Dynamic Heterogeneity-Aware Coded Cooperative Computation at the Edge

Yasaman Keshtkarjahromi, Yuxuan Xing, Hulya Seferoglu
ECE Department, University of Illinois at Chicago
ykesht2@uic.edu, yxing7@uic.edu, hulya@uic.edu

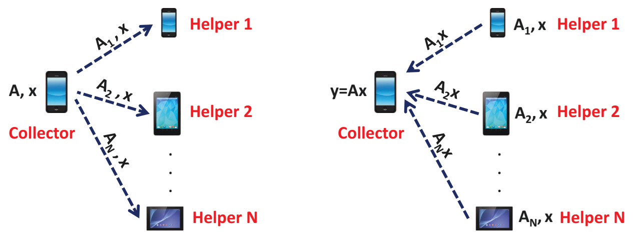
***Abstract*—Cooperative computation is a promising approach for localized data processing at the edge, *e.g.,* for Internet of Things (IoT). Cooperative computation advocates that computationally intensive tasks in a device could be divided into sub-tasks, and offloaded to other devices or servers in close proximity. However, exploiting the potential of cooperative computation is challenging mainly due to the heterogeneous and time-varying nature of edge devices. Coded computation, which advocates mixing data in sub-tasks by employing erasure codes and offloading these sub-tasks to other devices for computation, is recently gaining interest, thanks to its higher reliability, smaller delay, and lower communication costs. In this paper, we develop a coded cooperative computation framework, which we name Coded Cooperative Computation Protocol (`C3P`), by taking into account the heterogeneous resources of edge devices. `C3P` dynamically offloads coded sub-tasks to helpers and is adaptive to time-varying resources. We show that (i) task completion delay of `C3P` is very close to optimal coded cooperative computation solutions, (ii) the efficiency of `C3P` in terms of resource utilization is higher than 99%, (iii) `C3P` improves task completion delay significantly as compared to baselines via both simulations and in a testbed consisting of real Android-based smartphones.**

I. Introduction

Data processing is crucial for many applications at the edge including Internet of Things (IoT), but it could be computationally intensive and not doable if devices operate individually. 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 localized processing crucial.

Cooperative computation is a promising approach for edge computing, where computationally intensive tasks in a device (collector device) could be offloaded to other devices (helpers) in close proximity as illustrated in Fig. 1. These devices could be other IoT or mobile devices, local servers, or fog at the edge of the network [1], [2]. However, exploiting the potential of cooperative computation is challenging mainly due to the heterogeneous and time-varying nature of the devices at the edge. Indeed, these devices may have different and time-varying computing power and energy resources, and could be mobile. Thus, our goal is to develop a dynamic, adaptive, and heterogeneity-aware cooperative computation framework by taking into account the heterogeneity and time-varying nature of devices at the edge.

This work was supported in part by ARL grant W911NF-17-1-0032, NSF grant CNS-1801708, and U.S. Department of Commerce, National Institute of Standards and Technology award 70NANB17H188.



(a) Offloading sub-tasks from a collector to helpers

(b) Helpers send computed sub-tasks back to the collector

Fig. 1. Cooperative computation to compute $\mathbf{y} = A\mathbf{x}$. (a) Matrix A is divided into sub-matrices $A_1, A_2, ..., A_N$. Each sub-matrix along with the vector $\mathbf{x}$ is transmitted from the collector to one of the helpers. (b) Each helper computes the multiplication of its received sub-matrix with vector $\mathbf{x}$ and sends the computed value back to the collector.

We focus on the computation of linear functions. In particular, we assume that the collector's data is represented by a large matrix A and it wishes to compute the product $\mathbf{y} = A\mathbf{x}$, for a given vector $\mathbf{x}$, Fig. 1. In fact, matrix multiplication forms the atomic function computed over many iterations of several signal processing, machine learning, and optimization algorithms, such as gradient descent based algorithms, classification algorithms, etc. [3], [4], [5], [6].

In cooperative computation setup, matrix A is divided into sub-matrices $A_1, A_2, ..., A_N$ and each sub-matrix along with the vector $\mathbf{x}$ is transmitted from the collector to one of the helpers, Fig. 1(a). Helper n computes $A_n\mathbf{x}$, and transmits the computed result back to the collector, Fig. 1(b), who can process all returned computations to obtain the result of its original task; *i.e.,* the calculation of $\mathbf{y} = A\mathbf{x}$.

Coding in computation systems is recently gaining interest in large scale computing environments, and it advocates higher reliability and smaller delay [3]. In particular, coded computation (*e.g.,* by employing erasure codes) mixes data in sub-tasks and offloads these coded sub-tasks for computation, which improves delay and reliability. The following canonical example inspired from [3] demonstrates the effectiveness of coded computation.

Example 1: Let us consider that a collector device would like to calculate $\mathbf{y} = A\mathbf{x}$ with the help of three helper devices (helper 1, helper 2, and helper 3), where the number of rows in A is 6. Let us assume that each helper has a different runtime; helper 1 computes each row in 1 unit time, while the second and the third helpers require 2 and 10 units of time for computing one row, respectively. Assuming that these runtimes are random and not known a priori, one may divide A to three sub-matrices; A_1, A_2, and A_3; each with 2 rows. Thus, the completion time of these sub-matrices becomes 2, 4, and 20 at helpers 1, 2, and 3, respectively. Since the collector should receive all the calculated sub-matrices to compute its original task; *i.e.,* $\mathbf{y} = A\mathbf{x}$, the total task completion delay becomes $\max(2, 4, 20) = 20$.

As seen, helper 3 becomes a bottleneck in this scenario, which can be addressed using coding. In particular, A could be divided into two sub-matrices A_1 and A_2; each with 3 rows. Then, A_1 and A_2 could be offloaded to helpers 1 and 2, and $A_1 + A_2$ could be offloaded to helper 3. In this setup, runtimes become 3, 6, and 30 at helpers 1, 2, and 3, respectively. However, since the collector requires reply from only two helpers to compute $\mathbf{y} = A\mathbf{x}$ thanks to coding, the total task completion delay becomes $\max(3, 6) = 6$. As seen, the task completion delay reduces to 6 from 20 with the help of coding. $\square$

The above example demonstrates the benefit of coding for cooperative computation. However, offloading sub-tasks with equal sizes to all helpers, without considering their heterogeneous resources is inefficient. Let us consider the same setup in Example 1. If A_1 with 4 rows and A_2 with 2 rows are offloaded to helper 1 and helper 2, respectively, and helper 3 is not used, the task completion delay becomes $\max(4, 4) = 4$, which is the smallest possible delay in this example. Furthermore, the resources of helper 3 are not wasted, which is another advantage of taking into account the heterogeneity as compared to the above example. As seen, it is crucial to divide and offload matrix A to helpers by taking into account the heterogeneity of resources.

Indeed, a code design mechanism under such a heterogeneous setup is developed in [7], where matrix A is divided, coded, and offloaded to helpers by taking into account heterogeneity of resources. However, available resources at helpers are generally not known by the collector a priori and may vary over time, which is not taken into account in [7]. For example, the runtime of helper 1 in Example 1 may increase from 1 to 20 while computing (*e.g.,* it may start running another computationally intensive task), which would increase the total task completion delay. Thus, it is crucial to design a coded cooperation framework, which is dynamic and adaptive to heterogeneous and time-varying resources, which is the goal of this paper.

