Decentralized Uncoded Storage Elastic Computing with Heterogeneous Computation Speeds

Wenbo Huang[1], Xudong You[1], Kai Wan[1], Robert Caiming Qiu[1], and Mingyue Ji[2]
[1]Huazhong University of Science and Technology, 430074 Wuhan, China
[2]University of Utah, Salt Lake City, UT 84112, USA
Emails: {eric_huang, xudong_you, kai_wan, caiming}@hust.edu.cn, mingyue.ji@utah.edu

2024 IEEE International Symposium on Information Theory (ISIT) | 979-8-3503-8284-6/24/$31.00 ©2024 IEEE | DOI: 10.1109/ISIT57864.2024.10619573

***Abstract*—Elasticity plays an important role in modern cloud computing systems. Elastic computing allows virtual machines (i.e., computing nodes) to be preempted when high-priority jobs arise, and also allows new virtual machines to participate in the computation. This paper consider the elastic computing with heterogeneous speeds under uncoded storage. In 2018, Yang et al. introduced Coded Storage Elastic Computing (CSEC) to address the elasticity using coding technology, with lower storage and computation load requirements. However, CSEC is limited to certain types of computations (e.g., linear) due to the coded data storage based on linear coding. Then Centralized Uncoded Storage Elastic Computing (CUSEC) with heterogeneous computation speeds was proposed, which directly copies parts of data into the virtual machines. In all existing works in elastic computing, the storage assignment is centralized, meaning that the number and identity of all virtual machines possible used in the whole computation process are known during the storage assignment. In this paper, we consider Decentralized Uncoded Storage Elastic Computing (DUSEC) with heterogeneous computation speeds, where any available virtual machine can join the computation which is not predicted and thus coordination among different virtual machines' storage assignments is not allowed. Under a decentralized storage assignment originally proposed in coded caching by Maddah-Ali and Niesen, we propose a computing scheme with closed-form optimal computation time. We also run experiments over MNIST dataset with Softmax regression model through the Tencent cloud platform, and the experiment results demonstrate that the proposed DUSEC system approaches the state-of-art best storage assignment in the CUSEC system in computation time.**

I. Introduction

Cloud computing platforms provide elastic computation service at discount, while the computations are scheduled on the Virtual Machines (VMs) at a low-priority. It means that at each time step of the computation process (i) VMs will be preempted if a high-priority job arrives; (ii) new available VMs are allowed to join the computation at any time [1]–[3]. To efficiently tolerate the failures brought by preempted VMs, Yang et al [3] introduced Coded Storage Elastic Computing (CSEC) to address the elasticity using coding technology, with lower storage and computation load requirements. Following the original CSEC work, various works on the extensions such as elastic computing with heterogeneous storage or/and speeds, elastic computing against stragglers, optimization on the transition waste, were proposed in [4]–[8]. Despite the advantages of CSEC such as less storage overhead, it may be challenging to be applied to more involved computations (e.g., non-linear task, deep learning) due to the coded data storage. So we may prefer to place the data in an uncoded way by just assigning the raw data to the VMs. [9] proposed a framework for heterogeneous Uncoded Storage Elastic Computing (USEC) in matrix-vector computation and [10] considered a matrix-matrix computation task in uncoded storage systems.

To the best of our knowledge, in all existing works on elastic computing, the storage assignment is centralized, meaning that the number and identity of all virtual machines possibly used in the whole computation process are known in prior during the storage assignment. In this paper we consider Decentralized Uncoded Storage Elastic Computing (DUSEC), where the identity of the VMs participating into the computing process is not in prior known at the beginning of the whole computation process. In other words, any available VM can join the computation and thus coordination among different VMs' storage assignments is not allowed. Unlike the centralized system, there is no limit on the number of available VMs N at time step t in the DUSEC system; as N increases, the computation time decreases. For the computation task, we consider the linearly separable function [11], which is a function of K datasets $(D_1, \ldots, D_K)$ on a finite field $\mathbb{F}_q$. The task function can be seen as $K_c^{(t)}$ linear combinations of K intermediate messages. It was shown in [12] that such function could cover matrix-matrix multiplication, gradient descent, linear transform, etc., as special cases. In addition, we consider that VMs have heterogeneous computation speeds, and aim to minimize the computation time at time t defined as the largest computation time among all available VMs at time step t.

For this new problem, referred to as DUSEC with heterogeneous computation speeds, we consider the case $K_c^{(t)} = 1$, which covers matrix-vector multiplication and gradient descent tasks, and our main contribution is summarized as follows:

1) We use the decentralized storage assignment originally proposed in decentralized coded caching [13], where each VM randomly selects a fraction of datasets to store when it joins in the computation. By assuming the number of datasets is large enough, the storage assignment is symmetric (i.e., the number of datasets inclusively stored by a set of VMs only depends on the cardinality of this set). Considering the heterogeneous computation speeds of the

VMs, we formulate the computation assignment into a linear programming to achieve the minimum computation time at each time step t. We then solve the minimum computation time in closed-form, and propose a new algorithm on assigning the computation assignment while achieving this minimum computation time. Note that the algorithm complexity is $\mathcal{O}(N)$, linear with the number of available VMs at time step t, while the classic algorithm to solve this linear programming has complexity $\mathcal{O}(2^N)$ and cannot provide a closed-form solution.

