

2) heterogeneous computing speeds and heterogeneous storage capacity. The proposed combinatorial optimization problem is solved in two steps in Section IV and Section V, where the proposed low-complexity CEC computation assignment algorithms are introduced. Section VI summarizes the results and compares the performance across multiple designs. The paper is concluded in Section VII.

Notation Convention: We use $|\cdot|$ to denote the cardinality of a set or the length of a vector. Let $[n] := \{1, 2, \dots, n\}$ denote a set of integers from 1 to n . A bold symbol such as \mathbf{a} indicates a vector and $a[i]$ denotes the i -th element of \mathbf{a} . \mathbb{Z}^+ denotes the set of all positive integers; \mathbb{R}^+ denotes the set of all positive real numbers and \mathbb{Q}^+ denotes the set of all positive rational numbers. Finally, let \mathbb{R}_+^N be the set of all length- N vectors of real, positive numbers.

II. SYSTEM MODEL AND PROBLEM FORMULATION

A. System Model

We consider a set of N machines. Each machine $n \in [N]$ stores an integer number, $\sigma[n]$, of coded sub-matrices, which we refer to as cs-matrices, derived from a $q \times r$ data matrix, \mathbf{X} , where q can be large. Here, we denote the vector $\boldsymbol{\sigma} = (\sigma[1], \sigma[2], \dots, \sigma[N])$, $\sigma[n] \in \mathbb{Z}^+$, $\forall n \in [N]$ as the storage vector and $Z \triangleq \sum_{n=1}^N \sigma[n]$ is the total number of cs-matrices stored among all machines.¹ The cs-matrices stored at each machine are defined once and kept fixed during subsequent computations. The cs-matrices are specified by an $Z \times L$ MDS code generator matrix \mathbf{G} where $g_{i,\ell}$ denotes the element in the i -th row and ℓ -th column, where $Z, L \in \mathbb{Z}^+$. Let any L rows of \mathbf{G} be invertible. The data matrix, \mathbf{X} , is row-wise split into L disjoint, $\frac{q}{L} \times r$ uncoded sub-matrices, $\mathbf{X}_1, \dots, \mathbf{X}_L$. Then, for $i \in [Z]$, we define $\tilde{\mathbf{X}}_i = \sum_{\ell=1}^L g_{i,\ell} \mathbf{X}_\ell$, where $\tilde{\mathbf{X}}_i$ is the i -th cs-matrix. Each $\tilde{\mathbf{X}}_i, i \in [Z]$ has $\frac{q}{L}$ rows. Assume that machine n will store $\sigma[n]$ of these cs-matrices, specified by an index set \mathcal{Q}_n . In other words, machine n stores the cs-matrices of $\{\tilde{\mathbf{X}}_i : i \in \mathcal{Q}_n\}$, where $|\mathcal{Q}_n| = \sigma[n]$. We also assume that different machines store different cs-matrices: the sets $\mathcal{Q}_1, \dots, \mathcal{Q}_N$ are disjoint. Also, note that given a cs-matrix $\tilde{\mathbf{X}}_i, i \in [Z]$, there is a unique machine that stores this cs-matrix.

The machines collectively perform matrix-vector computations² over multiple time steps. In a given time step only a subset of the N machines are available to perform matrix computations. Note that, the available machines of each time step are known when we design the computation assignments. Specifically, in time step t , a set of available machines $\mathcal{N}_t \subseteq [N]$ aims to compute $\mathbf{y}_t = \mathbf{X} \mathbf{w}_t$ where \mathbf{w}_t is a vector of length r . The machines of $[N] \setminus \mathcal{N}_t$ are preempted. The number of cs-matrices stored at the available machines is $Z_t \triangleq \sum_{n \in \mathcal{N}_t} \sigma[n] \geq L$, which means that Z_t is at least L such that the desired computation can be recovered. Machines

¹Note that assuming $\sigma[n] \in \mathbb{Z}^+, \forall n \in [N]$ is for the ease of presentation. If any $\sigma[n]$ is a fractional number, then we can let Z be large such that all $\sigma[n]$ s can be a positive integer.

