhang, K. Ramchandran, Codes can reduce queueing delay in data centers, in *IEEE International Symposium on Information Theory (ISIT)* (2012)
40. G. Joshi, Y. Liu, E. Soljanin, Coding for fast content download, in *50th Annual Allerton Conference on Communication, Control, and Computing* (2012)
41. G. Liang, U.C. Kozat, TOFEC: achieving optimal throughput-delay trade-off of cloud storage using erasure codes, in *IEEE International Conference on Computer Communications*
42. S. Kadhe, E. Soljanin, A. Sprintson, Analyzing the download time of availability codes, in *IEEE International Symposium on Information Theory (ISIT)* (2015)
43. P. Peng, E. Soljanin, On distributed storage allocations of large files for maximum service rate. arXiv preprint arXiv:1808.07545 (2018)
44. Q. Yu, M.A. Maddah-Ali, A.S. Avestimehr, Straggler mitigation in distributed matrix multiplication: fundamental limits and optimal coding. arXiv preprint arXiv:1801.07487 (2018)
45. R.K. Maity, A.S. Rawat, A. Mazumdar, Robust gradient descent via moment encoding with ldpc codes. arXiv preprint arXiv:1805.08327 (2018)
46. L. Chen, Z. Charles, D. Papailiopoulos et al., Draco: robust distributed training via redundant gradients. arXiv preprint arXiv:1803.09877 (2018)
47. S. Kiani, N. Ferdinand, S.C. Draper, Exploitation of stragglers in coded computation, in *IEEE International Symposium on Information Theory (ISIT)* (IEEE, 2018), pp. 1988–1992
48. E. Ozfaturay, D. Gunduz, S. Ulukus, Speeding up distributed gradient descent by utilizing non-persistent stragglers. arXiv preprint arXiv:1808.02240 (2018)
49. M. Ye, E. Abbe, Communication-computation efficient gradient coding. arXiv preprint arXiv:1802.03475 (2018)
50. N. Ferdinand, S. Draper, Anytime stochastic gradient descent: a time to hear from all the workers. arXiv preprint arXiv:1810.02976 (2018)
51. S. Li, S.M.M. Kalan, A.S. Avestimehr, M. Soltanolkotabi, Near-optimal straggler mitigation for distributed gradient methods, in *IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)* (IEEE, 2018), pp. 857–866
52. N. Raviv, I. Tamo, R. Tandon, A.G. Dimakis, Gradient coding from cyclic MDS codes and expander graphs. arXiv preprint arXiv:1707.03858 (2017)
53. A. Severinson, A.G. i Amat, E. Rosnes, Block-diagonal coding for distributed computing with straggling servers, in *IEEE Information Theory Workshop (ITW)* (2017), pp. 464–468
54. U. Sheth, S. Dutta, M. Chaudhari, H. Jeong, Y. Yang, J. Kohonen, T. Roos, P. Grover, An application of storage-optimal matdot codes for coded matrix multiplication: fast k-nearest neighbors estimation. arXiv preprint arXiv:1811.11811 (2018)
55. R.G. D'Oliveira, S.E. Rouayheb, D. Karpuk, Gasp codes for secure distributed matrix multiplication. arXiv preprint arXiv:1812.09962 (2018)
56. W.-T. Chang, R. Tandon, On the capacity of secure distributed matrix multiplication, in *IEEE Global Communications Conference (GLOBECOM)* (IEEE, 2018), pp. 1–6
57. J. Kakar, S. Ebadifar, A. Sezgin, Rate-efficiency and straggler-robustness through partition in distributed two-sided secure matrix computation. arXiv preprint arXiv:1810.13006 (2018)
58. H. Yang, J. Lee, Secure distributed computing with straggling servers using polynomial codes. IEEE Trans. Inf. Forensics Secur. **14**(1), 141–150 (2018)
59. Q. Yu, N. Raviv, J. So, A.S. Avestimehr, Lagrange coded computing: Optimal design for resiliency, security and privacy. arXiv preprint arXiv:1806.00939 (2018)
60. R. Cramer, I.B. Damgrd, J.B. Nielsen, *Secure Multiparty Computation and Secret Sharing*, 1st edn. (Cambridge University Press, New York, 2015)
61. H. Takabi, E. Hesamifard, M. Ghasemi, Privacy preserving multi-party machine learning with homomorphic encryption, in *29th Annual Conference on Neural Information Processing Systems (NIPS)* (2016)
62. R. Hall, S.E. Fienberg, Y. Nardi, Secure multiple linear regression based on homomorphic encryption. J. Off. Stat. **27**(4), 669 (2011)
63. S. Gade, N.H. Vaidya, Private learning on networks: part II. arXiv preprint arXiv:1703.09185 (2017)
64. B. Chor, O. Goldreich, E. Kushilevitz, M. Sudan, Private information retrieval, in *IEEE 36th Annual Foundations of Computer Science (FOCS)* (1995), pp. 41–50
65. R. Bitar, S. El Rouayheb, Staircase-PIR: universally robust private information retrieval, in *IEEE Information Theory Workshop (ITW)* (2018), pp. 1–5
66. Y. Keshtkarjahromi, R. Bitar, V. Dasari, S. El Rouayheb, H. Seferoglu, Secure coded cooperative computation at the heterogeneous edge against byzantine attacks, in *IEEE Global Communication Conference (GLOBECOM)* (2019)
67. G.A. Seber, A.J. Lee, *Linear Regression Analysis*, vol. 329 (Wiley, Hoboken, 2012)
68. J.A. Suykens, J. Vandewalle, Least squares support vector machine classifiers. Neural Process. Lett. **9**(3), 293–300 (1999)
69. J. Lacan, V. Roca, J. Peltotalo, S. Peltotalo, Reed-solomon forward error correction (FEC) schemes. Technical report (2009)
70. G. Souaki, K. Halim, Random number generation based on MCU sources for IOT application, in *International Conference on Advanced Technologies for Signal and Image Processing (ATSIP)* (2017), pp. 1–6
71. N. Onizawa, S. Mukaida, A. Tamakoshi, H. Yamagata, H. Fujita, T. Hanyu, High-throughput/low-energy MTJ-based true random number generator using a multi-voltage/current converter. IEEE Trans. Very Large Scale Integr. (VLSI) Syst. **28**, 2171–2181 (2020)
72. R. Bitar, Y. Xing, Y. Keshtkarjahromi, V. Dasari, S. El Rouayheb, H. Seferoglu, Private and rateless adaptive coded matrix-vector multiplication. arXiv preprint arXiv:1909.12611 (2019)

73. N.B. Shah, K.V. Rashmi, P.V. Kumar, Information-theoretically secure regenerating codes for distributed storage, in *Proceedings of IEEE Global Communications Conference* (2011)
74. S.E. Rouayheb, E. Soljanin, A. Sprintson, Secure network coding for wiretap networks of type II. IEEE Trans. Inf. Theory **58**(3), 1361–1371 (2012). https://doi.org/10.1109/TIT.2011.2173631
75. R. Bitar, S. El Rouayheb, Staircase codes for secret sharing with optimal communication and read overheads. IEEE Trans. Inf. Theory **64**, 933–943 (2017). https://doi.org/10.1109/TIT.2017.2723019
76. Y. Keshtkarjahromi, Y. Xing, H. Seferoglu, Dynamic heterogeneity-aware coded cooperative computation at the edge. arXiv preprint arXiv:1801.04357v3 (2018)

Publisher's Note