In this paper, we design a coded cooperative computation framework for edge computing. In particular, we design a Coded Cooperative Computation Protocol (`C3P`), which packetizes rows of matrix A into packets, codes these packets using Fountain codes, and determines how many coded packets each helper should compute dynamically over time. We provide theoretical analysis of `C3P`'s task completion delay and efficiency, and evaluate its performance via simulations as well as in a testbed consisting of real Android-based smartphones as compared to baselines. The following are the key contributions of this work:

- We formulate the coded cooperative computation problem as an optimization problem. We investigate the non-ergodic and static solutions of this problem. As a dynamic solution to the optimization problem, we develop a coded cooperative computation protocol (`C3P`), which is based on Automatic Repeat reQuest (ARQ) mechanism. In particular, a collector device offloads coded sub-tasks to helpers gradually, and receives Acknowledgment (ACK) after each sub-task is computed. Depending on the time difference between offloading a sub-task to a helper and its ACK, the collector estimates the runtime of the helpers, and offloads more/less tasks accordingly. This makes `C3P` dynamic and adaptive to heterogeneous and time-varying resources at helpers.
- We characterize the performance of `C3P` as compared to the non-ergodic and static solutions, and show that (i) the gap between the task completion delays of `C3P` and the non-ergodic solution is finite even for large number of sub-tasks, *i.e.,* $R \to \infty$, and (ii) the task completion delay of `C3P` is approximately equal to the static solution for large numbers of sub-tasks. We also analyze the efficiency of `C3P` in each helper in closed form, where the efficiency metric represents the effective utilization of resources at each helper.
- We evaluate `C3P` via simulations as well as in a testbed consisting of real Android-based smartphones and show that (i) `C3P` improves task completion delay significantly as compared to baselines, and (ii) the efficiency of `C3P` in terms of resource utilization is higher than 99%.

The structure of the rest of this paper is as follows. Section II presents the coded cooperative computation problem formulation. Section III presents the ergodic and static solutions to coded cooperative computation problem and the design of `C3P`. Section IV provides the performance analysis of `C3P`. Section V presents the performance evaluation of `C3P`. Section VI presents related work. Section VII concludes the paper.

II. Problem Formulation

Setup. We consider a setup shown in Fig. 1, where the collector device offloads its task to helpers in the set $\mathcal{N}$ (where $N = |\mathcal{N}|$) via device-to-device (D2D) links such as Wi-Fi Direct and/or Bluetooth. In this setup, all devices could potentially be mobile, so the encounter time of the collector with helpers varies over time. *I.e.,* the collector can connect to less than N helpers at a time.

Application. As we described in Section I, we focus on computation of linear functions; *i.e.,* the collector wishes to compute $\mathbf{y} = A\mathbf{x}$ where $A = (a_{i,j}) \in \mathbb{R}^{R \times R}$, and $\mathbf{x} = (x_{i,j}) \in \mathbb{R}^{R \times 1}$. Our goal is to determine sub-matrix $A_n = (a_{i,j}) \in \mathbb{R}^{r_n \times R}$ that will be offloaded to helper n, where r_n is an integer.

Coding Approach. We use Fountain codes [8], [9], which are ideal in our dynamic coded cooperation framework thanks to their rateless property, low encoding and decoding complexity, and low overhead. In particular, the encoding and decoding complexity of Fountain codes could be as low as $O(R\log(R))$ for LT codes and $O(R)$ for Raptor codes and the overhead, *i.e.,* K, could be as low as 5% [10]. We note that Fountain codes perform better than (i) repetition codes thanks to randomization of sub-tasks by mixing them, (ii) maximum distance separable (MDS) codes as MDS codes require a priori task allocation (due to their block coding nature) and are not suitable for the dynamic and adaptive framework that we would like to develop, and (iii) network coding as the decoding complexity of network coding is too high [11], which introduces too much computation overhead at the collector which obsoletes the computation offloading benefit.

Packetization. In particular, we packetize each row of A into a packet and create R packets; $\Gamma = \{\rho_1, \rho_2, \ldots, \rho_R\}$. These packets are used to create Fountain coded packets, where ν_i is the ith coded packet. The coded packet ν_i is transmitted to a helper, where the helper computes the multiplication of $\nu_i \mathbf{x}$ and sends the result back to the collector. $R + K$ coded computed packets are required at the collector to decode the coded packets, where K is the coding overhead. Let $p_{n,i}$ be the jth coded packet generated by the collector and the ith coded packet transmitted to helper n; $p_{n,i} = \nu_j, j \geq i$.

Delay Model. Each transmitted packet $p_{n,i}$ experiences transmission delay between the collector and helper n as well as computing delay at helper n. Also, the computed packet $p_{n,i}\mathbf{x}$ experiences transmission delay while transmitted from helper n to the collector. The average round trip time (RTT) of sending a packet to helper n and receiving the computed packet, is characterized as RTT_n^{data}. The runtime of packet $p_{n,i}$ at helper n is a random variable denoted by $\beta_{n,i}$.[1] Assuming that r_n packets are offloaded to helper n, the total task completion delay for helper n to receive r_n coded packets, compute them, and send the results back to the collector becomes D_n, which is expressed as $D_n = RTT_n^{\text{data}} + \sum_{i=1}^{r_n} \beta_{n,i}$. Note that RTT_n^{data} in this formulation is due to transmitting the first packet $p_{n,1}$ and receiving the last computed packet $p_{n,r_n}\mathbf{x}$. The other packets can be transmitted while helpers are busy with processing packets; it is why we do not sum RTT_n^{data} across packets.

Problem Formulation. Our goal is to determine the task offloading set $\mathcal{R} = \{r_1, \ldots, r_N\}$ that minimizes the total task completion delay, *i.e.,* we would like to dynamically determine $\mathcal{R}$ that solves the following optimization problem:

$$\min_{\mathcal{R}} \max_{n \in \mathcal{N}} D_n$$

$$\text{subject to } \sum_{n=1}^{N} r_n = R, r_n \in \mathbb{N}, \forall n \in \mathcal{N}. \tag{1}$$

The objective of the optimization problem in (1) is to minimize the maximum of per helper task completion delays, which is equal to $\max_{n \in \mathcal{N}} D_n$, as helpers compute their tasks in parallel. The constraint in (1) is a task conservation constraint that guarantees that resources of helpers are not wasted, *i.e.,* the sum of the received computed tasks from all helpers is equal to the number of rows of matrix A. Note that this constraint is possible thanks to coding.[2] As we mentioned earlier, $R+K$ coded computed packets are required at the collector to decode the coded packets when we use Fountain codes. The constraint in (1) guarantees this requirement in an idealized scenario assuming that $K = 0$. The constraint $r_n \in \mathbb{N}$ makes sure that the number of tasks r_n is an integer. The solution of (1) is challenging as (i) $D_n = RTT_n^{\text{data}} + \sum_{i=1}^{r_n} \beta_{n,i}$ is a random variable and not known a priori, and (ii) it is an integer programming problem.

III. Problem Solution & C3P Design

In this section, we investigate the solution of (1) for non-ergodic, static, and dynamic setups.

A. Non-Ergodic Solution

Let us assume that the solution of (1) is

$$T^{\text{best}} = \max_{n \in \mathcal{N}} \Big(RTT_n^{\text{data}} + \sum_{i=1}^{r_n^{\text{best}}} \beta_{n,i} \Big), \tag{2}$$

where $r_n^{\text{best}} = \text{argmin}_{r_n \in \mathbb{N}} \max_{n \in \mathcal{N}} \Big(RTT_n^{\text{data}} + \sum_{i=1}^{r_n} \beta_{n,i} \Big)$. We note that (2) is a non-ergodic solution as it requires the perfect knowledge of $\beta_{n,i}$ a priori. Although we do not have a compact solution of T^{best}, the solution in (2) will behave as a performance benchmark for our dynamic and adaptive coded cooperative computation framework in Section IV-A.

B. Static Solution

We assume that RTT_n^{data} becomes negligible as compared to $\sum_{i=1}^{r_n} \beta_{n,i}$. This assumption holds in practical scenarios with large R, and/or when transmission delay is smaller than processing delay. Then, D_n can be approximated as $\sum_{i=1}^{r_n} \beta_{n,i}$, and the optimization problem in (1) becomes

$$\min_{\mathcal{R}} \max_{n \in \mathcal{N}} \sum_{i=1}^{r_n} \beta_{n,i}$$

$$\text{subject to } \sum_{n=1}^{N} r_n = R, r_n \in \mathbb{N}, \forall n \in \mathcal{N}. \tag{3}$$

As a static solution, we solve the expected value of the objective function in (3) by relaxing the integer constraint, *i.e.,*

[1]Our framework is compatible with any delay distribution, but for the sake of characterizing the efficiency of our algorithm, and simulating its task completion delay, we use shifted exponential distribution in Sections IV-D and V.