²It is straightforward to extend the proposed CEC designs to other applications outlined in [33] including matrix-matrix multiplications and linear regression because matrix-vector multiplication is integral to each of these applications.

in \mathcal{N}_t do not compute \mathbf{y}_t directly. Instead, each machine $n \in \mathcal{N}_t$ computes the set $\mathcal{V}_n = \bigcup_{i \in \mathcal{Q}_n} \{v = \tilde{\mathbf{X}}_i^{(j)} \mathbf{w}_t : j \in \mathcal{W}_i\}$, where $\tilde{\mathbf{X}}_i^{(j)}$ is the j -th row of $\tilde{\mathbf{X}}_i$, the cs-matrix stored by machine n , and $\mathcal{W}_i \subseteq [\frac{q}{L}]$ is the set of rows of $\tilde{\mathbf{X}}_i$ assigned to machine n in time step t for computing tasks.

B. Key Definitions

In the following, we introduce three key definitions that are useful to specify the computation assignments in a CEC network.

Definition 1: The computation load vector, $\boldsymbol{\mu}$, defined as

$$\mu[n] = \frac{|\mathcal{V}_n|}{(\frac{q}{L})} = \frac{\sum_{i \in \mathcal{Q}_n} |\mathcal{W}_i|}{(\frac{q}{L})}, \quad \forall n \in \mathcal{N}_t, \quad (1)$$

is the normalized number of rows computed by machine n in time step t . \diamond

Note that while $\boldsymbol{\mu}$, \mathcal{W}_i , and \mathcal{V}_n can change with each time step, reference to t is omitted for ease of presentation. Moreover, assume that machines have varying computation speeds defined by a strictly positive vector $\mathbf{s} = (s[1], s[2], \dots, s[N])$, $s[n] \in \mathbb{Q}^+, \forall n \in [N]$, fixed over all time steps. Here, computation speed is defined as the number of times a machine can compute its cs-matrix per unit time. We assume that the computation speed of each machine is known.

Definition 2: Given a computation load vector $\boldsymbol{\mu}$, the overall computation time is dictated by the machine(s) that takes the most time to perform its assigned computations. Thus, we define

$$c(\boldsymbol{\mu}) = \max_{n \in \mathcal{N}_t} \frac{\mu[n]}{s[n]}. \quad (2)$$

to be the overall computation time as a function of the computation load $\boldsymbol{\mu}$. \diamond

At each time step t , for each $j \in [\frac{q}{L}]$, the j -th row of L cs-matrices undergoes a vector-vector multiplication with \mathbf{w}_t . The results are sent to a master node which can resolve the elements of \mathbf{y}_t by the MDS code design. To ensure each row is assigned L times,³ we will introduce a general framework for defining computation assignments.

Given a cs-matrix $\tilde{\mathbf{X}}_i$, \mathcal{W}_i includes the subset of rows in $\tilde{\mathbf{X}}_i$ assigned to be computed by the machine that stores $\tilde{\mathbf{X}}_i$. In our design, instead of trying to determine this assignment row by row, we make the assignment in “blocks” of rows. Namely, each \mathcal{W}_i will include blocks of rows from $\tilde{\mathbf{X}}_i$. Furthermore, we will use a common set of blocks for the assignment of all $\mathcal{W}_i, i \in [Z]$, which we refer to as row sets. These are formally defined below.

Definition 3: Since each cs-matrix has $\frac{q}{L}$ rows, we partition the full index set of these rows $\{1, 2, \dots, \frac{q}{L}\}$ into F consecutive disjoint subsets, possibly with varying sizes, called row sets, denoted by $\mathcal{M}_t = (\mathcal{M}_1, \dots, \mathcal{M}_F)$, whose union gives the full index set. \diamond

³When each machine stores just 1 coded matrix, we say the computations are assigned to each machine as in the original CEC work [33]. Alternatively, when some machines store more than 1 matrix, each row is computed for L different matrices which are stored across a number of machines less than or equal to L .

