This article has been accepted for publication in IEEE Transactions on Network Science and Engineering. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TNSE.2021.3095040

On Batch-Processing Based Coded Computing for Heterogeneous Distributed Computing Systems

Baoqian Wang, Junfei Xie, *Member, IEEE*, Kejie Lu, *Senior Member, IEEE*, Yan Wan, *Senior Member, IEEE*, and Shengli Fu, *Senior Member, IEEE*

Abstract—In recent years, *coded distributed computing* (CDC) has attracted significant attention, because it can efficiently facilitate many delay-sensitive computation tasks against unexpected latencies in distributed computing systems. Despite such a salient feature, many design challenges and opportunities remain. In this paper, we focus on practical computing systems with heterogeneous computing resources, and design a novel CDC approach, called *batch-processing based coded computing* (BPCC), which exploits the fact that every computing node can obtain some coded results before it completes the whole task. To this end, we first describe the main idea of the BPCC framework, and then formulate an optimization problem for BPCC to minimize the task completion time by configuring the computation load. Through formal theoretical analyses, extensive simulation studies, and comprehensive real experiments on the Amazon EC2 computing clusters, we demonstrate promising performance of the proposed BPCC scheme, in terms of high computational efficiency and robustness to uncertain disturbances.

Index Terms—Coded distributed computing, heterogeneous computing cluster, batch-processing, asymptotic optimality, latency.

1 Introduction

In recent years, distributed computing has been widely adopted to perform various computation tasks in different computing systems [1]–[3]. For instance, to perform big data analytics in cloud computing systems, MapReduce [4] and Apache Spark [5] are the two prevalent modern distributed computing frameworks that process data in the order of petabytes.

Despite the importance of distributed computing, many design challenges remain. One major challenge is that many computing frameworks are vulnerable to uncertain disturbances, such as node/link failures, communication congestion, and slow-downs [6]. Such disturbances, which can be modeled as stragglers that are slow or even fail in returning results, have been observed in many large-scale computing systems such as cloud computing [7], mobile edge computing [8], and fog computing [9].

Baoqian Wang is with the Department of Electrical and Computer Engineering, San Diego State University, and University of California, San Diego, San Diego, CA, 92182 (e-mail: bwang4848@sdsu.edu*).*
Junfei Xie is with the Department of Electrical and Computer Engineering, San Diego State University, San Diego, CA, 92182 (e-mail: jxie4@sdsu.edu*). Corresponding author.*
*Kejie Lu is with the Department of Computer Science and Engineering, University of Puerto Rico at Mayagüez, Mayagüez, Puerto Rico, 00681, e-mail: (*kejie.lu@upr.edu*).*
Yan Wan is with the Department of Electrical Engineering, University of Texas at Arlington, Arlington, Texas, 76019 (e-mail: yan.wan@uta.edu*).*
Shengli Fu is with the Department of Electrical Engineering, University of North Texas, Denton, Texas, 76201 (e-mail: Shengli.Fu@unt.edu*).*
Manuscript received October 1, 2020.

A variety of solutions have been developed in the literature to address stragglers. For example, the authors of [10] proposed to identify and blacklist nodes that are in bad health and to run tasks only on well-performed nodes. However, empirical studies show that stragglers can occur in non-blacklisted nodes as well [11], [12]. As another type of solution, delayed computation tasks can be re-executed in a speculative manner [4], [10], [13], [14]. Nevertheless, such speculative execution techniques have to wait to collect the performance statistics of the tasks before generating speculative copies and thus have limitations in dealing with small jobs [12]. To avoid waiting and predicting stragglers, the authors of [12], [15] suggested to execute multiple clones of each task and use results generated by the fastest clones. Although their results show the promising performance of this approach in reducing the average completion time of small jobs, the extra resources required for launching clones can be considerably large, because multiple clones are executed for each task.

Instead of directly replicating the whole task, the *coding* techniques can be adopted to introduce arbitrary redundancy into the computation in a systematic way. However, until a few years ago, the coding techniques have been mostly known for their capability in improving the resilience of communication, storage and cache systems to uncertain disturbances [16]–[18]. Lee *et al.* [19], [20] presented the first *coded distributed computing* (CDC) scheme to speed up matrix multiplication and data shuffling. Since then, CDC has attracted significant attention in

This article has been accepted for publication in IEEE Transactions on Network Science and Engineering. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TNSE.2021.3095040

the distributed computing community. Although a variety of CDC schemes have been developed to solve different computation problems, most of these schemes assume homogeneous computing nodes, which is not a common case in realistic scenarios. Moreover, they require each worker node to first complete the computation task and then send back the whole result to the master node, which introduces significant delays [19]–[22].

In this paper, we focus on a classical CDC task: matrix-vector multiplication and propose a novel coding scheme, called batch-processing based coded computing (BPCC), to speed up the computational efficiency of general distributed computing systems with heterogeneous computing nodes and improve their robustness to uncertain disturbances. Unlike most existing CDC schemes, our BPCC allows each node to return partial computing results to the master node in batches before the whole computation task is completed. Therefore, BPCC achieves lower latency. Also worthy of note is that the partial results can be used to generate approximated solutions, e.g., by applying the singular value decomposition (SVD) approach in [23], which is very useful for applications that require timely but unnecessarily optimized decisions such as emergency response. To the best of our knowledge, such a BPCC framework has not been fully investigated in the literature.

This paper extends our earlier work presented in [24], and further makes the following new contributions.

1) *An optimal load allocation strategy.* For systems with heterogeneous computing nodes, equally distributing the computation load may lead to bad performance. To optimize the computational efficiency, we formulate an optimization problem for general BPCC with the assumption that the processing time of each computing node follows a shifted exponential distribution. To solve the optimization problem, we formulate alternative optimization problems, based on which we design an optimal load allocation scheme that assigns proper amount of load to each node to achieve the minimal expected task completion time.
2) *Comprehensive theoretical analyses.* We conduct formal theoretical analyses to prove the asymptotic optimality of BPCC and the impact of its important parameter. We also prove that it outperforms a state-of-the-art CDC scheme for heterogeneous systems, called Heterogeneous Coded Matrix Multiplication (HCMM) [21], [22].
3) *Extensive simulation and real experimental studies.* To further demonstrate the performance of BPCC, we compare it with three benchmark schemes, including the Uniform Uncoded, Load-Balanced Uncoded, and HCMM. The simulation results show the impact of BPCC parameters including number of batches and number of worker nodes. Specifically, the efficiency of BPCC improves with the increase of the number of batches and the solution of BPCC is optimal when the number of worker nodes approaches infinity. A sensitivity study shows the performance of BPCC when parameters in the computing model take erroneous values. Moreover, the simulation results also demonstrate that BPCC can improve computing performance by reducing the latency up to 73%, 56%, and 34% over the aforementioned three benchmark schemes, respectively. In the real experiments, we test all distributed computing schemes in the Amazon EC2 computing clusters. In particular, we deploy a heterogeneous computing cluster that consists of different machine instances in Amazon EC2. The results show that our BPCC scheme is more efficient and robust to uncertain disturbances than the benchmark schemes.

The rest of this paper is organized as follows. Section 2 presents the related work. In Section 3, we introduce the system model and the BPCC framework, and then formulate an optimization problem for BPCC. To solve the optimization problem, in Section 4, we provide a two-step alternative formulation for which we design the BPCC scheme and conduct solid theoretical analysis to prove its optimality and understand the impact of its parameter. We then present extensive simulation and experimental results in Section 5 and Section 6, respectively, before concluding the paper in Section 7. For better readability, we move the proofs of all lemmas, theorems and corollaries to the Appendix.

2 Related Work

Following the seminal work in [17], [19], [20], many different computation problems have been explored using codes, such as the gradients [25], large matrix-matrix multiplication [26], linear inverse problems [27], and non-linear operations [28]. Other relevant coded computation solutions include the "Short-Dot" coding scheme [29] that offers computation speed-up by introducing additional sparsity to the coded matrices and the unified coded framework [30], [31] that achieves the trade-off between communication load and computation latency.

While most CDC schemes consider homogeneous computing nodes, there have been a few recent studies that investigated CDC over heterogeneous computing clusters. In particular, Kim *et al.* [32], [33] considered the matrix-vector multiplication problem and presented an optimal load allocation method that achieves a lower bound of the expected latency. Reisizadeh *et al.* [21] introduced a different approach, namely Heterogeneous Coded Matrix Multiplication (HCMM), that can maximize the expected computing results aggregated at the master node. In [21], [22], the authors proved that the HCMM is asymptotically optimal under the assumption that the processing time of each computing node follows a shifted exponential or Weibull distribution. Also of interest, Keshtkarjahromi *et al.* [34] considered the scenario when computing nodes have time-varying computing powers, and introduced

a coded cooperative computation protocol that allocates tasks in a dynamic and adaptive manner. Narra *et al.* [35] also developed an adaptive load allocation scheme and utilized a LSTM-based model to predict the computation capability of the worker nodes.

To reduce the output delay, there have been some attempts to enable early return of partial results [23], [36], [37]. In particular, an anytime coding technique was introduced in [23], which adopts the SVD to allow early output of approximated result. Also of interest is the study presented in [36], which introduced a hierarchical approach to address the limitations of above coding techniques in terms of wastefully ignoring the work completed by slow worker nodes. In particular, to better utilize the work completed by each worker node, it partitions the total computation at each worker node into layers of sub-computations, with each layer encoding part of the job. It then processes each layer sequentially. The final result can be obtained after the master node recovers all layers. The simulation results demonstrate the effectiveness of this approach in reducing the computation latency. However, as the worker nodes have to process the layers in the same order, the results obtained by slow worker nodes for layers that have already been recovered are useless. Furthermore, this approach, as well as aforementioned approaches, assumes homogeneous computing nodes. Another relevant study is presented in [37], which introduced a rateless fountain coding scheme that can utilize partial results returned by worker nodes.