2) We perform experiments through real cloud platform with heterogeneous computation speeds, and run over the MINST dataset with a Softmax model. We demonstrate that in terms of the total processing time, the proposed algorithms on DUSEC approaches the start-of-the-art CUSEC scheme in [9], which requires the knowledge on the identity of the VMs at the beginning of the process.
3) We then extend the proposed DUSEC scheme by using the distributed gradient coding scheme in [14], such that the resulting elastic computing scheme can also tolerate potential stragglers in the computation process.[1]

Notation Convention: We use $|\cdot|$ to represent the cardinality of a set or the length of a vector and $[n] \triangleq \{1, 2, \ldots, n\}$. A bold symbol such as $\boldsymbol{a}$ indicates a vector and $a[i]$ denotes the i-th element of $\boldsymbol{a}$. Calligraphic symbols such as $\mathcal{A}$ presents a set with numbers as its elements. Bold calligraphic symbols such as $\boldsymbol{\mathcal{A}}$ represents a collection of sets.

II. Problem Formulation

A server uses VMs in a cloud to perform linearly separable computation tasks over multiple time steps. At each time step t, the computation task is a function of K datasets $D_1, \ldots, D_K$, which should be computed collaboratively by N_t available VMs in $\mathcal{N}_t$ with $N_t := |\mathcal{N}_t|$. As in [12], with the assumption that the function is linearly separable from the datasets, the computation task can be written as $K_c^{(t)} \leq K$ linear combinations of K messages,

$$\begin{aligned} &f^{(t)}(D_1, D_2, \ldots, D_K) = g^{(t)}(f_1^{(t)}(D_1), \ldots, f_K^{(t)}(D_K)) \\ &= g^{(t)}(W_1^{(t)}, \ldots, W_K^{(t)}) = \mathbf{F}^{(t)}[W_1^{(t)}; \ldots; W_K^{(t)}], \end{aligned} \tag{1}$$

where the i^{th} message is $W_i^{(t)} = f_i^{(t)}(D_i)$, representing the outcome of the component function $f_i^{(t)}(\cdot)$ applied to dataset D_i, and $\mathbf{F}^{(t)}$ represents the demand information matrix with dimension $K_c^{(t)} \times K$, known by all VMs. Each message $W_i^{(t)}$ contains L uniformly i.i.d. symbols on some finite field $\mathbb{F}_q$.[2]

In this paper, we mainly consider the case $K_c^{(t)} = 1$, which covers matrix-vector multiplication and gradient descent tasks (by letting $f_i^{(t)}(\cdot)$ be a gradient of loss function). In this case, we assume without loss of generality that the computation task at time step t is $W_1^{(t)} + \cdots + W_K^{(t)}$.

[1]The difference between elasticity and straggler is that, for elasticity at the beginning of each step time we know the identity of the computing nodes who has joined in or left; but we do not know which nodes will be stragglers.

[2]The proposed scheme can also work in the field of real numbers.

A. Decentralized Storage Assignment and Heterogeneous Computation speeds

We consider a decentralized system, where any VM may join the computation process unpredictably and thus coordination on storage assignment for different VMs is not allowed. We use the decentralized storage assignment in [15], where each VM stores a subset of the datasets independently at random. Assume that each VM n stores $\{D_i : i \in \mathcal{Z}_n\}$, with a equal storage size $|\mathcal{Z}_n| = M$. So each dataset is stored by each VM with probability $\frac{M}{K}$.

Since there is no coordination among VMs' storage assignment and the computation tasks at different time steps are independent, in the rest of this paper we only focus on one time step t; and to avoid heavy notations, we do not explicitly point out the time step index t in the notations. For example, we drop the superscript from $W_i^{(t)}$ and the i^{th} message becomes W_i. We further assume that the available VMs at the considered time step is $\mathcal{N} = [N]$.

According to the storage of the VMs in $[N]$, we can divide the K datasets into 2^N sets, where $\mathcal{A}_\mathcal{V}$ represents the sets of datasets assigned to all VMs in $\mathcal{V}$, $\{V_j : j \in \mathcal{V}\}$, for each $\mathcal{V} \subseteq [N]$. By assuming that K is large enough and then by the law of large numbers, we have

$$|\mathcal{A}_\mathcal{V}| = \left(\frac{M}{K}\right)^{|\mathcal{V}|} \left(1 - \frac{M}{K}\right)^{N-|\mathcal{V}|} K + o(K), \tag{2}$$

with probability approaching one when K is sufficiently large.

Note that by the decentralized storage assignment, there remain some datasets not stored by any VMs in $[N]$; thus we can only compute an approximated version of the original computation task. So we approximate the computation task to $\boldsymbol{y} = \sum_{i \in \bigcup_{j \in [N]} \mathcal{Z}_j} W_i$.

Without loss of generality, we assume that the computation speeds are in an ascending order as $s[1] \leq s[2] \leq \cdots \leq s[N]$. Denote $\boldsymbol{s} = (s[1], \ldots, s[N])$.

Next, we define $\mathcal{L}_\mathcal{V}$ as the set of datasets which are only assigned to $\{V_j, j \in \mathcal{V}\}$; thus

$$\mathcal{L}_\mathcal{V} = \bigcup_{r \subset \mathcal{V}} \mathcal{A}_r. \tag{3}$$

Specially, $\mathcal{L}_{[n]}$ can be written as

$$\mathcal{L}_{[n]} = \mathcal{A}_{\{1\}} \cup \cdots \cup \mathcal{A}_{\{n\}} \cup \mathcal{A}_{\{1,2\}} \cup \cdots \cup \mathcal{A}_{\{1,2,\ldots,n\}}. \tag{4}$$

By (2), we have with high probability that

$$|\mathcal{L}_{[n]}| = \left(\frac{K-M}{K}\right)^N \left\{\left(\frac{K}{K-M}\right)^n - 1\right\} K + o(K). \tag{5}$$