[2]We note that the optimal computation offloading problem, when coding is not employed, is formulated as $\min_{\Gamma_n} \max_{n \in \mathcal{N}} (RTT_n^{\text{data}} + \sum_{i=1}^{|\Gamma_n|} \beta_{n,i})$ subject to $\cup_{n=1}^{N} \Gamma_n = \Gamma$ where $\Gamma_n \subset \Gamma$ is the set of packets offloaded to helper n. As seen, the optimization problem in (1) is more tractable as compared to this problem thanks to employing Fountain codes.

$r_n \in \mathbb{N}$. The expected value of the objective function of (3) is expressed as $E[\max_{n\in\mathcal{N}} \sum_{i=1}^{r_n} \beta_{n,i}]$, which is greater than or equal to $\max_{n\in\mathcal{N}} \sum_{i=1}^{r_n} E[\beta_{n,i}] = \max_{n\in\mathcal{N}} r_n E[\beta_{n,i}]$ (noting that $\max(.)$ is a convex function, so $E[\max(.)] \geq \max(E[.])$), where expectation is across the packets. Assuming that the average task completion delay is $T = E[\max_{n\in\mathcal{N}} \sum_{i=1}^{r_n} \beta_{n,i}] \geq \max_{n\in\mathcal{N}} r_n \; E[\beta_{n,i}]$, (3) is converted to

$$\begin{aligned} &\min_{\mathcal{R}} \quad T \\ &\text{subject to } r_n E[\beta_{n,i}] \leq T, \forall n \in \mathcal{N} \\ &\qquad \sum_{n=1}^{N} r_n = R. \end{aligned} \tag{4}$$

We solve (4) using Lagrange relaxation (we omit the steps of the solution as it is straightforward); the optimal task offloading policy becomes

$$r_n^{\text{static}} = \frac{R}{E[\beta_{n,i}] \sum_{n=1}^{N} \frac{1}{E[\beta_{n,i}]}}, \tag{5}$$

and the optimal task completion delay becomes $T^{\text{static}} = \frac{R}{\sum_{n=1}^{N} \frac{1}{E[\beta_{n,i}]}}$. Although the solution in (5) is an optimal solution of (4), the algorithm that offloads r_n^{static} sub-tasks to helper n a priori (static allocation) loses optimality as it is not adaptive to the time-varying nature of resources (*i.e.,* $\beta_{n,i}$). Next, we introduce our Coded Cooperative Computation Protocol (C3P) that is dynamic and adaptive to time-varying resources and approaches to the optimal solution in (5) with increasing R.

C. Dynamic Solution: C3P

We consider the system setup in Fig. 1, where the collector connects to N helpers. In this setup, the collector device offloads coded packets gradually to helpers, and receives two ACKs for each packet; one confirming the receipt of the packet by the helper, and the second one (piggybacked to the computed packet $p_{n,i}\mathbf{x}$) showing that the packet is computed by the helper. Inspired by ARQ mechanisms [12], the collector transmits more/less coded packets based on the frequency of the received ACKs.

In particular, we define the transmission time interval $TTI_{n,i}$ as the time interval between sending two consecutive packets, $p_{n,i}$ and $p_{n,i+1}$, to helper n by the collector. The goal of our mechanism is to determine the best $TTI_{n,i}$ that reduces the task completion delay and increases helper efficiency (*i.e.,* exploiting the full potential of the helpers while not overloading them).

$TTI_{n,i}$ in an ideal scenario. Let $Tx_{n,i}$ be the time that $p_{n,i}$ is transmitted from the collector to helper n, $Tc_{n,i}$ be the time that helper n finishes computing $p_{n,i}$, and $Tr_{n,i}$ be the time that the computed packet (*i.e.,* by abusing the notation $p_{n,i}\mathbf{x}$) is received by the collector from helper n. We assume that the time of transmitting the first packet to each helper, *i.e.,* $p_{n,1}, \forall n \in \mathcal{N}$, is zero; *i.e.,* $Tx_{n,1} = 0$, $\forall n \in \mathcal{N}$.

Let us first consider the ideal scenario, Fig. 2(a), where $TTI_{n,i}$ is equal to $\beta_{n,i}$ for all packets that are transmitted to helper n. Indeed, if $TTI_{n,i} > \beta_{n,i}$, Fig. 2(b), helper n stays idle, which reduces the efficient utilization of resources and increases the task completion delay. On the other hand, if $TTI_{n,i} < \beta_{n,i}$, Fig. 2(c), packets are queued at helper n. This congested (overloaded) scenario is not ideal, because the collector can receive enough number of packets before all queued packets in helpers are processed, which wastes resources.

Determining $TTI_{n,i}$ in practice. Now that we know that $TTI_{n,i} = \beta_{n,i}$ should be satisfied for the best system efficiency and smallest task completion delay, the collector can set $TTI_{n,i}$ to $\beta_{n,i}$. However, the collector does not know $\beta_{n,i}$ a priori as it is the computation runtime of packet $p_{n,i}$ at helper n. Thus, we should determine $TTI_{n,i}$ without explicit knowledge of $\beta_{n,i}$.

Our approach in C3P is to estimate $\beta_{n,i}$ as $E[\beta_{n,i}]$, where expectation is taken over packets. We will explain how to calculate $E[\beta_{n,i}]$ later in this section, but before that let us explain how to use estimated $E[\beta_{n,i}]$ for setting $TTI_{n,i}$. It is obvious that if the computed packet $p_{n,i}\mathbf{x}$ is received at the collector before packet $p_{n,i+1}$ is transmitted from the collector to helper n, the helper will be idle until it receives packet $p_{n,i+1}$. Therefore, to better utilize resources at helper n, the collector should offload a new packet before or immediately after receiving the computed value of the previous packet, *i.e.,* $TTI_{n,i} \leq Tr_{n,i} - Tx_{n,i}$ should be satisfied as in Fig. 2. Therefore, if the calculated $E[\beta_{n,i}]$ is larger than $Tr_{n,i} - Tx_{n,i}$, then we set $TTI_{n,i}$ as $Tr_{n,i} - Tx_{n,i}$ to satisfy this condition. In other words, $TTI_{n,i}$ is set to

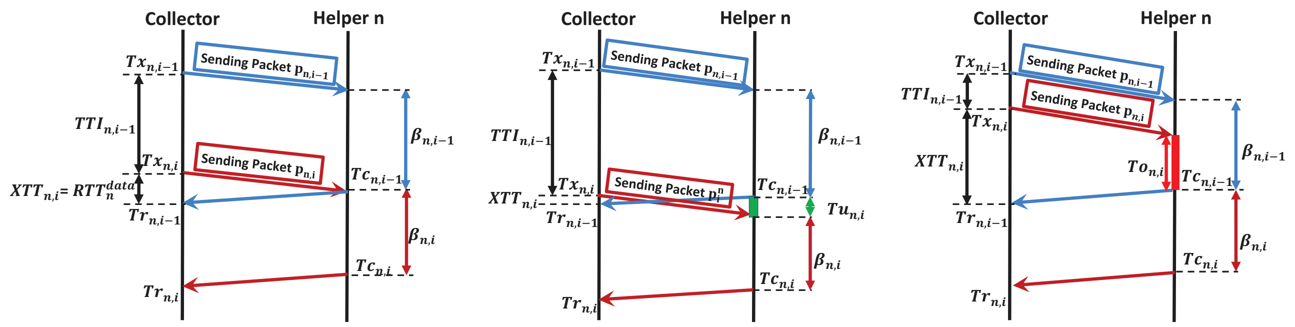
$$TTI_{n,i} = \min(Tr_{n,i} - Tx_{n,i}, E[\beta_{n,i}]). \tag{6}$$

Calculation of $E[\beta_{n,i}]$. In C3P, $E[\beta_{n,i}]$ is estimated using runtimes of previous packets:

$$E[\beta_{n,i}] \approx \frac{\sum_{j=1}^{m_n} \beta_{n,i}}{m_n}, \tag{7}$$

where m_n is the number of computed packets received at the collector from helper n before sending packet $p_{n,i+1}$. In order to calculate (7), the collector device should have $\beta_{n,i}$ values from the previous offloaded packets. A straightforward approach would be putting timestamps on sub-tasks to directly access the runtimes $\beta_{n,i}$ at the collector. However, this approach introduces overhead on sub-tasks. Thus, we also developed a mechanism, where the collector device infers $\beta_{n,i}$ by taking into account transmission and ACK times of sub-tasks. The details of this approach is provided in Appendix A of [13].