3 System Models

In this section, we first introduce the computing system for distributed matrix-vector multiplication. We then illustrate three computing schemes, including the proposed batch processing-based coded computing (BPCC). Finally, we formulate an optimization problem for BPCC.

3.1 Computing System

We consider a distributed computing system that consists of one master node and N ($N \in \mathbb{Z}^+$) computing nodes, a.k.a., worker nodes. Using this system, we investigate how to quickly solve a matrix-vector multiplication problem, which is one of the most basic building blocks of many computation tasks. Specifically, we consider a matrix-vector multiplication problem $\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}$, where $\boldsymbol{y} \in \mathbb{R}^r$ is the output vector to be calculated, $\boldsymbol{x} \in \mathbb{R}^m$ is the input vector to be distributed from a master node to multiple workers, and $\boldsymbol{A} \in \mathbb{R}^{r\times m}$ is an $r \times m$ dimensional matrix pre-stored in the system. Both r and m can be very large, which implies that calculating $\boldsymbol{A}\boldsymbol{x}$ at a single computing node is not feasible. Finally, we define $[n] = \{1, 2, \ldots, n\}$, where n is an arbitrary positive integer, i.e., $n \in \mathbb{Z}^+$.

3.2 Computing Schemes

3.2.1 Uncoded Distributed Computing

To solve the above problem, a traditional distributed computing scheme divides matrix $\boldsymbol{A}$ into a set of sub-matrices $\boldsymbol{A}_1, \boldsymbol{A}_2, \cdots, \boldsymbol{A}_N$, and pre-stores each sub-matrix $\boldsymbol{A}_i \in \mathbb{R}^{\ell_i \times m}$ in computing node i, where $\forall i \in [N], \ell_i \in \mathbb{Z}^+$ and $\sum_{i=1}^{N} \ell_i = r$. Upon receiving the input vector $\boldsymbol{x}$, the master node sends vector $\boldsymbol{x}$ to all worker nodes. Each worker node i then computes $\boldsymbol{y}_i = \boldsymbol{A}_i\boldsymbol{x}$ and returns the result to the master node. After all results are received, the master node aggregates the results and outputs $\boldsymbol{y} = [\boldsymbol{y}_1^T, \boldsymbol{y}_2^T, \cdots, \boldsymbol{y}_N^T]^T$, where ^{T} stands for transpose.

Due to the existence of uncertain system disturbances, the uncoded computing scheme may defer or even fail the computation, because the delay or loss of any $\boldsymbol{y}_i$, $i \in [N]$, will affect the calculation of the final result $\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}$. To address this issue, more computing nodes can be used to perform distributed computing. For instance, the master node can have two or more computing nodes to compute $\boldsymbol{y}_i$. This approach, however, is not efficient because the cost can be unnecessarily large.

3.2.2 Coded Distributed Computing (CDC)

In recent years, a more efficient computing paradigm, CDC, has been introduced to tackle the issue of uncertain disturbances. There are many CDC schemes in the literature, and we consider a generic CDC scheme as follows.

In this CDC scheme, $\boldsymbol{A}$ will first be used to calculate a larger matrix $\hat{\boldsymbol{A}} \in \mathbb{R}^{q\times m}$ with more rows, i.e., $q > r$, by using $\hat{\boldsymbol{A}} = \boldsymbol{H}\boldsymbol{A}$, where $\boldsymbol{H} \in \mathbb{R}^{q\times r}$ is the encoding matrix with the property that any r row vectors are linearly independent from each other [28]. In other words, we can use any r rows of $\boldsymbol{H}$ to create an $r \times r$ full-rank matrix. Note that this encoding procedure is performed offline and $\hat{\boldsymbol{A}}$ can be considered to be pre-stored in the system. Similar to the uncoded computing scheme, matrix $\hat{\boldsymbol{A}}$ can then be divided into N sub-matrices $\hat{\boldsymbol{A}}_1, \hat{\boldsymbol{A}}_2, \cdots, \hat{\boldsymbol{A}}_N$, where $\hat{\boldsymbol{A}}_i \in \mathbb{R}^{\ell_i \times m}, \forall i \in [N]$, $\sum_{i=1}^{N} \ell_i = q$, and each worker node i calculates $\hat{\boldsymbol{y}}_i = \hat{\boldsymbol{A}}_i\boldsymbol{x}$.

Different from the uncoded computing scheme, the master node does not need to wait for all worker nodes to complete their calculations, because it can recover $\boldsymbol{A}\boldsymbol{x}$ once the total number of rows of the received results is equal to or larger than r. In particular, suppose the master node receives $\hat{\boldsymbol{y}}_b \in \mathbb{R}^r$ at a certain time t, it can first infer that $\hat{\boldsymbol{y}}_b$ must satisfy

$$\hat{\boldsymbol{y}}_b = \hat{\boldsymbol{H}}_b\boldsymbol{A}\boldsymbol{x},$$

where $\hat{\boldsymbol{H}}_b \in \mathbb{R}^{r\times r}$ is a sub-matrix of the encoding matrix $\boldsymbol{H}$ corresponding to $\hat{\boldsymbol{y}}_b$. The master node can then calculate

$$\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x} = \hat{\boldsymbol{H}}_b^{-1}\hat{\boldsymbol{y}}_b. \tag{1}$$

3.2.3 BPCC

In the literature, most existing CDC schemes assume that each worker node i sends the complete $\hat{\boldsymbol{y}}_i$ to the master node when it is ready, which may incur large delays. To further speed up the computation, we propose a novel BPCC scheme and the main idea is to allow each worker node to return *partial results* to the master node.

Specifically, we consider that each worker node i equally divides the pre-stored encoded matrix $\hat{\boldsymbol{A}}_i$ row-wise into p_i sub-matrices, named as *batches*, where $p_i \in \mathbb{Z}^+$ is the number of batches and $p_i \leq \ell_i$. Except the last batch, each batch has $\lceil \frac{\ell_i}{p_i} \rceil = b_i$ rows. After receiving the input vector $\boldsymbol{x}$ from the master node, the worker node multiplies each batch with $\boldsymbol{x}$ and will send back the partial results once available. Suppose that the master node receives $s_i(t)$ batches from the worker node i by time t, where $0 \leq s_i(t) \leq p_i$, it can then recover the final result when $\sum_{i=1}^{N} \min(\ell_i, s_i(t)b_i) \geq r$, by using Eq. (1).

3.3 Problem Formulation

In the previous sub-section, we introduced the key idea of the BPCC scheme. In the following study, we focus on optimizing the performance of BPCC. Specifically, we consider minimizing the task completion time. This is achieved by allocating proper computation load (i.e., ℓ_i) to each worker node.

We now define T as the amount of time to complete a computation task. Given the number of batches for each worker node $\boldsymbol{p} = (p_1, p_2, \ldots, p_N)$, where $p_i \in \mathbb{Z}^+$, $\forall i \in [N]$, the optimization can be formulated as follows:

$$\begin{aligned} \mathcal{P}_{\text{main}} : \underset{\boldsymbol{\ell}}{\text{minimize}} \quad & \mathbb{E}[T] \\ \text{subject to} \quad & \ell_i \in \mathbb{Z}^+, \forall i \in [N] \\ & \ell_i \geq p_i, \forall i \in [N] \end{aligned} \tag{2}$$

where $\boldsymbol{\ell} = (\ell_1, \ell_2, \ldots, \ell_N)$.

To facilitate further analysis, we assume that the computation task scales with N, i.e., $r = \Theta(N)$. Next, we assume that the computing nodes are fixed with time-invariant computation capabilities, and the network maintains a stable communication delay during the computing process.

We now consider the behavior of waiting time, which is defined as the duration from the time that the master node distributes $\boldsymbol{x}$ to the time that it receives a certain result. For BPCC, we let $T_{k,i}$ be the waiting time for the master node to receive k batches from worker node i, $k \in \mathbb{Z}^+$. Clearly, $T_{k,i}$ can be modeled as a random variable following a certain probability distribution. Following the modeling technique used in recent studies [19], [20], [22], [36], we consider that $T_{k,i}$ follows a shifted exponential distribution defined below:

$$\Pr(T_{k,i} \leq t) = \begin{cases} 1 - e^{-\mu_i(\frac{t}{kb_i} - \alpha_i)} & \text{if } t \geq kb_i\alpha_i \\ 0 & \text{otherwise,} \end{cases} \tag{3}$$

where μ_i and α_i are straggling and shift parameters, respectively, and μ_i and α_i are **positive** constants for all $i \in [N]$. Furthermore, we assume that $T_{k,i}$ is independent from $T_{k',j}$, $\forall j \in [N]$, $j \neq i$, $k' \in \mathbb{Z}^+$.

Based on the above definitions and assumptions, we see that T must satisfy $\sum_{i=1}^{N} s_i(T)b_i \geq r$. In the following sections, we will first discuss how to solve the optimization problem, in which we will conduct theoretical analysis to show the optimality and advantages of BPCC. We will then conduct extensive simulation and real experimental studies to validate the assumptions and to evaluate performance of the optimization algorithm.

4 Main Results

In this section, we aim to solve the optimization problem $\mathcal{P}_{\text{main}}$. In particular, we will first provide a simplified formulation, for which we then apply a two-step alternative formulation. Next, we show how to solve the alternative problems and prove the optimality of the solution. We then analyze the impact of parameter $p_i, \forall i \in [N]$ on the solution, and finally prove that this solution outperforms a recent CDC scheme without batch processing.