To simply the notation, define $\alpha \triangleq \frac{K}{K-M}$ and $\beta \triangleq \left(\frac{K-M}{K}\right)^N$. Then we have

$$|\mathcal{A}_\mathcal{V}|/K = \beta\left(\alpha - 1\right)^{|\mathcal{V}|}, \ |\mathcal{L}_{[n]}|/K = \beta\left(\alpha^n - 1\right). \tag{6}$$

B. USEC under Decentralized Assignment

Each VM V_n where $n \in [N]$ computes $\sum_{\mathcal{V}\subseteq[N]:n\in\mathcal{V}} \sum_{i\in\mathcal{S}_{n,\mathcal{V}}} W_i$, where $\mathcal{S}_{n,\mathcal{V}} \subseteq \mathcal{A}_{\mathcal{V}}$ denotes the set of datasets in $\mathcal{A}_{\mathcal{V}}$ which should be computed by VM V_n. So the computation assignment for all VMs can be exactly determined by the collection, we have $\mathcal{S}_n \triangleq \bigcup_{\mathcal{V}\subset[N]:n\in\mathcal{V}} \mathcal{S}_{n,\mathcal{V}}$ and $\boldsymbol{M} = \{\mathcal{S}_n, \forall n \in [N]\}$. Then the computed results from VMs are transmitted to the server to recover $\boldsymbol{y}$.

Computation Load. We define $\mu[n]$, the computation load by each VM V_n, as the number of computed messages normalized by K; thus we have

$$\mu[n] = \sum_{\mathcal{V}\subseteq[N]:n\in\mathcal{V}} \frac{|\mathcal{S}_{n,\mathcal{V}}|}{K} = \sum_{\mathcal{V}\subseteq[N]:n\in\mathcal{V}} \mu[n,\mathcal{V}], \tag{7}$$

where we define $\mu[n,\mathcal{V}] = |\mathcal{S}_{n,\mathcal{V}}|/K$. Then the computation load vector for the VMs in $[N]$ is, $\boldsymbol{\mu} = (\mu[1], \cdots, \mu[n])$. Note that since the computation load is normalized by K, thus we can neglect the deviation terms and assume that

Computation Time. The computation time of each VM V_n where $n \in [N]$ is $\frac{\mu[n]}{s[n]}$. The overall computation time at the considered time step (or simply called computation time) is defined as largest computation time among all VMs in $[N]$,

$$c(\boldsymbol{M}) \triangleq \max_{n\in[N]} \frac{\mu[n]}{s[n]} = \max_{n\in[N]} \frac{\sum_{\mathcal{V}\subseteq[N]:n\in\mathcal{V}} \mu[n,\mathcal{V}]}{s[n]}. \tag{8}$$

For a fixed storage assignment $(\mathcal{Z}_1, \ldots, \mathcal{Z}_N)$, we can formulate the following optimization problem for the DUSEC system with heterogeneous computation speeds:

$$\underset{\boldsymbol{M}}{\text{minimize}} \quad c(\boldsymbol{M}) \tag{9a}$$

$$\text{subject to:} \quad \bigcup_{n\in\mathcal{V}} \mathcal{S}_{n,\mathcal{V}} = \mathcal{A}_{\mathcal{V}}, \ \forall \mathcal{V} \subseteq \mathcal{N}. \tag{9b}$$

The optimization problem in (9) is equivalent to the following linear programming:

$$\underset{\{\mu[n,\mathcal{V}]:n\in[N],\mathcal{V}\subseteq[N]\}}{\text{minimize}} \quad c(\boldsymbol{M}) = \max_{n\in[N]} \frac{\sum_{\mathcal{V}\subseteq[N]:n\in\mathcal{V}} \mu[n,\mathcal{V}]}{s[n]} \tag{10a}$$

$$\text{subject to:} \sum_{n\in\mathcal{V}} \mu[n,\mathcal{V}] = \beta(\alpha-1)^{|\mathcal{V}|}, \ \forall \mathcal{V} \subseteq [N], \tag{10b}$$

$$\mu[n,\mathcal{V}] \geq 0, \forall n \in [N], \mathcal{V} \subseteq [N]. \tag{10c}$$

III. Main Results

Instead of directly solving the linear programming in (10) by Lagrange multipliers or some other standard methods (which is normally hard to get a closed-form solution), we solve (10) by first proving a cut-set converse bound on the computation time and then proposing an algorithm which matches the converse bound. Thus we can obtain the optimal solution in closed-form for the problem in (10).

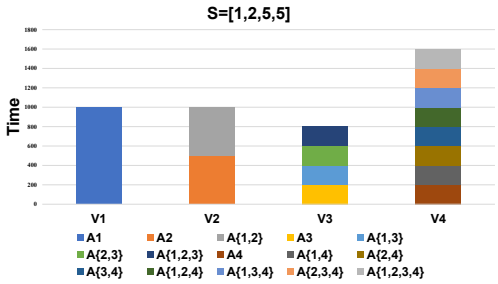

Fig. 1: Illustration of DUSEC when $\mathbf{s} = [1, 2, 5, 5]$ before rearrangement, $c(\boldsymbol{M}) = 0.1$

Theorem 1 (Optimal Computation Time): Under the DUSEC system with heterogeneous computation speeds, the optimal computation time is

$$c^\star = \min_{\boldsymbol{M}} c(\boldsymbol{M}) = \max_{n^*} \frac{|\mathcal{L}_{[n^*]}|/K}{\sum_{i=1}^{n^*} s[i]}, \tag{11a}$$

$$\text{where} \quad \frac{|\mathcal{L}_{[n^*]}|/K}{\sum_{i=1}^{n^*} s[i]} \geq \frac{|\mathcal{L}_{[n]} \setminus \mathcal{L}_{[n^*]}|/K}{\sum_{i=n^*+1}^{n} s[i]}, \ \forall n \in [n^*+1:N], \tag{10b}$$

$$\text{and} \quad \frac{|\mathcal{L}_{[n^*]}|/K}{\sum_{i=1}^{n^*} s[i]} \geq \frac{|\mathcal{L}_{[n]}|/K}{\sum_{i=1}^{n} s[i]}, \forall n \in [n^*-1]. \tag{10c}$$

Lemma 1: There exists a n^* that satisfies the Condition (10b) and Condition (10c) in Theorem 1.