Algorithm 1 Computation Assignment: Homogeneous Storage Capacity and Heterogeneous Computing Speeds

Input: μ^* , N_t , L , and q % μ^* is solution to first sub-problem (7)-(8).
1: $\mathbf{m} \leftarrow \mu^*$ % \mathbf{m} represents the remaining computation load vector to be assigned.
% Initialize \mathbf{m} as the optimal computation load vector μ^*
2: $f \leftarrow 0$
3: **while** \mathbf{m} contains a non-zero element **do**
4: $f \leftarrow f + 1$
5: $L' \leftarrow \sum_{n=1}^{N_t} m[n]$ % L' represents the sum of the remaining computation load
6: $N' \leftarrow$ number of non-zero elements in \mathbf{m}
7: $\ell \leftarrow$ indices that sort the non-zero elements of \mathbf{m} from smallest to largest⁵
8: $\mathcal{P}_f \leftarrow \{\ell[1], \ell[N' - L + 2], \dots, \ell[N']\}$ % specify machines that will compute \mathcal{M}_f
9: **if** $N' \geq L + 1$ **then** % α_f is the fraction of rows assigned to row set \mathcal{M}_f
10: $\alpha_f \leftarrow \min \left(\frac{L'}{L} - m[\ell[N' - L + 1]], m[\ell[1]] \right)$ % assign only a fraction of remaining rows to \mathcal{P}_f
% to ensure a FP solution exists at next iteration.
11: **else**
12: $\alpha_f \leftarrow m[\ell[1]]$ % assign all remaining un-assigned rows of machine $\ell[1]$ to \mathcal{P}_f .
13: **end if**
14: **for** $n \in \mathcal{P}_f$ **do**
15: $m[n] \leftarrow m[n] - \alpha_f$ % update remaining computation load at each machine
16: **end for**
17: **end while**
18: $F \leftarrow f$
19: Partition rows $\left[\frac{q}{L}\right]$ into F disjoint row sets: $\mathcal{M}_1, \dots, \mathcal{M}_F$ of size $\frac{\alpha_1 q}{L}, \dots, \frac{\alpha_F q}{L}$ rows
Output: $F, \mathcal{M}_1, \dots, \mathcal{M}_F$ and $\mathcal{P}_1, \dots, \mathcal{P}_F$

Definition 4: Given row sets $\mathcal{M}_t = (\mathcal{M}_1, \dots, \mathcal{M}_F)$, we define cs-matrix sets $\mathcal{P}_t = (\mathcal{P}_1, \dots, \mathcal{P}_F)$ where each \mathcal{P}_f includes the indices of L cs-matrices for which all rows in \mathcal{M}_f are computed by the machines that store these cs-matrices. Specifically, if $i \in \mathcal{P}_f$ and a machine n stores $\tilde{\mathbf{X}}_i$ ($i \in \mathcal{Q}_n$), then machine n will compute all rows in \mathcal{M}_f from $\tilde{\mathbf{X}}_i$, i.e., these rows are included in \mathcal{W}_i . This ensures that each row set \mathcal{M}_f is computed exactly L times using the L cs-matrices stored on these machines. \diamond

By the above definitions, the rows computed by machine $n \in \mathcal{N}_t$ in time step t are in the set

$$\bigcup_{i \in \mathcal{Q}_n} \mathcal{W}_i = \bigcup_{i \in \mathcal{Q}_n} \{\mathcal{M}_f : f \in [F], i \in \mathcal{P}_f\}. \quad (3)$$

Note that the sets $\mathcal{M}_1, \dots, \mathcal{M}_F$ and $\mathcal{P}_1, \dots, \mathcal{P}_F$ and F may vary with each time step.

Remark 1: When each machine only stores one cs-matrix, there is a one-to-one mapping between a cs-matrix and a machine, and thus \mathcal{P}_f also represents the set of machines that compute rows in \mathcal{M}_f . This is used in Example 1 and Algorithm 1, where we assume that the machines have heterogeneous speeds but homogeneous storage.

Remark 2: The row set and cs-matrix set pair $(\mathcal{M}_t, \mathcal{P}_t)$ combine to determine the computation assignment. The computation load μ is a function of $(\mathcal{M}_t, \mathcal{P}_t)$.

One example to illustrate \mathcal{M}_f and \mathcal{P}_f is given in Fig. 1(a). In this example, we consider the case that all machines have heterogeneous computing speed but with homogeneous storage, i.e., $\sigma[n] = 1, n \in [N]$. Each machine store one cs-matrix and the union of $\mathcal{M}_f, f \in [4]$ in different colors cover all the row indices, $\left[\frac{q}{L}\right]$. We let machine n stores

$\tilde{\mathbf{X}}_n$, then we can see that $\mathcal{P}_1 = \{1, 5, 6\}$, $\mathcal{P}_2 = \{2, 3, 4\}$, $\mathcal{P}_3 = \{3, 5, 6\}$, $\mathcal{P}_4 = \{4, 5, 6\}$. In this case, since there is a one-to-one mapping between the machine and the cs-matrix that it stores, the cs-matrix set \mathcal{P}_f can also be interpreted as an index set of the machines that compute the stored cs-matrices.