4.1 Notations for Asymptotic Analysis

For any two given functions $f(n)$ and $g(n)$, $f(n) = \Theta(g(n))$ if and only if there exist positive constants c_1, c_2, and n_0 such that $0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$ for all $n \geq n_0$; $f(n) = \mathcal{O}(g(n))$ if and only if there exist constants n_0 and c such that $f(n) \leq cg(n)$ for all $n \geq n_0$; and $f(n) = o(g(n))$, if and only if $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$.

4.2 A Simplified Formulation

We relax the constraint from $\ell_i \in \mathbb{Z}^+$ to $\ell_i \geq 0$, $\forall i \in [N]$ to simplify the analysis. We also remove the constraint $\ell_i \geq p_i$, $\forall i \in [N]$, by assuming that $p_i \in \mathbb{Z}^+$ is properly selected such that the optimal solution satisfies this constraint. Consequently, the problem in Eq. (2) can be formulated as follows:

$$\begin{aligned} \mathcal{P'}_{\text{main}} : \underset{\boldsymbol{\ell}}{\text{minimize}} \quad & \mathbb{E}[T] \\ \text{subject to} \quad & \ell_i \geq 0, \forall i \in [N], \end{aligned}$$

Once the above problem is solved, we can round each optimal load number ℓ_i up to its nearest integer using the ceiling function (denoted as $\lceil\ \rceil$). Note that the effect of this rounding step is negligible in practical applications with large load numbers, such as those considered in our simulation and experimental studies [22]. In cases when the derived load number ℓ_i is smaller than p_i, we reduce the value of p_i until this assumption holds. Note that we can always find such p_i that satisfies the constraint, as the derived load number ℓ_i is always larger than or equal to 1.

4.3 A Two-Step Alternative Formulation

To solve the above problem, which is NP-Hard, we provide a two-step alternative formulation, inspired by [22]. We will show later that this alternative formulation provides an asymptotically optimal solution to problem $\mathcal{P'}_{\text{main}}$.

The key idea of the two-step alternative formulation is to first maximize the amount of results accumulated at the master node by a feasible time t, i.e., $t \geq \max_i\{\alpha_i \ell_i\}$, and then minimize time t such that sufficient amount of results are available to recover the final result. In particular, we let $S(t) = \sum_{i=1}^{N} s_i(t)b_i$ be the amount of results received

by the master node by time t, where $b_i = \frac{\ell_i}{p_i}$ is the batch size. For a feasible time t, we first maximize the expected amount of results received by the master node, through solving the following problem:

$$\begin{aligned}\mathcal{P}_{\text{alt}}^{(1)}: \underset{\boldsymbol{\ell}}{\text{maximize}} \quad & \mathbb{E}\left[S(t)\right] \\ \text{subject to} \quad & \ell_i \geq 0, \forall i \in [N]\end{aligned}$$

After obtaining the solution to $\mathcal{P}_{\text{alt}}^{(1)}$, denoted as $\boldsymbol{\ell}^*(t) = (\ell_1^*(t), \cdots, \ell_N^*(t))$, we then minimize the time t such that there is a high probability that the results received by the master node by time t are sufficient to recover the final result, by solving

$$\begin{aligned}\mathcal{P}_{\text{alt}}^{(2)}: \text{minimize} \quad & t \\ \text{subject to} \quad & \Pr\left[S^*(t) < r\right] = o\left(\frac{1}{N}\right)\end{aligned}$$

where $S^*(t)$ is the amount of results received by the master node by time t for load allocation $\boldsymbol{\ell}^*(t)$.

4.4 Solution to the Two-Step Alternative Problem

To solve the two-step alternative problem, we first consider $\mathcal{P}_{\text{alt}}^{(1)}$. Note that, the expected amount of results received by the master node by time t is:

$$\begin{aligned}\mathbb{E}[S(t)] &= \sum_{i=1}^{N} \mathbb{E}[s_i(t) b_i] \\ &= \sum_{i=1}^{N} b_i \left[\sum_{k=1}^{p_i} k \Pr[s_i(t) = k]\right]\end{aligned} \tag{4}$$

where $s_i(t)$ is an integer in range $0 \leq s_i(t) \leq p_i$, and $\Pr[s_i(t) = k]$ is the probability that the master node receives exactly k batches from worker node i,

$$\begin{aligned}&\Pr[s_i(t) = k] \\ &= \begin{cases} 1 - \Pr(T_{1,i} \leq t), & k = 0 \\ \Pr(T_{k,i} \leq t) - \Pr(T_{k+1,i} \leq t), & 0 < k < p_i \\ \Pr(T_{p_i,i} \leq t). & k = p_i \end{cases}\end{aligned}$$

$\mathbb{E}[S(t)]$ in Eq. (4) can then be computed by:

$$\begin{aligned}\mathbb{E}[S(t)] &= \sum_{i=1}^{N} b_i \left[\sum_{k=1}^{p_i - 1} k \Pr[s_i(t) = k] + p_i \Pr[s_i(t) = p_i]\right] \\ &= \sum_{i=1}^{N} \sum_{k=1}^{p_i} b_i \Pr(T_{k,i} \leq t) \\ &= \sum_{i=1}^{N} \sum_{k=1}^{p_i} b_i \left(1 - e^{-\mu_i (\frac{t}{k b_i} - \alpha_i)}\right) \\ &= \sum_{i=1}^{N} \left(\ell_i - b_i \sum_{k=1}^{p_i} e^{-\mu_i (\frac{t}{k b_i} - \alpha_i)}\right) \\ &= \sum_{i=1}^{N} \left(\ell_i - \frac{\ell_i}{p_i} \sum_{k=1}^{p_i} e^{-\mu_i (\frac{t p_i}{k \ell_i} - \alpha_i)}\right)\end{aligned} \tag{5}$$

The solution to $\mathcal{P}_{\text{alt}}^{(1)}$ can then be obtained by solving the following equation for each $i \in [N]$:

$$\frac{\partial}{\partial \ell_i} \mathbb{E}\left[S(t)\right] = 1 - \left[\sum_{k=1}^{p_i} \left(\frac{1}{p_i} + \frac{\mu_i t}{\ell_i k}\right) e^{-\mu_i (\frac{t p_i}{k \ell_i} - \alpha_i)}\right] = 0,$$

which yields:

$$\ell_i^*(t) = \frac{t}{\lambda_i} \tag{6}$$

λ_i is the positive solution to the following equation:

$$\sum_{k=1}^{p_i} \left(\frac{1}{p_i} + \frac{\mu_i \lambda_i}{k}\right) e^{-\mu_i (\frac{\lambda_i p_i}{k} - \alpha_i)} = 1, \tag{7}$$

which is a constant independent of t. To show that Eq. (7) has a single positive solution, we can define an auxiliary function f_i for each i:

$$f_i(x) = \sum_{k=1}^{p_i} \left(\frac{1}{p_i} + \frac{\mu_i x}{k}\right) e^{-\mu_i (\frac{x p_i}{k} - \alpha_i)}.$$

We can see that $f_i(x)$ decreases monotonically with the increase of x when $x > 0$. We can also find that $f_i(0) = e^{\mu_i \alpha_i} > 1$ and $f_i(\infty) = 0$. Based on these statements, we know that a unique λ_i exists and can be efficiently solved using a numerical approach. Next, we show in Lemma 1 that λ_i has closed-form infimum and supremum.

Lemma 1. *Let λ_i, $i \in [N]$, be the positive solution to Eq. (7). Its infimum is given by*

$$\inf \lambda_i = \lim_{p_i \to \infty} \lambda_i = \alpha_i, \tag{8}$$

In addition, its supremum is given by

$$\sup \lambda_i = \frac{W(-e^{-\alpha_i \mu_i - 1}) + 1}{-\mu_i}, \tag{9}$$

which is attained when $p_i = 1$ and $W(\cdot)$ is the Lambert W function [38].

From Lemma 1, we can derive that the condition $t \geq \max_i\{\alpha_i \ell_i(t)\}$ holds, as $t = \ell_i^*(t)\lambda_i \geq \ell_i^*(t)\alpha_i$ for each work node i.

Next, we solve $\mathcal{P}_{\text{alt}}^{(2)}$. Since this problem is also NP-hard, we here provide an approximated solution. In particular, we approximate its optimal solution, denoted as t^*, with value τ^*, such that the expected amount of results accumulated at the master node by time τ^* equals to the amount of results required for recovering the final result, i.e., $\mathbb{E}[S^*(\tau^*)] = r$. To find the value of τ^*, we let

$$\mathbb{E}[S^*(t)] = r. \tag{10}$$

Then, using the load allocation $\ell_i^*(t)$ in Eq. (6), the expected amount of results received by the master node is:

$$\begin{aligned}\mathbb{E}[S^*(t)] &= \sum_{i=1}^{N} \left(\ell_i^*(t) - \frac{\ell_i^*(t)}{p_i} \sum_{k=1}^{p_i} e^{-\mu_i (\frac{t p_i}{k \ell_i^*(t)} - \alpha_i)}\right) \\ &= \sum_{i=1}^{N} \frac{t}{\lambda_i} \left(1 - \frac{1}{p_i} \sum_{k=1}^{p_i} e^{-\mu_i (\frac{\lambda_i p_i}{k} - \alpha_i)}\right).\end{aligned} \tag{11}$$

Algorithm 1: BPCC

Input: $r, N, \boldsymbol{p} = \{p_1, \ldots, p_N\}, \boldsymbol{\mu} = \{\mu_1, \ldots, \mu_N\}, \boldsymbol{\alpha} = \{\alpha_1, \ldots, \alpha_N\}$
Output: $\boldsymbol{\ell}$
for $i = 1 : N$ **do**
  Calculate λ_i by solving Eq. (7)
Calculate β by using Eq. (13)
for $i = 1 : N$ **do**
  Calculate ℓ_i^* by using Eq. (14)
Return $\boldsymbol{\ell} = \{\lfloor \ell_1^* \rfloor, \lfloor \ell_2^* \rfloor, \cdots, \lfloor \ell_N^* \rfloor\}$

We can then find the solution to Eq. (10) as follows:

$$\tau^* = \frac{r}{\beta} \tag{12}$$

where

$$\beta = \sum_{i=1}^{N} \frac{1}{\lambda_i} \left(1 - \frac{1}{p_i} \sum_{k=1}^{p_i} e^{-\mu_i (\frac{\lambda_i p_i}{k} - \alpha_i)} \right), \tag{13}$$

which is also a constant.