The proof of Lemma 1 and the converse bound in Theorem 1 are given in the extended version of this paper [16].

We provide an example to illustrate the main idea of the proposed scheme which achieves the optimal computation time in Theorem 1 and the general description is shown in [16] .

Example 1: We consider a system with parameters $K = 16000, N = 4, \alpha = 2, \mathbf{s} = [1, 2, 5, 5]$. There are $K = 16000$ datasets, and each VM can store $M = 8000$ datasets. By the law of the large number, we have $|\mathcal{L}_{[1]}|/K = 0.0625, |\mathcal{L}_{[2]}|/K = 0.1875, |\mathcal{L}_{[3]}|/K = 0.4375, |\mathcal{L}_{[4]}|/K = 0.9375$. From Theorem 1, we can get the $n^\star = 4$ and $c^\star = 0.0721$.

There are 4 iterations to get the computation load assignment $\mathcal{S} = \{\mathcal{S}_1, \mathcal{S}_2, \mathcal{S}_3, \mathcal{S}_4\}$. In the n-th iteration, we determine the computation load for $V_i, i \in [n]$ with datasets in $\mathcal{L}_{[n]}$, and t_i is denoted as the computation time of V_i and $t_{\mathcal{V}} = t_i, i \in \mathcal{V}$ (The above parameters will be updated after each iteration).

In the first iteration, VM V_1 computes $\mathcal{L}_{\{1\}} = \mathcal{A}_{\{1\}}$ within $t_1 = 0.0625$. We update $\mathcal{S}_1 = \mathcal{A}_{\{1\}}$.

In the second iteration, by letting $\mathcal{S}_2 = \mathcal{L}_{[2]} \setminus \mathcal{L}_{[1]}$, we have $t_2 = \frac{|\mathcal{L}_{[2]} \setminus \mathcal{L}_{[1]}|/K}{s[2]} = 0.0625$ which is equal to t_1. Then update $\mathcal{S}_1 = \mathcal{A}_{\{1\}}, \mathcal{S}_2 = \mathcal{A}_{\{2\}} \cup \mathcal{A}_{\{1,2\}}$.

In the third iteration, by letting $\mathcal{S}_3 = \mathcal{L}_{[3]} \setminus \mathcal{L}_{[2]}$, we have $t_3 = \frac{|\mathcal{L}_{[3]} \setminus \mathcal{L}_{[2]}|/K}{s[3]} = 0.05$, where $t_3 < t_2 = t_1$ satisfies Condition (10b) in Theorem 1. Then update $\mathcal{S}_1 = \mathcal{A}_{\{1\}}, \mathcal{S}_2 = \mathcal{A}_{\{2\}} \cup \mathcal{A}_{\{1,2\}}, \mathcal{S}_3 = \mathcal{A}_{\{3\}} \cup \mathcal{A}_{\{1,3\}} \cup \mathcal{A}_{\{2,3\}} \cup \mathcal{A}_{\{1,2,3\}}$.

In the fourth iteration, if we let that $\mathcal{S}_4 = \mathcal{L}_{[4]} \setminus \mathcal{L}_{[3]}$, then $t_4 = \frac{|\mathcal{L}_{[4]} \setminus \mathcal{L}_{[3]}|/K}{s[4]} = 0.1 > c^\star > t_3$, which contradicts with

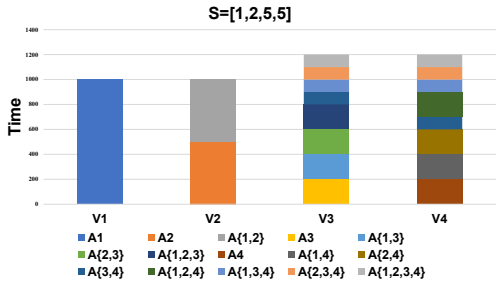

Fig. 2: Illustration of DUSEC after 4 iterations after rearranging $\mathcal{S}_3$ and $\mathcal{S}_4$, when $\mathbf{s} = [1, 2, 5, 5]$, $c(\boldsymbol{M}) = 0.075$

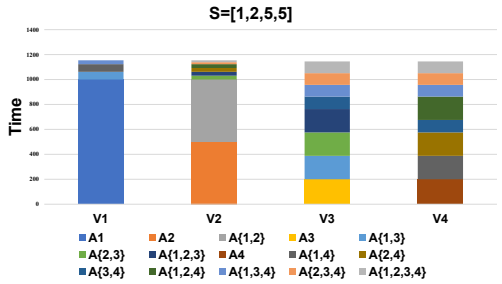

Fig. 3: Illustration of DUSEC when $\mathbf{s} = [1, 2, 5, 5]$ after all rearrangement, $c^\star = c(\boldsymbol{M}) = 0.0721$.