C3P in a nutshell. The main goal of C3P is to determine packet transmission intervals, $TTI_{n,i}$, according to (6), which is summarized in Algorithm 1. Note that Algorithm 1 has also a timeout value defined in line 7, which is needed for unresponsive helpers. If helper n is not responsive, $TTI_{n,i}$ is quickly increased as shown in line 6 so that fewer and fewer packets could be offloaded to that helper. In particular, C3P doubles $TTI_{n,i}$ when the timeout for receiving ACK occurs. This is inspired by additive increase multiplicative decrease



(a) Ideal case (b) Underutilized case (c) Congested case

Fig. 2. Different states of the system: (a) ideal case, (b) underutilized case, and (c) congested case.

Algorithm 1 C3P algorithm at the collector

```
1: Initialize: $TO_n = \infty$, $\forall n \in \mathcal{N}$.
2: while $R + K$ calculated packets have not been received do
3:     if Calculated packet $p_{n,i}\mathbf{x}$ is received before timeout expires then
4:         Calculate $TTI_{n,i}$ according to (7) and (6)
5:     else
6:         $TTI_{n,i} = 2 \times TTI_{n,i}$.
7:     Update timeout as $TO_n = 2TTI_{n,i}$
```

strategy of TCP, where the number of transmitted packets are halved to backoff quickly when the system is not responding.

After $TTI_{n,i}$ is updated when a transmitted packet is ACKed or timeout occurs, this interval is used to determine the transmission times of the next coded packets. In particular, coded packets are generated and transmitted one by one to all helpers with intervals $TTI_{n,i}$ until (i) $TTI_{n,i}$ is updated with a new ACK packet or when timeout occurs, or (ii) the collector collects $R + K$ computed packets. Next, we characterize the performance of C3P.

IV. Performance Analysis of C3P

A. Performance of C3P w.r.t. the Non-Ergodic Solution

In this section, we analyze the gap between C3P and the non-ergodic solution characterized in Section III-A. Let us first characterize the task completion delay of C3P as

$$T^{\text{C3P}} = \max_{n\in\mathcal{N}} \Big(RTT_n^{\text{data}} + \sum_{i=1}^{r_n^{\text{C3P}}} (\beta_{n,i} + Tu_{n,i})\Big), \tag{8}$$

where $r_n^{\text{C3P}} = \text{argmin}_{r_n} \max_{n\in\mathcal{N}} \Big(RTT_n^{\text{data}} + \sum_{i=1}^{r_n}(\beta_{n,i} + Tu_{n,i})\Big)$, and $Tu_{n,i}$ is per packet under-utilization time at helper n, which occurs as C3P does not have a priori knowledge of $\beta_{n,i}$, but it estimates $\beta_{n,i}$ and accordingly determines packet transmission times $TTI_{n,i}$ according to (6). The gap between T^{C3P} and T^{best} in (2) is upper bounded by:

$$\begin{aligned}
T^{\text{C3P}} - T^{\text{best}} &= \max_{n\in\mathcal{N}} \Big(RTT_n^{\text{data}} + \sum_{i=1}^{r_n^{\text{C3P}}} (\beta_{n,i} + Tu_{n,i})\Big) \\
&\quad - \max_{n\in\mathcal{N}} \Big(RTT_n^{\text{data}} + \sum_{i=1}^{r_n^{\text{best}}} \beta_{n,i}\Big) \\
&\le \max_{n\in\mathcal{N}} \Big(RTT_n^{\text{data}} + \sum_{i=1}^{r_n^{\text{best}}} (\beta_{n,i} + Tu_{n,i})\Big) \\
&\quad - \max_{n\in\mathcal{N}} \Big(RTT_n^{\text{data}} + \sum_{i=1}^{r_n^{\text{best}}} \beta_{n,i}\Big) \\
&\le \max_{n\in\mathcal{N}} \Big(RTT_n^{\text{data}} + \sum_{i=1}^{r_n^{\text{best}}} \beta_{n,i}\Big) + \max_{n\in\mathcal{N}} \sum_{i=1}^{r_n^{\text{best}}} Tu_{n,i} \\
&\quad - \max_{n\in\mathcal{N}} \Big(RTT_n^{\text{data}} + \sum_{i=1}^{r_n^{\text{best}}} \beta_{n,i}\Big) \\
&= \max_{n\in\mathcal{N}} \sum_{i=1}^{r_n^{\text{best}}} Tu_{n,i},
\end{aligned} \tag{9}$$

where the first inequality comes from $r_n^{\text{C3P}} = \text{argmin}_{r_n} \max_{n\in\mathcal{N}} \Big(RTT_n^{\text{data}} + \sum_{i=1}^{r_n}(\beta_{n,i} + Tu_{n,i})\Big)$ and the second inequality comes from the fact that $\max(f(x) + g(x)) \le (\max(f(x)) + \max(g(x)))$.[3] As seen, the gap between C3P and the non-ergodic solution is bounded with the sum of $Tu_{n,i}$. The next theorem characterizes $Tu_{n,i}$.

Theorem 1: $Tu_{n,i}$ is monotonically decreasing with increasing number of sub-tasks, and $\lim_{i\to\infty} Pr(Tu_{n,i} > 0) \to 0$.

Proof: Let us first consider the following that identifies the conditions for positive $Tu_{n,i+1}$.

Lemma 2: The necessary and sufficient conditions to satisfy $Tu_{n,i+1} > 0$ are

$$\sum_{j=i+1-k}^{i} \beta_{n,j} < kE[\beta_{n,i}], \forall k = 1, 2, \dots, i \tag{10}$$

Proof: The proof is provided in Appendix B of [13]. □

According to the conditions given in Lemma 2, the proba-

[3]Note that in (9), we assume that the runtime of packet i at helper n is the same both in the non-ergodic solution and C3P, which is necessary for fair comparison.

bility of $Tu_{n,i} > 0$ is calculated as:

$$Pr(Tu_{n,i} > 0) = \int_{0}^{E[\beta_{n,i}]} \int_{0}^{2E[\beta_{n,i}]-x_i} \cdots \int_{0}^{iE[\beta_{n,i}]-\sum_{j=2}^{i}\beta_{n,j}} f_{\beta_{n,1},\ldots,\beta_{n,i}}(x_1,\ldots,x_i)dx_1\ldots dx_i, \quad (11)$$

where $f_{\beta_{n,1},\ldots,\beta_{n,i}}(x_1,\ldots,x_i)$ is the joint probability density function of $(\beta_{n,1},\ldots,\beta_{n,i})$. With the assumption that $\beta_{n,j}, j = 1, 2, ..., i$ is from an i.i.d distribution, the joint probability distribution function of $\beta_{n,1},\ldots,\beta_{n,i}$ is the product of i probability distribution functions:

$$Pr(Tu_{n,i} > 0) = \int_{0}^{E[\beta_{n,i}]} \int_{0}^{2E[\beta_{n,i}]-x_i} \ldots \int_{0}^{iE[\beta_{n,i}]-\sum_{j=2}^{i}x_j} f_{\beta_{n,i}}(x_1)f_{\beta_{n,i}}(x_2)\ldots f_{\beta_{n,i}}(x_i)dx_1dx_2\ldots dx_i \quad (12)$$

$$= \int_{0}^{E[\beta_{n,i}]} f_{\beta_{n,i}}(x_i) \int_{0}^{2E[\beta_{n,i}]-x_i} f_{\beta_{n,i}}(x_{i-1}) \ldots \int_{0}^{(i-1)E[\beta_{n,i}]-\sum_{j=3}^{i}x_j} f_{\beta_{n,i}}(x_2) \int_{0}^{iE[\beta_{n,i}]-\sum_{j=2}^{i}x_j} f_{\beta_{n,i}}(x_1)dx_1dx_2\ldots dx_i \quad (13)$$