Combining the solutions to $\mathcal{P}_{\text{alt}}^{(1)}$ and $\mathcal{P}_{\text{alt}}^{(2)}$, we can then derive the load allocation:

$$\ell_i^*(\tau^*) = \frac{r}{\beta \lambda_i} \tag{14}$$

The procedures of BPCC are summarized in Algorithm 1.

4.5 Optimality Analysis

In this sub-section, we conduct theoretical analysis to investigate the performance of BPCC. Specifically, we first show in Lemma 2 the optimality of the approximated solution τ^* to $\mathcal{P}_{\text{alt}}^{(2)}$. We then show in Theorem 3 that the solution provided by BPCC is asymptotically optimal. Finally, we show in Theorem 4 the accuracy of τ^* in approximating the expected execution time of BPCC.

Lemma 2. *Let t^* be the optimal solution to $\mathcal{P}_{alt}^{(2)}$, and τ^* be the approximated solution given by Eq. (12). If the batch processing time follows the shifted exponential distribution in Eq. (3) and $r = \Theta(N)$, then*

$$\tau^* - o(1) < t^* \leq \tau^* + o(1). \tag{15}$$

Based on Lemma 2, we next show the asymptotic optimality of BPCC in Theorem 3.

Theorem 3. *Consider problem $\mathcal{P}'_{\text{main}}$ with the batch processing time following the shifted exponential distribution in Eq. (3) and $r = \Theta(N)$. Let $\mathbb{E}[T_{\text{BPCC}}]$ and $\mathbb{E}[T_{\text{OPT}}]$ be the expected execution time of BPCC and the optimal value of $\mathcal{P}'_{\text{main}}$, respectively. The BPCC is asymptotically optimal, i.e.,*

$$\lim_{N \to \infty} \mathbb{E}\left[T_{\text{BPCC}}\right] = \lim_{N \to \infty} \mathbb{E}\left[T_{\text{OPT}}\right] \tag{16}$$

Theorem 3 and Lemma 2 further lead to the following theorem.

Theorem 4. *Let τ^* be the approximated solution given by Eq. (12) and $\mathbb{E}[T_{\text{BPCC}}]$ be the expected execution time of BPCC. If the batch processing time follows the shifted exponential distribution in Eq. (3) and $r = \Theta(N)$, then*

$$\tau^* = \lim_{N \to \infty} \mathbb{E}\left[T_{\text{BPCC}}\right] \tag{17}$$

4.6 Analysis of the Impact of Parameter p

In the BPCC scheme shown in Algorithm 1, we note that $\boldsymbol{p}$ is the only parameter that can be tuned, while the other parameters, including r, N, $\boldsymbol{u}$ and $\boldsymbol{\alpha}$, are determined by the specific computation task and properties of the distributed computing system. In this sub-section, we analyze the impact of this important parameter $\boldsymbol{p}$ on the performance of BPCC in Theorem 5. We then show in Theorem 6 that the approximated execution time of BPCC, i.e., τ^* given by Eq. (12), has closed-form infimum and supremum.

Theorem 5. *Consider problem $\mathcal{P}'_{\text{main}}$ with the batch processing time following the shifted exponential distribution in Eq. (3) and $r = \Theta(N)$. Let τ^* be the approximated execution time of BPCC given by Eq. (12). Then the increase of any p_i, $i \in [N]$, will cause τ^* to decrease.*

Theorem 6. *Consider problem $\mathcal{P}'_{\text{main}}$ with the batch processing time following the shifted exponential distribution in Eq. (3) and $r = \Theta(N)$. Let τ^* be the approximated execution time of BPCC given by Eq. (12). Then*

$$\begin{aligned} \inf \tau^* &= \lim_{p_i \to \infty, \forall i \in [N]} \tau^* \\ &= \frac{r}{\sum_{i=1}^{N} \frac{1}{\alpha_i} (1 - e^{\mu_i \alpha_i} \int_0^1 e^{-\frac{\mu_i \alpha_i}{x}} dx)}, \end{aligned} \tag{18}$$

and

$$\begin{aligned} \sup \tau^* &= \max \tau^* \\ &= \sum_{i=1}^{N} \frac{1}{\sup \lambda_i} \left(1 - e^{-\mu_i (\sup \lambda_i - \alpha_i)} \right), \end{aligned} \tag{19}$$

which is attained when $p_i = 1$, $\forall i \in [N]$. Here $\sup \lambda_i$ is given by Eq. (9).

From Theorem 6 and Eq. (14), we can derive the following corollary.

Corollary 6.1. *Consider problem $\mathcal{P}'_{\text{main}}$ with the batch processing time following the shifted exponential distribution in Eq. (3) and $r = \Theta(N)$. Let ℓ_i^* be the solution of BPCC given by Eq. (14). Then when the approximated execution time τ^* of BPCC given by Eq. (12) converges to its infimum, ℓ_i^* converges to $\hat{\ell}_i$, where*

$$\hat{\ell}_i = \frac{r}{\alpha_i \sum_{j=1}^{N} \frac{1}{\alpha_j} (1 - e^{\mu_j \alpha_j} \int_0^1 e^{-\frac{\mu_j \alpha_j}{x}} dx)}. \tag{20}$$

4.7 Comparison with HCMM

In this sub-section, we compare the performance of BPCC with HCMM [22], a state-of-the-art CDC scheme for heterogeneous worker nodes, and show that BPCC outperforms HCMM in computational efficiency.

HCMM can be considered as a special case of BPCC with $p_i = 1, \forall i \in [N]$. It assigns each worker node i with load $\ell_{H,i} = \frac{r}{\beta_H \lambda_{H,i}}$, where $\lambda_{H,i}$ is the positive solution to $e^{\mu_i \lambda_{H,i}} = e^{\alpha_i \mu_i}(\mu_i \lambda_{H,i} + 1)$ and $\beta_H = \sum_{i=1}^{N} \frac{\mu_i}{1+\mu_i \lambda_{H,i}}$. Theorem 7 shows that BPCC is more efficient than HCMM.

Theorem 7. *Consider problem $\mathcal{P}'_{\text{main}}$, with the batch processing time following a shifted exponential distribution in Eq. (3) and $r = \Theta(N)$. Let T_{BPCC} and T_{HCMM} be the execution times of BPCC and HCMM, respectively. Then,*

$$\lim_{N\to\infty} \mathbb{E}\left[T_{\text{BPCC}}\right] \leq \lim_{N\to\infty} \mathbb{E}\left[T_{\text{HCMM}}\right]$$

5 Simulation Studies

In this section, we conduct simulation studies to evaluate the performance of the proposed BPCC scheme. Specifically, we first explain the simulation settings, including the distributed computing schemes and scenarios. We then elaborate on the impact of important parameters, including p_i, N, μ_i and α_i, on the performance of the BPCC scheme. Finally, we compare the proposed BPCC scheme with benchmark schemes, including the state-of-the-art HCMM scheme [22].

5.1 Simulation Settings

5.1.1 Distributed Computing Schemes

In this study, we consider four distributed computing schemes:

- **Uniform Uncoded:** This method divides the computation loads equally, i.e., $\ell_i = \frac{r}{N}, \forall i \in [N]$.
- **Load-Balanced Uncoded [22]:** This method divides the computation loads according to the computing capabilities of the worker nodes. In particular, the computation load assigned to each worker node i is inversely proportional to the expected time for this node to compute an inner product, i.e., $\ell_i \propto (\frac{\mu_i}{\mu_i \alpha_i + 1})$ and $\sum_{i=1}^{N} \ell_i = r$.
- **HCMM [22]:** In this method, the load assignment method in [22] is used. Note that this is a special case of Algorithm 1, in which $p_i = 1, \forall i \in [N]$. The HCMM and BPCC have the exactly same load allocation for each worker.
- **BPCC:** In this scheme, Algorithm 1 is used, where $\boldsymbol{p}$ are the parameters to configure.

5.1.2 Computation Scenarios

To evaluate the performance of different distributed computing schemes, we consider the following four computation scenarios:

- **Scenario 1:** $r = 1 \times 10^4$ and $N = 10$.
- **Scenario 2:** $r = 2 \times 10^4$ and $N = 10$.
- **Scenario 3:** $r = 1 \times 10^4$ and $N = 20$.
- **Scenario 4:** $r = 2 \times 10^4$ and $N = 20$.

5.1.3 Simulation Method

In our simulation, we implement all the aforementioned distributed computing schemes in MATLAB. We assume that the processing time of each node follows the shifted exponential distribution in Eq. (3). Specifically, for each experiment of a scenario, we choose the straggling parameters $\mu_i, \forall i \in [N]$ randomly in $[1, 50]$, and calculate each shift parameter $\alpha_i = \frac{1}{\mu_i}$. In each experiment, we simulate every distributed computing scheme for 100 times, in each of which the computing time of a node is simulated by using its straggling and shift parameters.

5.2 Parameter Impact Analysis

In this sub-section, we investigate the impacts of parameters in BPCC, including number of batches, number of worker nodes, and the straggling and shift parameters in the computing model.