Condition (10b) in Theorem 1.

To reduce $c(\boldsymbol{M}) = t_4$, parts of datasets both stored by VMs V_3 and V_4 should be computed by VM V_3. With $t_3 = 0.05$ and $t_4 = 0.1$, we compute $t_{\{3,4\}} = \frac{|\mathcal{L}_{[4]} \setminus \mathcal{L}_{[2]}|/K}{s[3]+s[4]} = 0.075$, and rearrange $\mathcal{S}_3$ and $\mathcal{S}_4$. VM V_3 needs to compute more datasets which come from $\mathcal{A}_{\{3,4\}} \cup \mathcal{A}_{\{1,3,4\}} \cup \mathcal{A}_{\{2,3,4\}} \cup \mathcal{A}_{\{1,2,3,4\}}$ with size $\delta = \left(t_{\{3,4\}} - t_3\right) s[3] = 0.125$.

The main non-trivial step is how to select this 0.125 computation load. The original $\mathcal{S}_3$ is equal to $\mathcal{A}_{\{3\}} \cup \mathcal{A}_{\{1,3\}} \cup \mathcal{A}_{\{2,3\}} \cup \mathcal{A}_{\{1,2,3\}}$, which consists of $\mathcal{A}_{\{3\}\cup\alpha}, \alpha \subset [2]$, the original $\mathcal{S}_4$ consists of $\mathcal{A}_{\{4\}\cup\alpha}, \alpha \subset [3]$, and the overlap consists of $\mathcal{A}_{\{3,4\}\cup\alpha}, \alpha \subset [2]$. From the size of rearranged computation load $\delta = 0.125$ and $\mu[3] = 0.25$, $\mu[4] = 0.5$, we can update $\mathcal{S}_3$ into[3]

$$\mathcal{S}_3 = \mathcal{A}_{\{3\}} \cup \mathcal{A}_{\{1,3\}} \cup \mathcal{A}_{\{2,3\}} \cup \mathcal{A}_{\{1,2,3\}} \cup 0.5\mathcal{A}_{\{3,4\}} \cup 0.5\mathcal{A}_{\{1,3,4\}} \cup 0.5\mathcal{A}_{\{2,3,4\}} \cup 0.5\mathcal{A}_{\{1,2,3,4\}},$$

and update $\mathcal{S}_4$ into

$$\mathcal{S}_4 = \mathcal{A}_{\{4\}} \cup \mathcal{A}_{\{1,4\}} \cup \mathcal{A}_{\{2,4\}} \cup \mathcal{A}_{\{1,2,4\}} \cup 0.5\mathcal{A}_{\{3,4\}} \cup 0.5\mathcal{A}_{\{1,3,4\}} \cup 0.5\mathcal{A}_{\{2,3,4\}} \cup 0.5\mathcal{A}_{\{1,2,3,4\}},$$

where there is no overlap between $\mathcal{S}_3$ and $\mathcal{S}_4$. As illustrated in Fig. 2, after the computation load rearrangement, $c(\boldsymbol{M}) = t_3 = t_4 = 0.075 > t_1$ while the maximum time still contradicts with Condition (10b) in Theorem 1.

We further rearrange $\{\mathcal{S}_1, \mathcal{S}_2, \mathcal{S}_3, \mathcal{S}_4\}$ With $t_{\{1,2\}} = 0.0625, t_{\{3,4\}} = 0.075, c^\star = t_{\{1,2,3,4\}} = 0.0721, \delta = 0.0288$, the overlap of $\mathcal{L}_{[2]}$ and $\mathcal{L}_{[4]} \setminus \mathcal{L}_{[2]}$ is

$$\mathcal{A}_{\{1,3\}} \cup \mathcal{A}_{\{1,4\}} \cup \mathcal{A}_{\{1,3,4\}} \cup \mathcal{A}_{\{2,3\}} \cup \mathcal{A}_{\{2,4\}} \cup \mathcal{A}_{\{2,3,4\}} \cup \mathcal{A}_{\{1,2,3\}} \cup \mathcal{A}_{\{1,2,4\}} \cup \mathcal{A}_{\{1,2,3,4\}},$$

which consists of $\mathcal{A}_{\{1\}\cup\alpha}, \mathcal{A}_{\{2\}\cup\alpha}, \mathcal{A}_{\{1,2\}\cup\alpha}, \alpha \subsetneqq \{3,4\}$, or $\mathcal{A}_{\{3\}\cup\alpha}, \mathcal{A}_{\{4\}\cup\alpha}, \mathcal{A}_{\{3,4\}\cup\alpha}, \alpha \subsetneqq [2]$.