$$< \int_{0}^{E[\beta_{n,i}]} f_{\beta_{n,i}}(x_i) \int_{0}^{2E[\beta_{n,i}]-x_i} f_{\beta_{n,i}}(x_{i-1}) \ldots \int_{0}^{(i-1)E[\beta_{n,i}]-\sum_{j=3}^{i}x_j} f_{\beta_{n,i}}(x_2)dx_2\ldots dx_i \quad (14)$$

$$= \int_{0}^{E[\beta_{n,i}]} f_{\beta_{n,i}}(x_{i-1}) \int_{0}^{2E[\beta_{n,i}]-x_{i-1}} f_{\beta_{n,i}}(x_{i-2})\ldots \int_{0}^{(i-1)E[\beta_{n,i}]-\sum_{j=2}^{i-1}x_j} f_{\beta_{n,i}}(x_1)dx_1\ldots dx_{i-1}, \quad (15)$$

where the last inequality comes from the fact that $\int_{0}^{iE[\beta_{n,i}]-\sum_{j=2}^{i}x_j} f_{\beta_{n,i}}(x_1)dx_1$ is less than 1, because the probability density function is integrated over a finite range of variable x_1, and the last equality comes from a change of variables in the integrals. (15) is equal to $Pr(Tu_{n,i-1} > 0)$ and thus $Pr(Tu_{n,i} > 0) < Pr(Tu_{n,i-1} > 0)$. Similarly, we can show that:

$$Pr(Tu_{n,j} > 0) < Pr(Tu_{n,j-1} > 0), \forall j = 2, 3, \ldots, i \quad (16)$$

From the above equation, we can conclude that as i gets larger and larger, $Pr(Tu_{n,i} > 0)$ gets smaller, and $\lim_{i\to\infty} Pr(Tu_{n,i} > 0) \to 0$ is satisfied. This concludes the proof. □

We can conclude from Theorem 1 that the rate of the increase in the gap between C3P and the non-ergodic solution decreases with increasing the number of sub-tasks and eventually the rate becomes zero for $R \to \infty$. Therefore, the gap becomes finite even for $R \to \infty$.

B. *Performance of* C3P *w.r.t the Static Solution*

In this section, we analyze the performance of C3P as compared to the static solution characterized in Section III-B. The next theorem characterizes the task completion delay of C3P as well as the optimal task offloading policy.

Theorem 3: The task completion delay of C3P approaches to

$$T^{\texttt{C3P}} \approx \frac{R+K}{\sum_{n=1}^{N}\frac{1}{E[\beta_{n,i}]}}, \quad (17)$$

with increasing R and the number of offloaded tasks to helper n is approximated as

$$r_n^{\texttt{C3P}} \approx \frac{R+K}{E[\beta_{n,i}]\sum_{n=1}^{N}\frac{1}{E[\beta_{n,i}]}}. \quad (18)$$

Proof: Proof is provided in Appendix C of [13]. □

Theorem 3 shows that the task completion delay of C3P is getting close to the static solution T^{static} characterized in Section III-B with increasing R. The gap between T^{static} and $T^{\texttt{C3P}}$ is $\frac{K}{\sum_{n=1}^{N}\frac{1}{E[\beta_{n,i}]}}$ which is due to the coding overhead of Fountain codes, which becomes negligible for large R.

C. *Performance of* C3P *w.r.t. Repetition Codes*

In this section, we demonstrate the performance of C3P as compared to repetition coding with Round-robin (RR) scheduling through an illustrative example. Repetition codes with RR scheduling works as follows. Uncoded packets from the set $\Gamma = \{\rho_1, \rho_2, \ldots, \rho_R\}$ is offloaded to helpers one by one (in round robin manner) depending on their sequence in Γ. For example, ρ_1 is offloaded to helper 1, ρ_2 is offloaded to helper 2, and so on. When all the packets are offloaded from Γ, we start again from the first packet in the set (so it is a repetition coding). Note that whenever a packet is computed and a corresponding ACK is received, the packet is removed from Γ. Thus, this RR scheduling continues until Γ becomes an empty set. We use $TTI_{n,i}$ in (6) to determine the next scheduling time for helper n. The next example demonstrates the benefit of C3P as compared to this repetition coding mechanism with RR scheduling.

Example 2: We consider the same setup in Example 1. We assume that per-packet runtimes are $\beta_{1,1} = 1, \beta_{1,2} = 1, \beta_{1,3} = 0.5, \beta_{1,4} = 1, \beta_{1,5} = 1.5$, $\beta_{2,1} = 1.5, \beta_{2,2} = 3.5$, and $\beta_{3,1} = 3, \beta_{3,2} = 2.5$, and the transmission times of packets are negligible.

As seen in Fig. 3(a), RR scheduler sends ρ_1, ρ_2, and ρ_3 to helpers 1, 2, and 3, respectively at time $t = 0$. At time $t = 1$, the computed packet $\rho_1\mathbf{x}$ is received at the collector, and ρ_4, which is the next packet selected by RR scheduler, is transmitted to helper 1. Similarly, at time $t = 1.5$, $\rho_2\mathbf{x}$ is received at the collector, and ρ_5 is transmitted to helper 2. Similarly, the next packets are transmitted to helpers until the result of all packets are received at the collector, which is achieved at time $t = 5$. As seen, the resources of helper 1 is wasted while computing ρ_3, because those resources could have been used for computing a new packet. C3P addresses this problem thanks to employing Fountain codes.

In particular, at time $t = 0$, three Fountain coded packets of ν_1, ν_2, ν_3 are created and transmitted to the three helpers, *i.e.,* $p_{1,1} = \nu_1, p_{2,1} = \nu_2, p_{3,1} = \nu_3$. At time $t = 1$, a new coded packet of ν_4 is created and transmitted as a second packet to helper 1, *i.e.,* $p_{1,2} = \nu_4$. This continues until 6 computed coded packets (assuming that the overhead of Fountain codes, *i.e.,* K is zero) are received at the collector, which is achieved at time $t = 3.5$. $\square$

Example 2 shows that the task completion delay is reduced from 5 to 3.5 when we use Fountain codes, which is significant. Section V shows extensive simulation results to support this illustrative example.

D. *Efficiency of* C3P

In this section, we characterize the efficiency of C3P in the worst case scenario when per task runtimes follow the shifted exponential distribution. We call it the worst case efficiency, because we take into account per packet under-utilization $Tu_{n,i}$ in efficiency calculation, but the fact that $Tu_{n,i}$ is monotonically decreasing, which is stated in Theorem 1 is not used.

Theorem 4: Assume that the runtime of each packet, *i.e.,* $\beta_{n,i}$, is a random variable according to an i.i.d shifted exponential distribution of

$$F_{\beta_{n,i}}(t) = Pr(\beta_{n,i} < t) = 1 - e^{-\mu_n(t-a_n)}, \qquad (19)$$

with mean $a_n + 1/\mu_n$ and shifted value of a_n. The expected value of the duration that helper n is underutilized per packet is characterized as:

$$E[Tu_{n,i}] = \begin{cases} \frac{1}{(e\mu_n)}\left(1 - e^{(\mu_n RTT_n^{data})}\right) + RTT_n^{data}, \\ \qquad \text{if } RTT_n^{data} < \frac{1}{\mu_n} \\ \frac{1}{(e\mu_n)}, \quad \text{otherwise.} \end{cases} \qquad (20)$$

Proof: The proof is provided in Appendix D of [13]. $\square$

We define the efficiency of helper n in the worst case as $\gamma_n = 1 - E[Tu_{n,i}]/E[\beta_{n,i}]$. Note that $E[Tu_{n,i}]$ is the expected time that helper n is underutilized per packet in the worst case, while $E[\beta_{n,i}]$ is the expected runtime duration, *i.e.,* the expected time that helper n works per packet. Thus, $E[Tu_{n,i}]/E[\beta_{n,i}]$ becomes the under-utilization ratio of helper n in the worst case, so $\gamma_n = 1 - E[Tu_{n,i}] \;/\; E[\beta_{n,i}]$ becomes the worst case efficiency. From (20) and replacing $E[\beta_{n,i}]$ with $a_n + 1/\mu_n$, γ_n is expressed as the following:

$$\gamma_n = \begin{cases} \frac{1+a_n\mu_n-\mu_n RTT_n^{\text{data}}-1/e+\exp(\mu_n RTT_n^{\text{data}}-1)}{1+a_n\mu_n}, \\ \qquad \text{if } RTT_n^{\text{data}} < 1/\mu_n \\ \frac{e(1+a_n\mu_n)-1}{e(1+a_n\mu_n)}, \qquad \text{otherwise.} \end{cases} \qquad (21)$$

We show through simulations (in Section V) that, (i) γ_n in (21) is larger than 99%, which is significant as (21) is the worst case efficiency, and (ii) C3P's efficiency is even larger than γ_n as γ_n in (21) is the efficiency in the worst case, where the under-utilization time period has the maximum value.