5.2.1 Number of batches

The number of batches p_i is an important parameter to configure. In Section 4.6, we have theoretically analyzed its impact on the performance of BPCC. Here we conduct simulation studies to demonstrate its impact described in Theorem 5, Theorem 6 and Corollary 6.1. In particular, two experiments are designed.

In the first experiment, we show that the approximated execution time τ^* of BPCC given by Eq. (12) decreases with the increase of any p_i, $i \in [N]$, as presented in Theorem 5. In particular, we vary the number of batches for one of the worker nodes and fix the number of batches for the others. Specially, we vary p_1 and let $p_j = 1, \forall j \in [N] \setminus \{1\}$. As shown in Fig. 1(a), τ^* indeed decreases as p_1 increases.

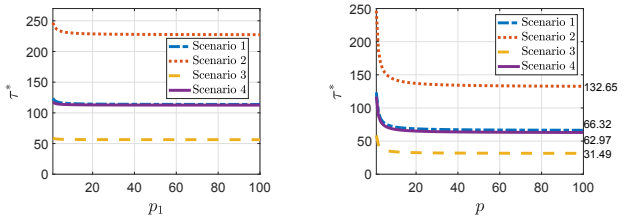
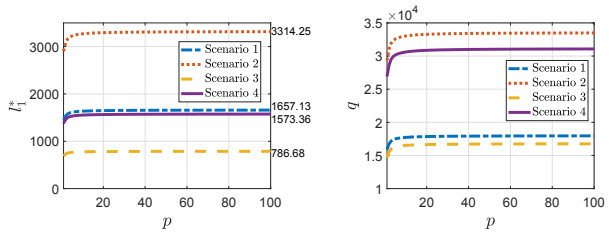


Figure 1. The approximated execution time τ^* of BPCC at different values of a) p_1, when $p_j = 1, \forall j \in [N] \setminus \{1\}$, and b) p, when $p_i = p, \forall i \in [N]$, in different scenarios.

In the second experiment, we show that the approximated execution time τ^* and the load ℓ_i^* tend to converge as p_i increases for all $i \in [N]$, as presented in Theorem 6 and Corollary 6.1. In this experiment, we vary p_i simultaneously for all $i \in [N]$. In other words, we let $p_i = p \in \mathbb{Z}^+$, $\forall i \in [N]$ and vary the value of p. As shown in Fig. 1(b), τ^* decreases with the increase of p and finally converges. Note that when $p = 100$, τ^* equals to 66.32, 132.65, 31.49, 62.97 for the four scenarios, respectively, which are already very close to its theoretical infimum 65.77, 131.54, 31.22, 62.45, computed by Eq. (18). Fig. 2(a) shows the trajectory

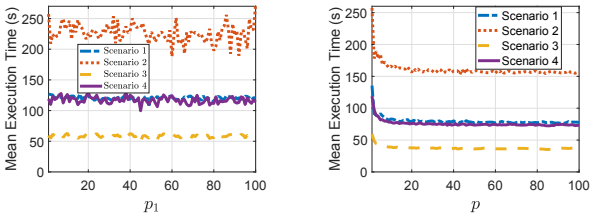


(a) (b)

Figure 2. The value of a) load ℓ_1^* and b) total load $q = \sum_{i=1}^{N} \ell_i^*$ at different values of p, when $p_i = p, \forall i \in [N]$, in different scenarios.

of the load allocated to one of the worker nodes, i.e., ℓ_1^*, which decreases and finally converges as p increases. Note that when $p = 100$, ℓ_1^* equals to 1657.13, 3314.25, 786.68, 1573.36 for the four scenarios, respectively, which are very close to the values of $\hat{\ell}_1$ given by Eq. (20), i.e., 1659.79, 3319.59, 787.95, 1575.90. Of interest, if we set $p_i = \lfloor \hat{\ell}_i \rfloor$ given by Eq. (20), $\forall i \in [N]$, τ^* equals to 65.81, 131.58, 31.26, 62.47 and ℓ_1^* equals to 1659.62, 3319.42, 787.73, 1575.68 for the four scenarios, respectively, which are almost the same as the associated $\inf \tau^*$ and $\hat{\ell}_1$, respectively.

Fig. 2(a) shows the impact of parameter p_i on the load ℓ_i^* for one of the work nodes. In Fig. 2(b), we also show its impact on the total load $q = \sum_{i=1}^{N} \ell_i^*$. As we can see, the total load q also increases with the increase of p, where $p_i = p$, $\forall i \in [N]$. This indicates that a larger p_i will require more storage space at the worker nodes. Note that the worker nodes will stop execution once the master node receives sufficient amount of results for recovering the final result. Therefore, a larger total load q does not increase the computation load for the worker nodes. This study tells us that the configuration of parameter p_i should trade off between computational efficiency and storage consumption.



(a) (b)

Figure 3. The value of a) load ℓ_1^* and b) total load $q = \sum_{i=1}^{N} \ell_i^*$ at different values of p, when $p_i = p, \forall i \in [N]$, in different scenarios.

As τ^* is an approximation of BPCC's execution time, we also show in Fig. 3 the impact of p_i on the expected execution time $\mathbb{E}[T_{\text{BPCC}}]$ of BPCC, which is estimated using the Monte Carlo simulation method, specifically, by repeating each experiment for 100 times and averaging the times to execute the BPCC scheme. Comparing Fig. 1 and Fig. 3, we can see that τ^* approximates $\mathbb{E}[T_{\text{BPCC}}]$ generally well. The fluctuations are caused by the uncertainty of the computation times and the relatively weak estimation capability of the Monte Carlo method, which requires large number of simulations to obtain an accurate mean estimate. As we will show in the next study, the approximation accuracy of τ^* is impacted by the number of worker nodes N.

5.2.2 Number of worker nodes

As we have theoretically proved in Theorem 4, the approximated execution time τ^* converges to the true expected execution time $\mathbb{E}[T_{\text{BPCC}}]$ of BPCC, when the number of worker nodes N approaches infinity. To demonstrate this theorem, we vary N and set $r = 100N + 10000$, and record the approximation error of τ^*, given by $|\tau^* - \mathbb{E}[T_{\text{BPCC}}]|$, for each value of N. The results are shown in Fig. 4. Note that, instead of the four scenarios described in Section 5.1.2, we design four new scenarios for this study, where the configuration of each scenario is specified in the figure. As we can see, the approximation error of τ^* decreases with the increase of the number of worker nodes, and finally converges to zero.

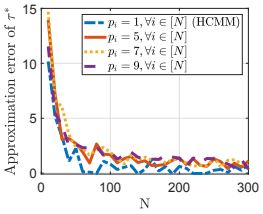


Figure 4. The approximation error of τ^* at different values of N with $r = 100N + 10000$.

From the above studies, we can see that, as the number of batches p_i for any worker node $i \in [N]$ increases, the efficiency of BPCC improves, but the demand for storage also increases. Because storage consumption is not our main concern in this study, in the following experiments, we set p_i to its maximum value possible, i.e., $p_i = \lfloor \hat{\ell}_i \rfloor$, $\forall i \in [N]$, considering that a valid p_i should be a positive integer smaller than or equal to ℓ_i^* and ℓ_i^* converges to $\hat{\ell}_i$ as p_i, $\forall i \in [N]$, increases.

5.2.3 Straggling and shift parameters

In BPCC, to determine the load numbers ℓ_i, we need to know the values of the straggling and shift parameters, μ_i and α_i, which are estimated by measuring the actual execution behaviors in real experiments. To understand the impact of parameter estimation errors to the performance of BPCC, we conduct a sensitivity study. In particular, to study how sensitive BPCC is to the estimation errors associated with the straggling parameters μ_i, we fix the shift parameters α_i and deviate each μ_i from its true value by randomly picking a value from the interval $(\mu_i^{min}, \mu_i^{max})$, where $\mu_i^{min} = \mu_i^*(1-\Delta)$, $\mu_i^{max} = \mu_i^*(1+\Delta)$, μ_i^* is the true value and $\Delta > 0$ represents the degree of deviation. As μ_i

This article has been accepted for publication in IEEE Transactions on Network Science and Engineering. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TNSE.2021.3095040

should be positive, we let $\mu_i^{min} = 0$, if $\Delta > 1$. Fig. 5(a) shows the relative change of the mean execution time, measured by $\frac{\hat{\mathbb{E}}'[T]-\hat{\mathbb{E}}[T]}{\hat{\mathbb{E}}[T]}$, at different values of Δ in different scenarios, where $\hat{\mathbb{E}}[T]$ and $\hat{\mathbb{E}}'[T]$ are the mean execution time obtained by using the true and erroneous parameter values, respectively. Similarly, we plot in Fig. 5(b) the relative change of the mean execution time when the shift parameters α_i suffer from estimation errors. As we can see, the deviation of straggling parameters μ_i has less impact on the performance of BPCC than that of shift parameters α_i, and BPCC is robust to small errors in general.

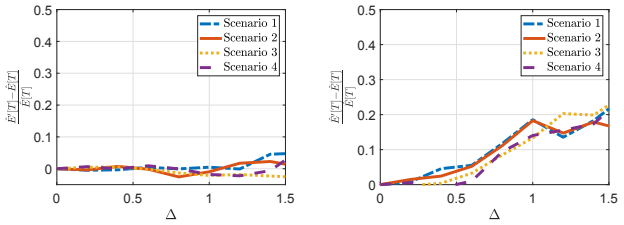


Figure 5. Relative change of the mean execution time when a) the straggling parameters μ_i and b) the shift parameters α_i suffer from different degrees of deviation from their true values in different scenarios.