Notice that the original $\mathcal{S}_1, \mathcal{S}_2$ consists of $\mathcal{A}_{\{1\}}, \mathcal{A}_{\{2\}}, \mathcal{A}_{\{1,2\}}$ and the original $\mathcal{S}_3, \mathcal{S}_4$ consists of $\mathcal{A}_{\{3\}\cup\alpha}, \mathcal{A}_{\{4\}\cup\alpha}, \mathcal{A}_{\{3,4\}\cup\alpha}, \forall \alpha \subset [2]$ correspondingly. Denote the rearranged $\mathcal{S}_{n,\mathcal{V}\cup\mathcal{Q}}, n \in \mathcal{V}, \mathcal{V} \subsetneqq [2], \mathcal{Q} \subsetneqq \{3,4\}$ as $\mathcal{S}'_{n,\mathcal{V}\cup\mathcal{Q}}$, with the constraint that the summation of $|\mathcal{S}'_{n,\mathcal{V}\cup\mathcal{Q}}|, n \in \mathcal{V}, \forall \mathcal{V} \subsetneqq [2], \mathcal{Q} \subsetneqq \{3,4\}$ is equal to δ, where $|\mathcal{S}'_{n,\mathcal{V}\cup\mathcal{Q}}|$ is the size of $\mathcal{S}'_{n,\mathcal{V}\cup\mathcal{Q}}$. Our proposed scheme let $|\mathcal{S}'_{n,\mathcal{V}\cup\mathcal{Q}}|$ be proportional to $\frac{|\mathcal{S}_{n,\mathcal{V}}|}{|\mathcal{L}_{[2]}|}$ and $\frac{|\mathcal{A}_\mathcal{Q}|}{|\mathcal{L}_{\{3,4\}}|}$, while the summation of $|\mathcal{S}'_{n,\mathcal{V}\cup\mathcal{Q}}|, n \in \mathcal{V}, \forall \mathcal{V} \subsetneqq [2], \mathcal{Q} \subsetneqq \{3,4\}$ is equal to δ. As a result, it can be designed as $\mathcal{S}'_{n,\mathcal{V}\cup\mathcal{Q}} = \frac{\delta|\mathcal{A}_\mathcal{Q}||\mathcal{S}_{n,\mathcal{V}}|}{|L_{[2]}||L_{\{3,4\}}||\mathcal{A}_{\mathcal{V}\cup\mathcal{Q}}|}\mathcal{A}_{\mathcal{V}\cup\mathcal{Q}}$. When $n \in \mathcal{Q}, \mathcal{Q} \subsetneqq \{3,4\}, \mathcal{V} \subsetneqq [2]$, there is $\mathcal{S}'_{n,\mathcal{Q}\cup\mathcal{V}} = \frac{\delta|\mathcal{A}_\mathcal{V}||\mathcal{S}_{n,\mathcal{Q}}|}{|L_{[2]}||L_{\{3,4\}}|\mathcal{A}_{\mathcal{Q}\cup\mathcal{V}}|}\mathcal{A}_{\mathcal{Q}\cup\mathcal{V}}$. The computation load rearrangement of $\mathcal{S}_n$ for each $n \in [2]$ is

$$\mathcal{S}'_{n,\mathcal{V}\cup\mathcal{Q}} = \frac{\delta|\mathcal{A}_\mathcal{Q}||\mathcal{S}_{n,\mathcal{V}}|}{|L_{[2]}||L_{\{3,4\}}||\mathcal{A}_{\mathcal{V}\cup\mathcal{Q}}|}\mathcal{A}_{\mathcal{V}\cup\mathcal{Q}}, \quad \forall \mathcal{V} \subsetneqq [2], n \in \mathcal{V}, \mathcal{Q} \subsetneqq \{3,4\},$$

and the new computation load of $\mathcal{S}_n$ for each $n \in \{3,4\}$ is

$$\mathcal{S}_{n,\mathcal{Q}\cup\mathcal{V}} \setminus \mathcal{S}'_{n,\mathcal{Q}\bigcup\mathcal{V}} = \mathcal{S}_{n,\mathcal{Q}\cup\mathcal{V}} \setminus \frac{\delta|\mathcal{A}_\mathcal{V}||\mathcal{S}_{n,\mathcal{Q}}|}{|L_{\{3,4\}}||L_{[2]}||\mathcal{A}_{\mathcal{Q}\cup\mathcal{V}}|}\mathcal{A}_{\mathcal{Q}\cup\mathcal{V}}, \quad \forall \mathcal{Q} \subsetneqq \{3,4\}, n \in \mathcal{Q}, \mathcal{V} \subsetneqq [2].$$

For example, when $n = 1, \mathcal{V} = \{1\}, \mathcal{Q} = \{3\}$, there is $\mathcal{S}'_{1,\{1,3\}} = \frac{\delta|\mathcal{A}_{\{3\}}||\mathcal{S}_{1,\{1\}}|}{|L_{[2]}||L_{\{3,4\}}||\mathcal{A}_{\{1,3\}}|}\mathcal{A}_{\{1,3\}} = 0.0513\mathcal{A}_{\{1,3\}}$ and when $n = 3, \mathcal{Q} = \{3,4\}, \mathcal{V} = \{1\}$, there is $\mathcal{S}_{3,\{1,3,4\}} \setminus \frac{\delta|\mathcal{A}_{\{1\}}||\mathcal{S}_{3,\{3,4\}}|}{|L_{[2]}||L_{\{3,4\}}||\mathcal{A}_{\{1,3,4\}}|}\mathcal{A}_{\{1,3,4\}} = 0.4744\mathcal{A}_{\{1,3,4\}}$.

From the proposed scheme above, we get $c^\star = c(\boldsymbol{M}) = t_1 = t_2 = t_3 = t_4 = 0.0721$, which is shown in Fig. 3, and the optimal computation load assignment $\{\mathcal{S}_1, \mathcal{S}_2, \mathcal{S}_3, \mathcal{S}_4\}$ is presented in TABLE I.

□

IV. Evaluations on Tencent Cloud

We evaluate the proposed algorithm using Softmax Regression on Tencent cloud platform. The goal is to compare the performance difference in terms of accuracy and computation time under DUSEC and CUSEC system.

Softmax Regression: Softmax regression is a model designed for solving multi-class classification problems. In Softmax regression, the model starts with a linear transformation of the input feature vector x using the weight matrix W and bias vector b to compute scores for each class, and we train the model by updating W and b by $W := W - \alpha\nabla_W J(W, b)$, where $J(\cdot)$ is the loss function.