C. Formulation of a Combinatorial Optimization Framework

In a given time step t , our goal is to define the computation assignment, \mathcal{M}_t and \mathcal{P}_t , such that the resulting computation load vector defined in (1) has the minimum computation time. In time step t , given $\mathcal{N}_t, \mathcal{Q}_1, \dots, \mathcal{Q}_N$ and \mathbf{s} , the optimal computation time, c^* , is the minimum of computation time defined by all possible computation assignments, $(\mathcal{M}_t, \mathcal{P}_t)$. Hence, based on all the conditions discussed before and given the storage vector σ , we can formulate the following combinatorial optimization problem.

$$\underset{(\mathcal{M}_t, \mathcal{P}_t)}{\text{minimize}} \quad c(\mu(\mathcal{M}_t, \mathcal{P}_t)) \quad (4a)$$

$$\text{subject to:} \quad \bigcup_{\mathcal{M}_f \in \mathcal{M}_t} \mathcal{M}_f = \left[\frac{q}{L}\right], \quad (4b)$$

$$\mathcal{P}_f \subseteq \bigcup_{n \in \mathcal{N}_t} \mathcal{Q}_n, \quad \forall \mathcal{P}_f \in \mathcal{P}_t, \quad (4c)$$

$$|\mathcal{P}_f| = L, \quad \forall \mathcal{P}_f \in \mathcal{P}_t, \quad (4d)$$

$$|\mathcal{M}_t| = |\mathcal{P}_t|, \quad (4e)$$

The objective function (4a) is the overall computation time of a computation load vector μ , which is a function of $(\mathcal{M}_t, \mathcal{P}_t)$ and \mathbf{s} . Conditions (4b)-(4e) specifies the constraints on $(\mathcal{M}_t, \mathcal{P}_t)$, which are to be optimized over. Specifically, (4b) ensures that the union of the row sets \mathcal{M}_f