5.3 Comparative Performance Studies

In this sub-section, we compare the performance of the proposed BPCC scheme with three benchmark schemes, including Uniform Uncoded, Load-Balanced Uncoded and HCMM. The parameter p_i in BPCC is set to $p_i = \lfloor \hat{\ell}_i \rfloor$, $\forall i \in [N]$.

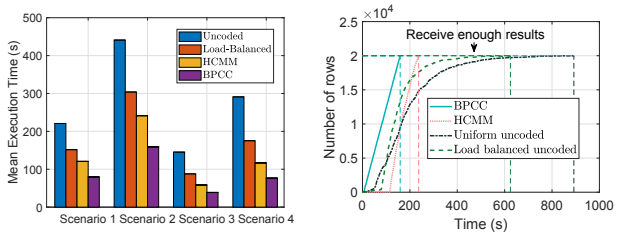


Figure 6. a) Comparison of the mean execution time of different schemes in different scenarios. b) The average total number of rows of inner product results received by the master node over time for different schemes in Scenario 2.

Fig. 6(a) shows the mean execution times for all schemes, grouped by the computation scenario. We can clearly observe that the proposed BPCC scheme outperforms other benchmark schemes in all scenarios. For instance, BPCC achieves performance improvement of up to 73% over the Uniform Uncoded scheme, up to 56% over Load-Balanced Uncoded scheme, and up to 34% over HCMM. Note that the execution times are directly derived by using the computing model in Eq. (3) and the decoding times are not considered here.

In Fig. 6(a), the performance is expressed in terms of the mean execution time, which corresponds to $\mathbb{E}[T_{\text{BPCC}}]$ for different schemes. In Fig. 6(b), we show the average amount of received results over time for Scenario 2, which corresponds to $\mathbb{E}[S(t)]$ in the theoretical analysis. Remarkably, we can observe from the figure that the master node can quickly receive results from the worker nodes from the very beginning. On the other hand, under the three benchmark schemes, there is a certain duration at the beginning when the master node does not receive any result. This phenomenon occurs because our BPCC scheme allows partial results to be returned, which is very useful for certain applications that can utilize partial results. In Fig. 6(b), we also indicate the time when the master node receives the required amount of results, i.e., r. Such a time corresponds to τ^* (i.e., $\mathbb{E}[S(\tau^*)] = r$).

6 Experiments on the Amazon EC2 Computing Cluster

In this section, we evaluate the performance of the proposed BPCC scheme in the real distributed computing system. Specifically, we implement three benchmark schemes and the proposed BPCC scheme in the Amazon EC2 computing platform [39], which is a classical cloud computing system.

6.1 Experiment Settings

To implement the proposed BPCC and the three benchmark schemes over Amazon EC2 clusters [1], we apply a standard distributed computing interface, *Message Passing Interface* (MPI) [40], by using an open-source package: `mpi4py` [41], which provides interfaces in Python. Moreover, to encode and decode matrices in BPCC, we use the Luby Transform (LT) codes with peeling decoder [37] that are adopted by HCMM [22]. The utilization of LT code relaxes the constraint of recovering the final computation result from any r rows to any $r(1+\epsilon)$ rows, where $\epsilon > 0$ is desired to be as small as possible. In this study, we adopt the configuration in [22] and set $\epsilon = 0.13$. The parameter p_i in BPCC is set to $p_i = \lfloor \hat{\ell}_i \rfloor$, $\forall i \in [N]$.

To evaluate the performance of the four computation schemes, we consider the following four scenarios:

- **Scenario 1:** $r = 0.5 \times 10^4$ and $N = 5$, where one *r4.2xlarge* instance, two *r4.xlarge* instances and two *t2.large* instances are used as the worker nodes.
- **Scenario 2:** $r = 1 \times 10^4$ and $N = 10$, where two *r4.2xlarge instances* instances, four *r4.xlarge* instances, and four *t2.large* instances are used as the worker nodes.
- **Scenario 3:** $r = 1.5 \times 10^4$ and $N = 10$, where four *r4.2xlarge* instance and six *r4.xlarge* instances are used as the worker nodes.

1. The source code can be found at https://github.com/BaoqianWang/Batch-Processing-Coded-Computation.

This article has been accepted for publication in IEEE Transactions on Network Science and Engineering. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TNSE.2021.3095040

- **Scenario 4:** $r = 2 \times 10^4$ and $N = 15$, where seven *r4.2xlarge* instance and eight *r4.xlarge* instances are used as the worker nodes.

In all above scenarios, the master node runs in a *m4.xlarge* instance, and the size of the input vector $\boldsymbol{x} \in \mathbb{R}^m$ is set to $m = 5 \times 10^5$.

6.2 Parameter Estimation

In our previous design and analysis, we have assumed that the task completion time T on each node follows a shifted exponential distribution in a general form:

$$\Pr[T \leq t] = 1 - e^{-\mu(\frac{t}{r} - \alpha)} = 1 - e^{-\frac{\mu}{r}(t - \alpha r)}, \qquad (21)$$

when $t \geq \alpha r$. Therefore, $\mathbb{E}[T] = \frac{r}{\mu} + \alpha r$.

Based on the assumption above, we conduct extensive experiments and measure the actual execution behaviors to estimate the values of the straggling and shift parameters, μ and α, for different types of instances. Particularly, let $t_c(r) = \frac{r}{\mu}$ and $t_0(r) = \alpha r$, we run tasks of different sizes. For each task size r, we execute the task repeatedly for $M = 1000$ times and obtain the execution times $\{T_1, T_2, \ldots, T_M\}$. The maximum likelihood estimates of $t_0(r)$ and $t_c(r)$ are then given by $\hat{t}_0(r) = \min_{l \in [M]} T_l$ and $\hat{t}_c(r) = \frac{1}{M} \sum_{l=1}^{M} T_l - \hat{t}_0(r)$, respectively [42], [43]. With $\hat{t}_0(r)$ and $\hat{t}_c(r)$ for different task sizes r, we can then estimate the values of μ and α by using the least squares estimation. Fig. 7 shows the estimated *cumulative distribution function* (CDF) of the processing time of a t2.xlarge instance when task size $r = 500$. The estimated α and μ for different types of Amazon EC2 instances are summarized in Table 1. These estimated parameters will be used to allocate computation loads for all computing schemes, except the uniform uncoded scheme.

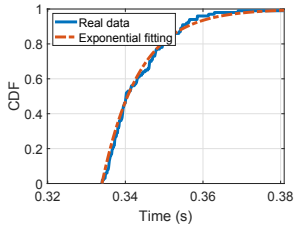


Figure 7. The CDF of the processing time of an Amazon EC2 t2.xlarge instance for computing a task with $r = 500$.

Table 1
Estimated computing parameters of different types of Amazon EC2 instances

Instances	μ	α
r4.xlarge	9.42×10^4	1.75×10^{-4}
r4.2xlarge	9.25×10^4	1.60×10^{-4}
t2.medium	2.15×10^4	5.18×10^{-4}
t2.large	3.90×10^4	2.25×10^{-4}

6.3 Experimental Results

To evaluate the performance of the proposed BPCC scheme running on the heterogeneous Amazon clusters, we design three experiments.

6.3.1 Experiment 1

In this experiment, we compare the performance of BPCC with the three benchmark schemes in different scenarios. For each scenario, we run each scheme 100 times and record the mean execution time ($\mathbb{E}[T]$). To evaluate the robustness of these schemes to uncertain disturbances, we introduce unexpected stragglers that are randomly chosen in each run. In particular, we randomly select 20% of the worker nodes to be stragglers in each run. As stragglers can be slow in computing or returning results (e.g, when communication congestion happens), such stragglers are emulated by delaying the return of computing results such that the computing time observed by the master node is three times of the actual computing time. Fig. 8(a) illustrates the mean execution time of different distributed computing schemes in different scenarios, which also highlights the decoding time required by the coded schemes including BPCC and HCMM. We can see from the figure that the proposed BPCC scheme outperforms all benchmark schemes in all scenarios. Specifically, the performance improves up to 79% compared with Uncoded scheme, up to 78% compared with Load-Balanced scheme, and up to 62% compared with HCMM.

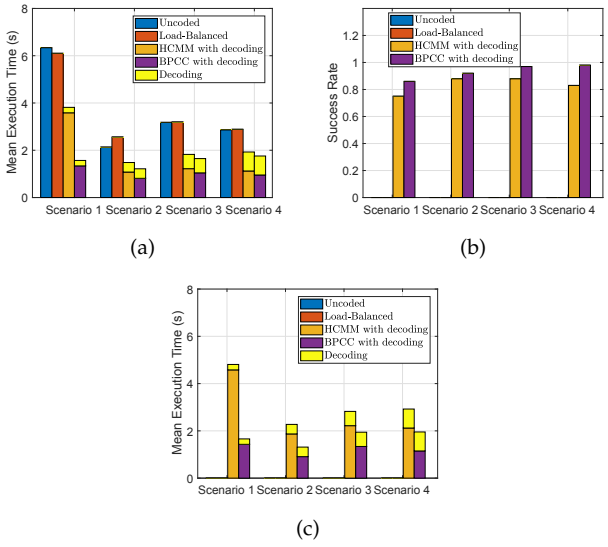


Figure 8. a) The mean execution time of different schemes in different scenarios at the presence of unexpected stragglers with finite delay. The b) success rate and c) mean execution time of different schemes in different scenarios at the presence of unexpected stragglers with infinite delay.

As stragglers can also fail in returning any results (e.g., when nodes/links fail), we also consider such stragglers and emulate them by setting the delay time to infinity. Since no results will be returned by such stragglers, the

This article has been accepted for publication in IEEE Transactions on Network Science and Engineering. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TNSE.2021.3095040

computation task can fail. Fig. 8(b) shows the success rate (measured by the ratio of successful runs) of each scheme in different scenarios. The mean execution time of successful runs is shown in Fig. 8(c). As we can see, the Uncoded and Load-Balanced schemes fail to complete the task in all runs, as no redundancy is introduced in these schemes. Both HCMM and BPCC can successfully complete the task in most runs, but HCMM has a lower success rate and is less efficient than BPCC.