[3]With a slight abuse of notation, $\alpha\mathcal{A}_\mathcal{V}$ represents an α fraction of datasets in $\mathcal{A}_\mathcal{V}$.

TABLE I: The computation load assignment for $s = [1, 2, 5, 5]$ system

	$\mathcal{S}_1$	$\mathcal{S}_2$	$\mathcal{S}_3$	$\mathcal{S}_4$
$\mathcal{A}_{\{1\}}$	1	0	0	0
$\mathcal{A}_{\{2\}}$	0	1	0	0
$\mathcal{A}_{\{3\}}$	0	0	1	0
$\mathcal{A}_{\{4\}}$	0	0	0	1
$\mathcal{A}_{\{1,2\}}$	0	1	0	0
$\mathcal{A}_{\{1,3\}}$	0.0513	0	0.9487	0
$\mathcal{A}_{\{1,4\}}$	0.0513	0	0	0.9487
$\mathcal{A}_{\{2,3\}}$	0	0.0513	0.9487	0
$\mathcal{A}_{\{2,4\}}$	0	0.0513	0	0.9487
$\mathcal{A}_{\{3,4\}}$	0	0	0.5	0.5
$\mathcal{A}_{\{1,2,3\}}$	0	0.0513	0.9487	0
$\mathcal{A}_{\{1,2,4\}}$	0	0.0513	0	0.9487
$\mathcal{A}_{\{1,3,4\}}$	0.0513	0	0.4744	0.4744
$\mathcal{A}_{\{2,3,4\}}$	0	0.0513	0.4744	0.4744
$\mathcal{A}_{\{1,2,3,4\}}$	0	0.0513	0.4744	0.4744

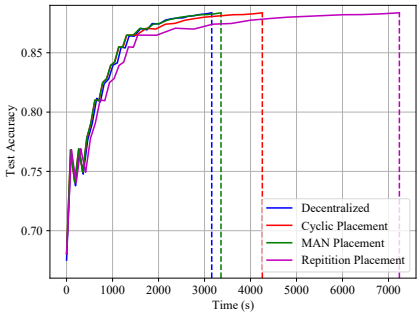

Fig. 4: Results using DUSEC and CUSEC designs on Tencent cloud platform. The y-axis represents the normalized mean square error between the true dominant eigenvector and the estimated eigenvector.

The network has one `S5.2XLARGE16` master machine with 8 vCPUs and 16 GiB of memory. The worker VMs consist of 2 `S5.LARGE8` instances, each with 4 vCPUs and 8 GiB of memory, and 2 `S5.2XLARGE16` instances, each with 8 vCPUs and 16 GiB of memory. We specified the number of cores that each worker node participates in computation to simulate different speeds, normalized as $\{s[1] = 1, s[2] = 2, s[3] = 5, s[4] = 5\}$. We consider the case when $K = 60000$ during the experiments.

In our experiment, we conduct a comparison between DUSEC and CUSEC considering the presence of preemption. For the CUSEC system, we opt for repetition, cyclic, MAN [17] assignment. The results demonstrate that the proposed DUSEC system approaches the state-of-art assignment in the CUSEC system in computation time (refer to Fig. 4). The DUSEC performs even better because of the less computation load under the decentralized assignment.

V. Extension to Straggler Mitigation

In this section, we encode the transmission by each VM of the proposed scheme in Theorem 1 in order to mitigate up to s unpredictable stragglers in the computation process.

Note that without elasticity, coded distributed computing against stragglers was well studied in the literature [14], [18]–[20] to compute the linearly separable function described in Section II with $K_c = 1$. In the proposed schemes [14], [18]–[20], various coding techniques were used to encode the messages computed by each VM n (i.e, the messages W_j where $j \in \mathcal{Z}_n$), who then sends the coded messages to the server. As a result, after receiving the transmissions by a fix number of VMs, the server can recover the computation task, while the communication cost is reduced compared to the uncoded transmission. In particular, a unified coding scheme was proposed in [14] which works for any computation assignment $\boldsymbol{M}$ if each message is computed by at least $s+1$ VMs, in order to tolerate s stragglers.

Next we provide an example to illustrate how to combine the proposed elastic scheme with the scheme in [14], in order to tolerate s straggler, while the general description could be found in [16] .

Example 2 ($N = 3$, $s = 1$): We consider the system, where the system should tolerate up to $s = 1$ straggler. For the simplicity, we assume that the computation speeds of the three VMs are $s[1] \ll s[2] = s[3]$. Note that the identity of the straggler is not known before the transmission. Hence, each dateset which is assigned to only one VM in [3] will not be considered into the transmission. In other words, we only consider the datesets in $\mathcal{A}_{\mathcal{V}}$ where $\mathcal{V} \subseteq [3]$ and $|\mathcal{V}| > 1$. Denote $W_{\mathcal{V}}$ as the sum of the messages in $\mathcal{A}_{\mathcal{V}}$. For each $\mathcal{V} \subseteq [3]$ and $|\mathcal{V}| = 2$, to tolerate 1 straggler, $W_{\mathcal{V}}$ should be completely computed by the VMs in $\mathcal{V}$. For $\mathcal{V} \subseteq [3]$ and $|\mathcal{V}| = 3$, i.e., $\mathcal{V} = \{1, 2, 3\}$, we should determine the computation assignment by solving an optimization problem (see the optimization problem in [16]). Different from the orignal optimization problem in (10), the main difference of the optimization problem is that each dataset should be computed totally $s + 1$ times, instead of 1. In this example, since $s[1] \ll s[2] = s[3]$, we let $W_{\{1,2,3\}}$ completely computed by the workers in $\{2, 3\}$.