V. Performance Evaluation of C3P

In this section, we evaluate the performance of our algorithm; Coded Cooperative Computation Protocol (C3P) via simulations and using real Android-based smartphones.

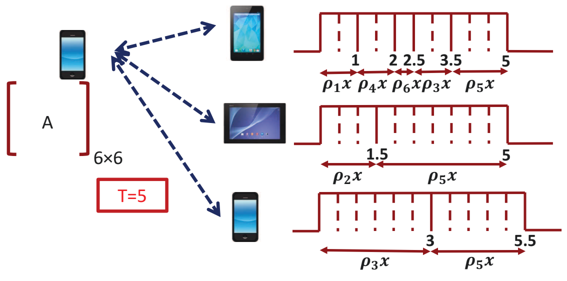
A. *Simulation Results*

We consider two scenarios: (i) Scenario 1, where the system resources for each helper vary over time. In this scenario, the runtime for computing each packet $p_{n,i}, \forall i$ at each helper n is an i.i.d. shifted exponential random variable with shifted value a_n and mean $a_n + 1/\mu_n$, and (ii) Scenario 2, where the runtime for computing packets in helper n does not change over time, *i.e.,* $\beta_{n,i} = \beta_n, \forall i$, and $\beta_n, \forall n \in \mathcal{N}$ is a shifted exponential random variable with shifted value a_n and mean $a_n + 1/\mu_n$.

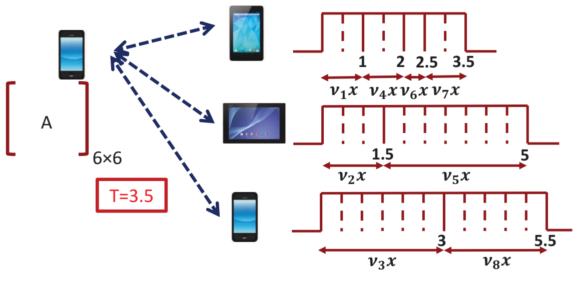
In our simulations, each simulated point is obtained by averaging over 200 iterations for $N = 100$ helpers. The transmission rate for sending each packet from the collector to each helper n and from helper n to the collector is a Poisson random variable with the average selected uniformly between 10 Mbps and 20 Mbps for each helper n. The size of a transmitted packet $p_{n,i}$ is set to $B_x = 8R$ bits, where R is the number of rows of matrix A, and it varies from 500 to $20,000$ in our simulations. The sizes of a computed packet $p_{n,i}\mathbf{x}$ and an acknowledgement packet are set to $B_r = 8$ bits and $B_{\text{ack}} = 1$ bit, respectively. These are the parameters that are used for creating all plots unless otherwise is stated.

Task Completion Delay vs. Number of Rows: We evaluate C3P for Scenarios 1 and 2 and compare its task completion delay with: (i) Static solution, which is the task completion delay characterized in Section III-B for both Scenarios 1 and 2. (ii) Non-ergodic solution, which is a realization of the non-ergodic problem characterized in Section III-A by knowing $\beta_{n,i}$ a priori at the collector and setting $TTI_{n,i}$ as $\beta_{n,i}$. (iii) Uncoded: r_n packets without coding are assigned to each helper n, and the collector waits to receive computed values from all helpers. The number of assigned packets to each helper n is inversely proportional to the mean of $\beta_{n,i}$, *i.e.,* $r_n \propto \frac{1}{a_n+1/\mu_n}$. (iv) HCMM: Coded cooperative framework developed in [7] using block codes. We introduce 5% coding overhead for C3P, static, and non-ergodic solutions.

Fig. 4(a) shows completion delay versus number of rows for Scenario 1, where the runtime for computing each packet by helper n, $\beta_{n,i}, \forall i$, is a shifted exponential random variable with shifted value of $a_n = 0.5$ and mean of $a_n + 1/\mu_n$, where μ_n is selected uniformly from $\{1, 2, 4\}$. As seen, C3P performs close to the static and non-ergodic solutions. This shows the effectiveness of our proposed algorithm. In addition, C3P performs better than the baselines. In particular, in average, 30% and 24% improvement is obtained by C3P over HCMM and no coding, respectively. Fig. 4(b) considers the same setup but for Scenario 2, where the runtime for computing r_n packets by helper n is $r_n\beta_n$, where β_n is selected from a shifted exponential distribution with $a_n = 0.5, \forall n \in \mathcal{N}$ and μ_n, which is selected uniformly from $\{1, 2, 4\}$. As seen, for this scenario, C3P performs close to the static and non-ergodic

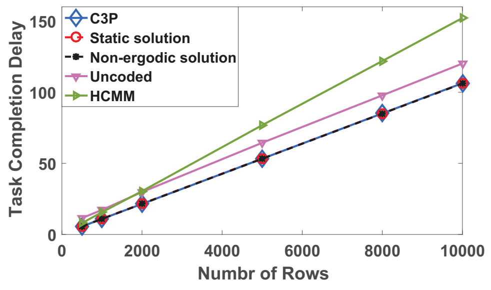


(a) Repetition codes with RR scheduling

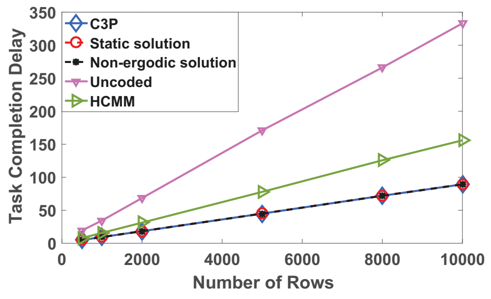


(b) C3P

Fig. 3. Performance of C3P with respective to repetition codes with RR scheduling.

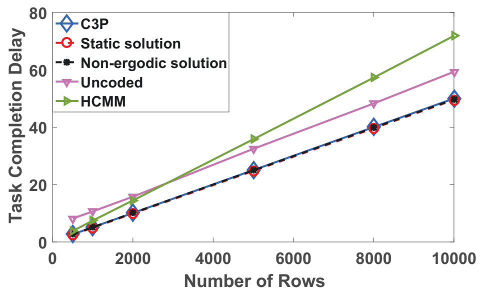


(a) Scenario 1

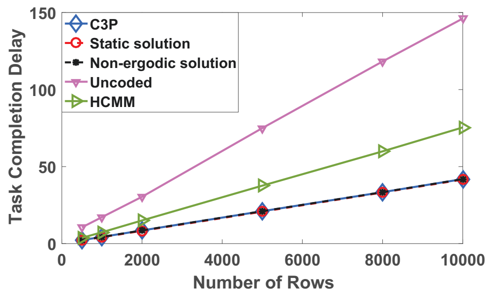


(b) Scenario 2

Fig. 4. Task completion delay vs. number of rows/packets for (i) Scenario 1, and (ii) Scenario 2, where the runtime for computing one row by helper n is selected from a shifted exponential distribution with $a_n = 0.5, \forall n \in \mathcal{N}$ and μ_n, which is selected uniformly from $\{1, 2, 4\}$.