In Fig. 9, we selectively show the average amount of received results over time ($\mathbb{E}[S(t)]$) in Scenario 4. As expected, in our BPCC scheme, the master node continuously receives results from the very beginning. However, in other schemes, the master node needs to wait a long time before receiving any result.

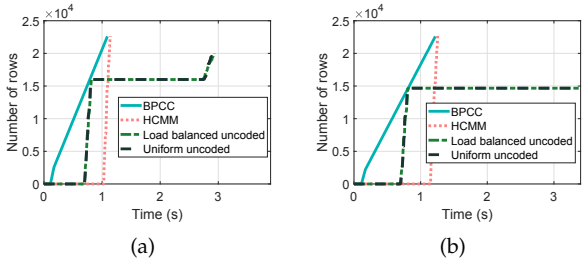


Figure 9. The average total number of rows of inner product results received by the master node over time for different schemes in Scenario 4 at the presence of unexpected stragglers with a) finite and b) infinite delay.

6.3.2 Experiment 2

In the second experiment, we study the impact of the number of unexpected stragglers on the performance of the four computation schemes, by varying the percentage of stragglers from 0% to 60%. Similar as Experiment 1, stragglers can delay in returning results for finite or infinite amount of time.The mean execution time of each scheme at the presence of different numbers of stragglers with finite delay for Scenario 4 is shown in Fig. 10(a). As we can see, when there is no straggler, the Uniform Uncoded scheme and the Load-Balanced Uncoded scheme achieve the best performance, as they do not involve any computation redundancy, compared with the coded schemes. However, when stragglers exist, our BPCC scheme achieves the best performance, indicating its high robustness to uncertain stragglers. We can also observe from Fig. 10(a) that the performances of all schemes degrade with the increase of the number of stragglers. Of interest, the performance degradation of the three benchmark schemes slows down when the number of stragglers reaches to a certain value. This is because worker nodes in these schemes won't return any result to the master node until the whole assigned task is completed and all stragglers would delay returning the result for a period that is three times of the task computation time. We also note that the performance of HCMM is even worse than the two uncoded schemes when the percentage of stragglers exceeds 20%. This is because each worker node in HCMM is assigned with more computation load, compared with the uncoded schemes, which causes the stragglers in HCMM to wait for a longer time before returning any result.

In case when stragglers delay in returning results for infinite amount of time, the success rate and the mean execution time of each scheme are shown in Fig. 10(b) and Fig. 10(c), respectively. As expected, both the Uncoded and Load-Balanced schemes fail to complete the task. Additionally, the performances of BPCC and HCMM degrade with the increase of the number of stragglers, and BPCC outperforms HCMM.

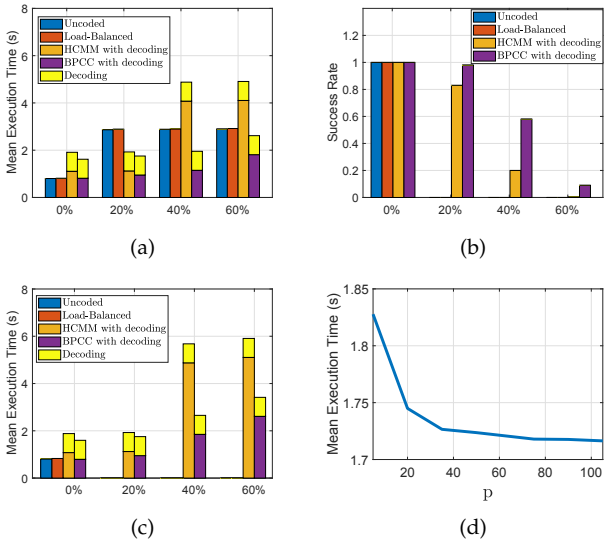


Figure 10. a) The mean execution time of different schemes in Scenario 4 when different percentages of unexpected stragglers with finite delay are present. The b) success rate and c) mean execution time of different schemes in Scenario 4 when different percentages of unexpected stragglers with infinite delay are present. d) The mean execution time of BPCC at different values of p in Scenario 4.

6.3.3 Experiment 3

In the third experiment, we evaluate the impact of the number of batches p_i on the performance of BPCC running on the Amazon clusters. Similar as the simulation study, we let $p_i = p$, $\forall i \in [N]$, and vary the value of p from 5 to 100. Fig. 10(d) shows the mean execution time of BPCC at different values of p under the settings described in Scenario 4 and Experiment 1 when unexpected stragglers with finite delay are present. As expected, the efficiency of BPCC improves with the increase of p.

7 Conclusion

In this paper, we systematically investigated the design and evaluation of a novel *coded distributed computing* (CDC) framework, namely, *batch-processing based coded computing* (BPCC), for heterogeneous computing systems. The key idea of BPCC is to optimally exploit partial coded results

calculated by all distributed computing nodes. Under this BPCC framework, we then investigated a classical CDC problem, matrix-vector multiplication, and formulated an optimization problem for BPCC to minimize the expected task completion time, by configuring the computation load. The BPCC was proved to provide an asymptotically optimal solution and outperform a state-of-the-art CDC scheme for heterogeneous clusters, namely, *heterogeneous coded matrix multiplication* (HCMM). Theoretical analysis reveals the impact of BPCC's key parameter, i.e., number of batches, on its performance, the results of which infer the worst and best performance that BPCC can achieve. To evaluate the performance of the proposed BPCC scheme and better understand the impacts of its parameters, we conducted extensive simulation studies and real experiments on the Amazon EC2 computing clusters. The simulation and experimental results verify theoretical results and also demonstrate that the proposed BPCC scheme outperforms all benchmark schemes in computing systems with uncertain stragglers, in terms of the task completion time and robustness to stragglers. In the future, we will further enhance BPCC by jointly optimizing load allocation and the number of batches to achieve a tradeoff between computational efficiency and storage consumption, and explore its other properties, such as the convergence rate. We will also consider distributed computing systems with mobile computing nodes and other optimization objectives, such as minimizing the energy consumption.

ACKNOWLEDGEMENT

We would like to thank the National Science Foundation (NSF) under Grants CI-1953048/1730589/1730675/1730570/1730325, CAREER-2048266 and CAREER-1714519 for the support of this work.

REFERENCES

[1] S. Kartik and C. S. Ram Murthy, "Task allocation algorithms for maximizing reliability of distributed computing systems," *IEEE Transactions on Computers*, vol. 46, no. 6, pp. 719–724, 1997.

[2] B. Hong and V. K. Prasanna, "Distributed adaptive task allocation in heterogeneous computing environments to maximize throughput," in *Proceedings of the 2004 IPDPS*, Santa Fe, New Mexico, April 2004.

[3] K. Lu, J. Xie, Y. Wan, and S. Fu, "Toward UAV-Based Airborne Computing," *IEEE Wireless Communications*, vol. 26, no. 6, pp. 172–179, 2019.

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

[5] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster Computing with Working Sets," in *Proceedings of the 2010 HotCloud*, Boston, USA, June 2010.

[6] J. Dean and L. A. Barroso, "The tail at scale," *Communications of the ACM*, vol. 56, no. 2, pp. 74–80, 2013.

[7] C.-S. Yang, R. Pedarsani, and A. S. Avestimehr, "Timely-throughput optimal coded computing over cloud networks," in *Proceedings of the 20th ACM International Symposium on Mobile Ad Hoc Networking and Computing*, Catania, Italy, July 2019.

[8] K. T. Kim, C. Joe-Wong, and M. Chiang, "Coded edge computing," in *Proceedings of the 2020 IEEE INFOCOM*, Virtual, July 2020.

[9] J. Yue and M. Xiao, "Coding for distributed fog computing in internet of mobile things," *IEEE Transactions on Mobile Computing*, 2020.

[10] M. Zaharia, A. Konwinski, A. D. Joseph, R. Katz, and I. Stoica, "Improving MapReduce Performance in Heterogeneous Environments," in *Proceedings of the 2008 OSDI*, San Diego, CA, December 2008.

[11] J. Dean, "Achieving Rapid Response Times in Large Online Services," 2012. [Online]. Available: https://research.google/pubs/pub44875/

[12] G. Ananthanarayanan, A. Ghodsi, S. Shenker, and I. Stoica, "Effective Straggler Mitigation: Attack of the Clones," in *Proceedings of the 2013 NSDI*. Lombard, IL: USENIX, April 2013.

[13] G. Ananthanarayanan, S. Kandula, A. Greenberg, I. Stoica, Y. Lu, B. Saha, and E. Harris, "Reining in the outliers in map-reduce clusters using Mantri," in *Proceedings of the 2010 OSDI*, BC, Canada, October 2010.

[14] S. Melnik, A. Gubarev, J. J. Long, G. Romer, S. Shivakumar, M. Tolton, and T. Vassilakis, "Dremel: Interactive analysis of web-scale datasets," *Proceedings of the VLDB Endowment*, vol. 3, no. 1, pp. 330–339, 2010.

[15] G. Ananthanarayanan, A. Ghodsi, S. Shenker, and I. Stoica, "Why let resources idle? aggressive cloning of jobs with dolly," in *Proceedings of the 2012 HotCloud*. Boston, MA: USENIX, June 2012.

[16] C. Liu, Q. Wang, X. Chu, Y.-W. Leung, and H. Liu, "Esetstore: An erasure-coded storage system with fast data recovery," *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 9, pp. 2001–2016, 2020.