Then based on the computation assignment, we apply the coding technique in [14], to let VMs V_1, V_2, V_3 transmit

$$T_1 = \frac{1}{2}W_{\{1,2\}} + W_{\{1,3\}},$$
$$T_2 = \frac{1}{2}W_{\{1,2\}} + W_{\{2,3\}} + W_{\{1,2,3\}},$$
$$T_3 = W_{\{1,3\}} - W_{\{2,3\}} - W_{\{1,2,3\}},$$

respectively. It can be seen that, from any 2 coded messages of T_1, T_2, T_3, the server can recover $W_{\{1,2\}} + W_{\{1,3\}} + W_{\{2,3\}} + W_{\{1,2,3\}}$. □

Acknowledgement: The work of W. Huang, X. You, K. Wan and R. C. Qiu was partially funded by NSFC-12141107. The work of M. Ji was partially funded by NSF Award 2312227 and CAREER Award 2145835.

REFERENCES

[1] X. Meng, J. Bradley, B. Yavuz, E. Sparks, S. Venkataraman, D. Liu, J. Freeman, D. Tsai, M. Amde, S. Owen *et al.*, "Mllib: Machine learning in apache spark," *The journal of machine learning research*, vol. 17, no. 1, pp. 1235–1241, 2016.

[2] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin *et al.*, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," *arXiv preprint arXiv:1603.04467*, 2016.

[3] Y. Yang, M. Interlandi, P. Grover, S. Kar, S. Amizadeh, and M. Weimer, "Coded elastic computing," in *2019 IEEE International Symposium on Information Theory (ISIT)*, July 2019, pp. 2654–2658.

[4] S. Kiani, T. Adikari, and S. C. Draper, "Hierarchical coded elastic computing," in *ICASSP 2021 - 2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2021, pp. 4045–4049.

[5] N. Woolsey, R.-R. Chen, and M. Ji, "Coded elastic computing on machines with heterogeneous storage and computation speed," *IEEE Transactions on Communications*, vol. 69, no. 5, pp. 2894–2908, 2021.

[6] N. Woolsey, J. Kliewer, R.-R. Chen, and M. Ji, "A practical algorithm design and evaluation for heterogeneous elastic computing with stragglers," *arXiv preprint arXiv:*, 2021.

[7] X. Zhong, J. Kliewer, and M. Ji, "Matrix multiplication with straggler tolerance in coded elastic computing via lagrange code," in *in 2023 IEEE International Conference on Communications (ICC)*, 2023, pp. 136–141.

[8] H. Dau, R. Gabrys, Y. C. Huang, C. Feng, Q. H. Luu, E. Alzahrani, and Z. Tari, "Optimizing the transition waste in coded elastic computing," in *2020 IEEE International Symposium on Information Theory (ISIT)*. IEEE, 2020, pp. 174–178.

[9] M. Ji, X. Zhang, and K. Wan, "A new design framework for heterogeneous uncoded storage elastic computing," in *2022 20th International Symposium on Modeling and Optimization in Mobile, Ad hoc, and Wireless Networks (WiOpt)*, 2022, pp. 269–275.

[10] X. Zhong, J. Kliewer, and M. Ji, "Uncoded storage coded transmission elastic computing with straggler tolerance in heterogeneous systems," *arXiv:2401.12151*, Jan. 2024.

[11] D. Mosk-Aoyama and D. Shah, "Fast distributed algorithms for computing separable functions," *IEEE Transactions on Information Theory*, vol. 54, no. 7, pp. 2997–3007, 2008.

[12] K. Wan, H. Sun, M. Ji, and G. Caire, "Distributed linearly separable computation," *IEEE Transactions on Information Theory*, vol. 68, no. 2, pp. 1259–1278, 2021.

[13] M. A. Maddah-Ali and U. Niesen, "Decentralized coded caching attains order-optimal memory-rate tradeoff," *Networking, IEEE/ACM Transactions on*, vol. 23, no. 4, pp. 1029–1040, Aug 2015.

[14] T. Jahani-Nezhad and M. A. Maddah-Ali, "Optimal communication-computation trade-off in heterogeneous gradient coding," *IEEE Journal on Selected Areas in Information Theory*, vol. 2, no. 3, pp. 1002–1011, 2021.

[15] M. A. Maddah-Ali and U. Niesen, "Decentralized coded caching attains order-optimal memory-rate tradeoff," *IEEE/ACM Transactions on Networking*, vol. 23, no. 4, pp. 1029–1040, 2015.

[16] W. Huang, X. You, K. Wan, R. C. Qiu, and M. Ji, "Decentralized uncoded storage elastic computing with heterogeneous computation speeds," *arXiv preprint arXiv:2403.00585*, 2024.

[17] M. A. Maddah-Ali and U. Niesen, "Fundamental limits of caching," *IEEE Transactions on Information Theory*, vol. 60, no. 5, pp. 2856–2867, 2014.

[18] R. Tandon, Q. Lei, A. G. Dimakis, and N. Karampatziakis, "Gradient coding: Avoiding stragglers in distributed learning," in *International Conference on Machine Learning*. PMLR, 2017, pp. 3368–3376.

[19] M. Ye and E. Abbe, "Communication-computation efficient gradient coding," in *International Conference on Machine Learning*. PMLR, 2018, pp. 5610–5619.

[20] H. Cao, Q. Yan, and X. Tang, "Adaptive gradient coding," *IEEE/ACM Trans. Networking*, vol. 30, no. 2, pp. 717–734, Apr. 2022.