(a) Scenario 1



(b) Scenario 2

Fig. 5. Task completion delay vs. number of rows/packets for (i) Scenario 1, and (ii) Scenario 2, where the runtime for computing one row by each helper n is selected from a shifted exponential distribution with μ_n, which is selected uniformly from $\{1, 3, 9\}$ for different helpers and $a_n = 1/\mu_n, \forall n \in \mathcal{N}$

solutions. C3P performs better than HCMM, and HCMM performs better than no coding. In particular, in average, 40% and 69% improvement is obtained by C3P over HCMM and no coding, respectively. Note that uncoded performs better than HCMM for Scenario 1, as HCMM is designed for Scenario 2, so it does not work well in Scenario 1. C3P performs well in both scenarios.

Fig. 5 shows completion delay versus number of rows for both Scenarios 1 and 2, where the runtime for computing the rows by each helper n, is from a shifted exponential distribution with $\mu_n, n \in \mathcal{N}$ selected uniformly from $\{1, 3, 9\}$ and $a_n = 1/\mu_n$ (different shifted values for different helpers). As seen, C3P performs close to static and non-ergodic solutions and much better than the baselines. In particular, for Scenario 1, more than 30% and 15% improvement is obtained by C3P over HCMM and no coding, respectively. Also, for Scenario 2, in average, 42% and 73% improvement is obtained by C3P over HCMM and no coding, respectively.

Efficiency: We calculated the efficiency of helpers for different simulation setups and compared it with the theoretical efficiency obtained in (21) for Scenario 1. For all simulation setups, the average efficiency over all helpers was around 99% and the theoretical efficiency was a little lower than the simulated efficiency. *E.g.,* for $R = 8000$ rows, where $\mu_n, n \in \mathcal{N}$ is selected uniformly from $\{1, 3, 9\}$ and $a_n = 1/\mu_n$, the average of efficiency over all helpers is 99.7072% and the average of theoretical efficiency is 99.4115%. This is expected as the theoretical efficiency is calculated for the worst case scenario.

We also calculate the efficiency of helpers for Scenario 2. For all simulation setups, the average efficiency over all helpers was around 99%, *e.g.,* for $R = 8000$ rows, where $\mu_n, n \in \mathcal{N}$ is selected uniformly from $\{1, 3, 9\}$ and $a_n = 1/\mu_n$, the average of efficiency over all helpers was 99.9267%. Note that the theoretical efficiency for Scenario 1 is 100%. The simulated efficiency is lower than the theoretical one, because the simulation underutilizes the helpers when transmitting

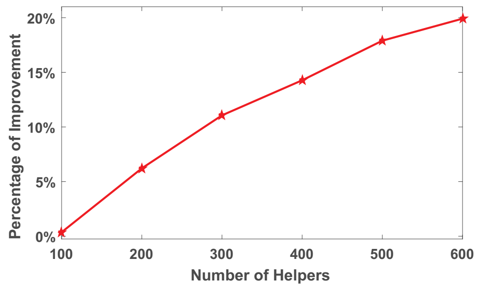


Fig. 6. Percentage of improvement of `C3P` over repetition codes with RR scheduling in terms of the task completion delay.

the very first packet to each helper, *i.e.,* before the collector estimates the resources of helpers.

`C3P` *as Compared to Repetition Coding and Round Robin Scheduling:* Fig. 6 shows the percentage of improvement of `C3P` over repetition coding with RR scheduling in terms of task completion delay. The number of rows is selected as $R = 2000$ with 5% overhead for `C3P` and the number of helpers varies from $N = 100$ to $N = 600$. The transmission rate for sending each packet from the collector to each helper n and from helper n to the collector is a Poisson random variable with the average selected uniformly between 0.1 Mbps and 0.2 Mbps for each helper n. The other parameters are the same as the parameters used in Fig. 4(a). As seen, by increasing the number of helpers, more improvement is gained by `C3P` compared to the repetition coding with RR scheduling.

B. Evaluation in a Testbed

We implemented a testbed of a collector and multiple helpers using real mobile devices, specifically Android 6.0.1 based Nexus 6P and Nexus 5 smartphones. All the helpers are connected to the collector device using Wi-Fi Direct connections. We conducted our experiments using our testbed in a lab environment where several other Wi-Fi networks were operating in the background. We located all the devices in close proximity of each other (within a few meters distance).

We implemented both `C3P` and repetition coding with RR scheduling in our testbed. The collector device would like to calculate matrix multiplication $\mathbf{y} = A\mathbf{x}$, where A is a $1\text{K} \times 10\text{K}$ matrix and $\mathbf{x}$ is a $10\text{K} \times 1$ vector. Matrix A is divided into 20 sub-matrices, each of which is a $50 \times 10\text{K}$ matrix. A sub-task to be processed by a helper is the multiplication of a sub-matrix with vector $\mathbf{x}$. There is collector device (Nexus 5) and varying number of helpers (Nexus 6P).

Fig. 7 shows task completion delay versus number of helpers for both `C3P` and repetition codes with RR scheduling. In this setup, each helper receives a sub-task, processes it, and waits for a random amount of time (exponential random variable with mean 10 seconds), which may arise due to other applications running at smartphones, and then sends the result back to the collector. As can be seen, the task completion delay reduces with increasing number of helpers in both algorithms. When there is one helper `C3P` performs worse, which is expected. In particular, `C3P` introduces coding overhead, and the number of helpers is very small to see the benefit of coding. On the other hand, when the number of helpers increases, we start seeing the benefit of coding. For example, when the number of helpers is 5, `C3P` improves 14% over repetition codes with RR scheduling. This result confirms our simulation results in Fig. 6 in a testbed with real Android-based smartphones.

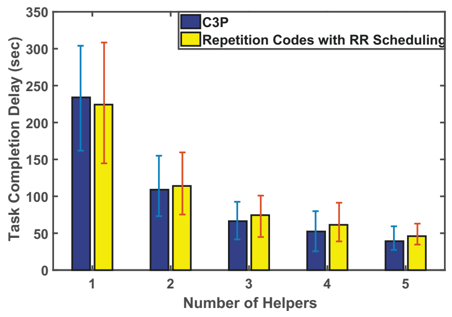


Fig. 7. Task completion delay versus number of helpers.

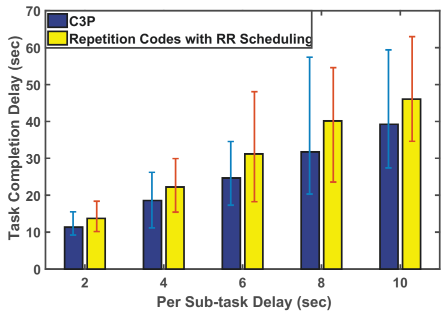


Fig. 8. Task completion delay versus per sub-task delay.

Fig. 8 shows the task completion delay versus per sub-task random delays at helpers. There are 5 helpers in this scenario. As can be seen, `C3P` improves more over repetition codes with RR scheduling when delay increases as it increases heterogeneity, and `C3P` designed to take into account heterogeneity.

VI. Related Work

Mobile cloud computing is a rapidly growing field with the goal of providing extensive computational resources to mobile devices as well as higher quality of experience [14], [15], [16]. The initial approach to mobile cloud computing has been to offload resource intensive tasks to remote clouds by exploiting Internet connectivity of mobile devices. This approach has received a lot of attention which led to extensive literature in the area [17], [18], [19], [20], [21]. The feasibility of computation offloading to remote cloud by mobile devices [22] as well as energy efficient computation offloading [23], [24] has been considered in the previous work. As compared to this line of work, our focus is on edge computing rather than remote clouds.

There is an increasing interest in edge computing by exploiting connectivity among mobile devices [25]. This approach suggests that if devices in close proximity are capable of processing tasks cooperatively, then local area computation groups could be formed and exploited for computation. Indeed, cooperative computation mechanisms by exploiting device-to-device connections of mobile devices in close proximity are developed in [25] and [26]. A similar approach is considered in [27] with particular focus on load balancing across workers. As compared to this line of work, we consider coded cooperative computation.