[17] S. Li, M. A. Maddah-Ali, and A. S. Avestimehr, "Coded MapReduce," in *Proceedings of the 2015 Allerton*, IL, USA, September 2015.

[18] Y. Zhu, P. P. Lee, Y. Xu, Y. Hu, and L. Xiang, "On the speedup of recovery in large-scale erasure-coded storage systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 7, pp. 1830–1840, 2013.

[19] K. Lee, M. Lam, R. Pedarsani, D. Papailiopoulos, and K. Ramchandran, "Speeding up distributed machine learning using codes," in *Proceedings of the 2016 IEEE ISIT*, Barcelona, Spanish, July 2016.

[20] ——, "Speeding Up Distributed Machine Learning Using Codes," *IEEE Transactions on Information Theory*, vol. 64, no. 3, pp. 1514–1529, 2018.

[21] A. Reisizadeh, S. Prakash, R. Pedarsani, and S. Avestimehr, "Coded computation over heterogeneous clusters," in *Proceedings of the 2017 IEEE ISIT*, Aachen, Germany, June 2017.

[22] A. Reisizadeh, S. Prakash, R. Pedarsani, and A. S. Avestimehr, "Coded computation over heterogeneous clusters," *IEEE Transactions on Information Theory*, vol. 65, no. 7, pp. 4227–4242, 2019.

[23] N. S. Ferdinand and S. C. Draper, "Anytime coding for distributed computation," in *Proceedings of the 2016 Allerton*, IL, USA, September 2016.

[24] B. Wang, J. Xie, K. Lu, Y. Wan, and S. Fu, "Coding for Heterogeneous UAV-based Networked Airborne Computing," in *Proceedings of IEEE GLOBECOM Workshop on Computing-Centric Drone Networks*, Hawaii, USA, December 2019.

[25] R. Tandon, Q. Lei, A. G. Dimakis, and N. Karampatziakis, "Gradient Coding: Avoiding Stragglers in Distributed Learning," in *Proceedings of the 34th International Conference on Machine Learning*, 2017.

[26] K. Lee, C. Suh, and K. Ramchandran, "High-dimensional coded matrix multiplication," in *Proceedings of the 2017 IEEE ISIT*, Aachen, Germany, June 2017.

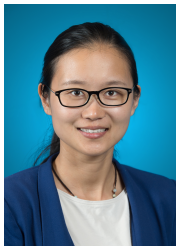
[27] Y. Yang, P. Grover, and S. Kar, "Coded Distributed Computing for Inverse Problems," *Advances in Neural Information Processing Systems*, pp. 710–720, 2017.

[28] K. Lee, R. Pedarsani, D. Papailiopoulos, and K. Ramchandran, "Coded computation for multicore setups," in *Proceedings of the 2017 IEEE ISIT*, Aachen, Germany, June 2017.

[29] S. Dutta, V. Cadambe, and P. Grover, "Short-Dot: Computing Large Linear Transforms Distributedly Using Coded Short Dot Products," *arXiv*, apr 2017. [Online]. Available: https://arxiv.org/abs/1704.05181

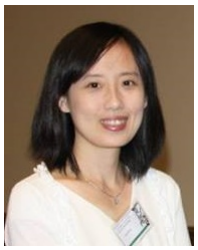
[30] S. Li, M. A. Maddah-Ali, and A. Salman Avestimehr, "A unified coding framework for distributed computing with straggling servers," in *Proceedings of the IEEE Globecom 2016 Workshops*, Washington, DC USA, December 2016.
[31] S. Li, M. A. Maddah-Ali, Q. Yu, and A. S. Avestimehr, "A Fundamental Tradeoff Between Computation and Communication in Distributed Computing," *IEEE Transactions on Information Theory*, vol. 64, no. 1, pp. 109–128, 2018.
[32] M. Kim, J.-Y. Sohn, and J. Moon, "Coded Matrix Multiplication on a Group-Based Model," *arXiv*, jan 2019. [Online]. Available: http://arxiv.org/abs/1901.05162
[33] D. Kim, H. Park, and J. Choi, "Optimal Load Allocation for Coded Distributed Computation in Heterogeneous Clusters," *arXiv*, 2019. [Online]. Available: http://arxiv.org/abs/1904.09496
[34] Y. Keshtkarjahromi, Y. Xing, and H. Seferoglu, "Dynamic heterogeneity-aware coded cooperative computation at the edge," in *Proceedings of the 2018 ICNP*, Athens, Greece, September 2018.
[35] K. G. Narra, Z. Lin, M. Kiamari, S. Avestimehr, and M. Annavaram, "Slack squeeze coded computing for adaptive straggler mitigation," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, Denver, CO, November 2019.
[36] N. Ferdinand and S. C. Draper, "Hierarchical Coded Computation," in *Proceedings of 2018 IEEE ISIT*, Vail, USA, June 2018.
[37] A. Mallick, M. Chaudhari, U. Sheth, G. Palanikumar, and G. Joshi, "Rateless codes for near-perfect load balancing in distributed matrix-vector multiplication," *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, vol. 3, no. 3, pp. 1–40, 2019.
[38] R. M. Corless, G. H. Gonnet, D. E. Hare, D. J. Jeffrey, and D. E. Knuth, "On the lambertw function," *Advances in Computational mathematics*, vol. 5, no. 1, pp. 329–359, 1996.
[39] "Amazon EC2." [Online]. Available: https://aws.amazon.com/ec2/
[40] W. Gropp, E. Lusk, and A. Skjellum, *Using MPI: Portable Parallel Programming with the Message-Passing Interface*. The MIT Press, 1999.
[41] "mpi4py." [Online]. Available: https://mpi4py.readthedocs.io/en/stable/
[42] J.-X. Pan and K.-T. Fang, "Maximum likelihood estimation," in *Growth curve models and statistical diagnostics*. Springer, 2002, pp. 77–158.
[43] D. Sharma, "Estimation of the reciprocal of the scale parameter in a shifted exponential distribution," *Sankhyā: The Indian Journal of Statistics, Series A*, pp. 203–205, 1977.
[44] W. Rudin *et al.*, *Principles of mathematical analysis*. McGraw-hill New York, 1964, vol. 3.
[45] R. Combes, "An extension of McDiarmid's inequality," *arXiv*, 2015. [Online]. Available: http://arxiv.org/abs/1511.05240

Baoqian Wang received his B.S. degree from Yangtze University, Wuhan China, in 2017, and M.S. degree in Computer Science from Texas A&M University-Corpus Christi. He is currently a Ph.D. student in the joint doctoral program of University of California, San Diego and San Diego State University. His research interests include distributed computing, reinforcement learning and robotics.



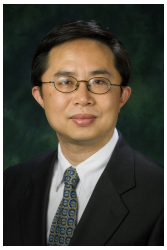
Junfei Xie (S'13-M'16) is currently an assistant professor in the Department of Electrical and Computer Engineering at San Diego State University. She received the B.S. degree in Electrical Engineering from University of Electronic Science and Technology of China (UESTC), Chengdu, China, in 2012. She received the M.S. degree in Electrical Engineering in 2013 and the Ph.D. degree in Computer Science and Engineering from University of North Texas, Denton, TX, in 2016. From 2016 to 2019, she was an Assistant Professor in the Department of Computing Sciences at Texas A&M University-Corpus Christi. She is the recipient of the NSF CAREER Award. Her current research interests include large-scale dynamic system design and control, managing and mining big spatiotemporal data, airborne networks, airborne computing, complex information system, and air traffic flow management, etc.

Kejie Lu (S'01-M'04-SM'07) received the BSc and MSc degrees in Telecommunications Engineering from Beijing University of Posts and Telecommunications, Beijing, China, in 1994 and 1997, respectively. He received the PhD degree in Electrical Engineering from the University of Texas at Dallas in 2003. In 2004 and 2005, he was a Postdoctoral Research Associate in the Department of Electrical and Computer Engineering, University of Florida. In July 2005, he joined University of Puerto Rico at Mayagüez, where he is currently a Professor. His research interests include architecture and protocols design for computer and communication networks, performance analysis, network security, and wireless communications.



Yan Wan (S'08-M'09-SM'17) is currently an Associate Professor in the Electrical Engineering Department at the University of Texas at Arlington. She received her Ph.D. degree in Electrical Engineering from Washington State University in 2008 and then postdoctoral training at the University of California, Santa Barbara. From 2009 to 2016, she was an Assistant Professor and then an Associate Professor at the University of North Texas.

Her research interests lie in the modeling, evaluation and control of large-scale dynamical networks, cyber-physical system, stochastic networks, decentralized control, learning control, networking, uncertainty analysis, algebraic graph theory, and their applications to UAV networking, UAV traffic management, epidemic spread, complex information networks, and air traffic management. Wan's research has led to over 130 publications and successful technology transfer outcomes. She has been recognized by several prestigious awards, including the NSF CAREER Award, RTCA William E. Jackson Award, and the U.S. Ignite and GENI demonstration awards. Wan is currently the Vice President of IEEE Comsoc Fort Worth Chapter and Technical Committee Member of AIAA Intelligent Systems Society. She is also the Associate Editor of the Transactions of the Institute of Measurement and Control.



Shengli Fu (S'01-M'04-SM'07) Shengli Fu received his B.S. and M.S. degrees in telecommunication engineering from Beijing University of Posts and Telecommunications, Beijing, China, in 1994 and 1997, respectively, the M.S. degree in computer engineering from the Wright State University, Dayton, OH, in 2002, and the Ph.D. degree in electrical engineering from the University of Delaware, Newark, DE, in 2005. He is currently a professor and the Chair of the Department of Electrical Engineering, University of North Texas, Denton, TX. His research interests include coding and information theory, wireless communications and sensor networks, and aerial networks.