Coded cooperative computation is shown to provide higher reliability, smaller delay, and reduced communication cost in MapReduce framework [28], where computationally intensive tasks are offloaded to distributed server clusters [29]. In [3] and [30], coded computation for matrix multiplication is considered, where matrix A is divided into sub-matrices and each sub-matrix is sent from the master node (called collector in our work) to one of the worker nodes (called helpers in our work) for matrix multiplication with the assumption that the helpers are homogeneous. In [3], workload of the worker nodes is optimized such that the overall runtime is minimized. Fountain codes are employed in [31] for coded computation, but for homogeneous resources. In [7], the same problem is considered, but with the assumption that workers are heterogeneous in terms of their resources. Compared to this line of work, we develop `C3P`, a practical algorithm that is (i) adaptive to the time-varying resources of helpers, and (ii) does not require any prior information about the computation capabilities of the helpers. As shown, our proposed method reduces the task completion delay significantly as compared to prior work.

VII. Conclusion

In this paper, we designed a Computation Control Protocol (`C3P`), where heterogeneous edge devices with computation capabilities and energy resources are connected to each other. In `C3P`, a collector device divides tasks into sub-tasks, offloads them to helpers by taking into account heterogeneous resources. `C3P` is (i) a dynamic algorithm that efficiently utilizes the potential of each helper, (ii) adaptive to the time-varying resources at helpers. We analyzed the performance of `C3P` in terms of task completion delay and efficiency. Simulation and experiment results in an Android testbed confirm that `C3P` reduces the completion delay significantly as compared to baselines and is efficient.

References

[1] Y. Li and W. Wang, "Can mobile cloudlets support mobile applications?," *in Proc. of IEEE INFOCOM*, Toronto, ON, 2014.

[2] M. Chen, Y. Hao, Y. Li, C. F. Lai and D. Wu, "On the computation offloading at ad hoc cloudlet: architecture and service modes," *in IEEE Communications Magazine*, vol. 53, no. 6, pp. 18-24, June 2015.

[3] K. Lee, M. Lam, R. Pedarsani, D. Papailiopoulos and K. Ramchandran, "Speeding up distributed machine learning using codes," *in Proc. of IEEE International Symposium on Information Theory (ISIT)*, Barcelona, Spain, July 2016.

[4] C. Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton, and G. Hullender, "Learning to rank using gradient descent," *in Proc. of the ACM 22nd International Conference on Machine Learning (ICML)*, Bonn, Germany, Aug. 2005.

[5] T. Zhang, "Solving large scale linear prediction problems using stochastic gradient descent algorithms," *in Proc. of the ACM 21st International Conference on Machine Learning (ICML)*, Banff, Canada, July 2004.

[6] L. Bottou, "Large-scale machine learning with stochastic gradient descent," *in Proc. of the International Conference on Computational Statistics (COMPSTAT)*, Paris, France, Aug. 2010.

[7] A. Reisizadeh, S. Prakash, R. Pedarsani and A. S. Avestimehr, "Coded Computation over Heterogeneous Clusters," *in Proc. of IEEE International Symposium on Information Theory (ISIT)*, Aachen, Germany, June 2017.

[8] M. Luby, "LT codes," *in Proc. of The 43rd Annual IEEE Symposium on Foundations of Computer Science*, 2002.

[9] A. Shokrollahi, "Raptor codes," *in IEEE Transactions on Information Theory*, vol. 52, no. 6, pp. 2551-2567, June 2006.

[10] D. J. C. MacKay, "Fountain codes," *in IEE Proceedings - Communications*, vol. 152, no. 6, pp. 1062-1068, Dec. 2005.

[11] T. Ho, R. Koetter, M. Medard, D. R. Karger and M. Effros, "The benefits of coding over routing in a randomized setting," *in Proc. of IEEE International Symposium on Information Theory (ISIT)*, Yokohama, Japan, 2003.

[12] S. Lin, D. J. Costello, and M. J. Miller. "Automatic-repeat-request error-control schemes," *in IEEE Communications Magazine*, vol. 22(12), pp. 5-17, 1984.

[13] Y. Keshtkarjahromi, Y. Xing, H. Seferoglu, "Dynamic Heterogeneity-Aware Coded Cooperative Computation at the Edge," available in ArXiv, arXiv:1801.04357v2.

[14] H. T. Dinh, C. Lee, D. Niyato, and P. Wang, "A survey of mobile cloud computing: architecture, applications, and approaches," *Wireless Communications and Mobile Computing*, vol. 13, no. 8, October 2011.

[15] N. Fernando, S. W. Loke, and W. Rahayu, "Mobile cloud computing: A survey," *Future Generation Computer Systems*, vol. 29, no. 1, pp. 84 – 106, 2013.

[16] Z. Sanaei, S. Abolfazli, A. Gani, and R.Buyya, "Heterogeneity in mobile cloud computing: Taxonomy and open challenges," *Communications Surveys Tutorials, IEEE*, vol. 16, no. 1, pp. 369–392, First 2014.

[17] M. Gordon, D. Jamshidi, S. Mahlke, Z. Mao, and X. Chen, "Comet: Code offload by migrating execution transparently," *in Proc. OSDI*, Hollywood, CA, October 2012.

[18] E. Cuervo, A. Balasubramanian, D. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, "Maui: Making smartphones last longer with code offload," *in Proc. ACM MobiSys*, San Francisco, CA, June 2010.

[19] Y. Zhang, G. Huang, X. Liu, W. Zhang, H. Mei, and S. Yang, "Refactoring android java code for on-demand computation offloading," *in OOPSLA*, Tuscon, AZ, October 2012.

[20] R. Kemp, N. Palmer, T. Kielmann, and H. Bal, "Cuckoo: A computation offloading framework for smartphones," *Mobile Computing, Applications, and Services*, vol. 76, pp. 59–79, 2012.

[21] D. T. Hoang, D. Niyato, and P. Wang, "Optimal admission control policy for mobile cloud computing hotspot with cloudlet," *in Wireless Communications and Networking Conference (WCNC)*, 2012 IEEE, April 2012, pp. 3145–3149.

[22] S. Kosta, A. Aucinas, P. Hui, R. Mortier, and X. Zhang, "Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading," *in INFOCOM*, 2012 Proceedings IEEE, March 2012, pp. 945–953.

[23] Y. Geng, W. Hu, Y. Yang, W. Gao, and G. Cao, "Energy-efficient computation offloading in cellular networks," *in 2015 IEEE 23rd International Conference on Network Protocols (ICNP)*, Nov 2015, pp. 145–155.

[24] W. Zhang, Y. Wen, and D. O. Wu, "Energy-efficient scheduling policy for collaborative execution in mobile cloud computing," *in INFOCOM, 2013 Proceedings IEEE*, April 2013, pp. 190–194.

[25] R. K. Lomotey and R. Deters, "Architectural designs from mobile cloud computing to ubiquitous cloud computing - survey," *in Proc. IEEE Services*, Anchorage, Alaska, June 2014.

[26] E. Miluzzo, R. Caceres and Y. Chen, "Vision: mclouds - computing on clouds of mobile devices," *in ACM workshop on Mobile cloud computing and services*, Low Wodd Bay, Lake District, UK, June 2012.

[27] T. Penner, A. Johnson, B. V. Slyke, M. Guirguis and Q. Gu, "Transient clouds: Assignment and collaborative execution of tasks on mobile devices," *in Proc. IEEE GLOBECOM*, Austin, TX, Dec. 2014.

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

[29] S. Li, M. A. Maddah-Ali and A. S. Avestimehr, "Coded MapReduce," *in 53rd Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, Monticello, IL, 2015.

[30] S. Dutta, V. Cadambe and P. Grover, "Short-dot: Computing large linear transforms distributedly using coded short dot products," *in Advances In Neural Information Processing Systems (NIPS)*, Barcelona, Spain, Dec. 2016.

[31] A. Mallick, M. Chaudhari and G. Joshi, "Rateless Codes for Near-Perfect Load Balancing in Distributed Matrix-Vector Multiplication," available in ArXiv, arXiv:1804.10331v2.