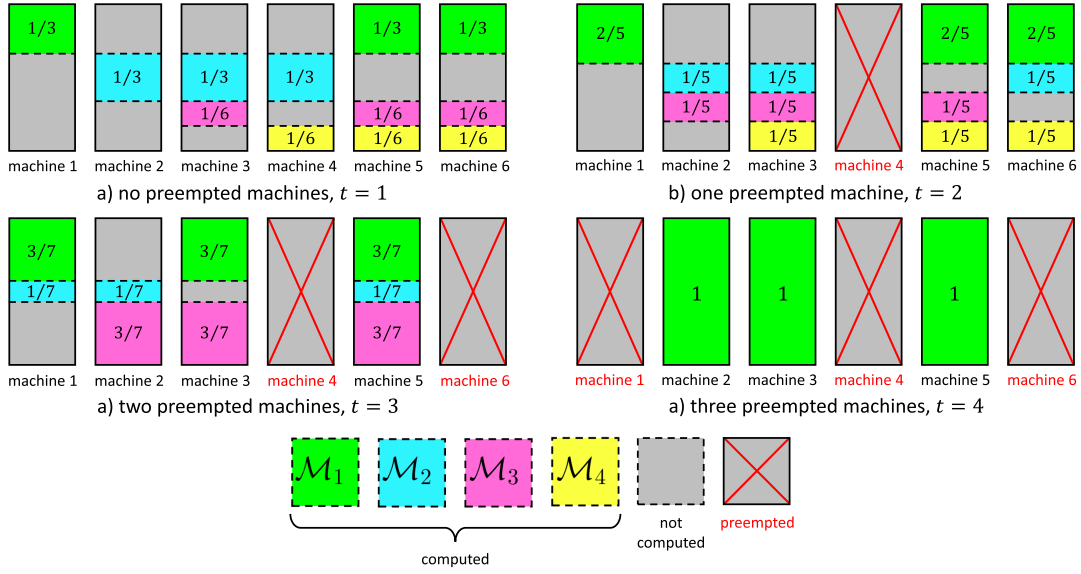


Fig. 1. An illustration of the optimal computation assignments in Example 1 over 4 time steps on a heterogeneous CEC network where machines have heterogeneous computing speed and homogeneous storage space. In this example, $N = 6$, $Z = 6$, $L = 3$ and $\mathcal{Q}_n = \{n\}, \forall n \in [6]$. At $t = 1$, there are $F = 4$ row sets \mathcal{M}_1 (green) \mathcal{M}_2 (blue), \mathcal{M}_3 (magenta), \mathcal{M}_4 (yellow), each is assigned to $L = 3$ cs-matrices. The number labeled in the center of each row set is the fraction of the rows in that row set. The row sets change over time. At $t = 3$, there are $F = 3$ row sets, and at $t = 4$, there is only $F = 1$ row set.

equals the full set of $\frac{q}{L}$ rows. i.e., all rows are assigned to be computed by some available machines at time t . Condition (4c) ensures that the cs-matrices in \mathcal{P}_f are stored by only available machines at time t , and hence, each row set \mathcal{M}_f is only assigned to be computed from these available machines. Condition (4d) ensures that each row set is computed from exactly L cs-matrices. Condition (4e) ensures that each row set has a corresponding cs-matrix set, i.e., the number of row sets equals the number of cs-matrix set.

Remark 3: In Sections IV and V, we precisely solve the combinatorial optimization problem of (4) and summarize this result in Theorem 2. We decompose this problem into two sub-problems. First, a convex optimization problem to find an optimal computation load vector μ without the consideration of a specific computation assignment. This means that we will solve the problem of (4) by treating μ as a real vector without considering whether such a computation assignment is feasible. Second, given the optimal μ solved in the previous convex optimization problem, we solve a computation assignment problem or a filling problem in order to find a $(\mathcal{M}_t, \mathcal{P}_t)$ that meets the optimal computation load. Moreover, we show that an optimal assignment, $(\mathcal{M}_t, \mathcal{P}_t)$, can be found via a low complexity algorithm that completes in at most N_t iterations and the number of computation assignments, F , is at most N_t .

Remark 4: Note that, the optimization problem of (4) does not consider the time required for decoding or communications, but only optimizes the time of overall computation. Furthermore, our problem formulation considers the optimal computation assignment to available machines that will remain available during that computation time step. In this work, we do not consider elastic events or stragglers during a computation time step.

III. TWO CEC EXAMPLES

In this section, we will discuss two CEC examples to illustrate the proposed approaches. Example 1 considers

the scenario where machines have heterogeneous computing speeds but homogeneous storage constraints. Example 2 considers the more general scenario where machines have both heterogeneous storage space and computing speeds.

A. Example 1: CEC With Heterogeneous Computing Speeds, Homogeneous Storage Constraints

We consider a system with a total of $N = 6$ machines where each has the storage capacity to store $\frac{1}{3}$ of the data matrix \mathbf{X} . In time step t , the machines have the collective goal of computing $\mathbf{y}_t = \mathbf{X}\mathbf{w}_t$ where \mathbf{w}_t is some vector. In order to allow for preempted machines, \mathbf{X} is split row-wise into $L = 3$ sub-matrices, \mathbf{X}_1 , \mathbf{X}_2 and \mathbf{X}_3 and an MDS code is used to construct the cs-matrices $\{\tilde{\mathbf{X}}_n : n \in [N]\}$ which are stored among the machines. In particular, when the storage space among machines is homogeneous, machine n stores only one cs-matrix $\tilde{\mathbf{X}}_n$ and $\sigma = [1, \dots, 1]$. In this case, there is a one-to-one mapping between cs-matrices and machines. This placement is designed such that any element of \mathbf{y}_t can be recovered by obtaining the corresponding coded computation from any $L = 3$ machines. To recover the entirety of \mathbf{y}_t , we split the cs-matrices into row sets, such that each set is used for computation at $L = 3$ machines.

The machines have relative computation speeds defined by $\mathbf{s} = [2, 2, 3, 3, 4, 4]$, where machines 5 and 6 are the fastest and can perform row computations twice as fast as machines 1 and 2. Machines 3 and 4 are the next fastest and can perform matrix computations 1.5 times as fast as machines 1 and 2. Our goal is to assign computations, or rows of the cs-matrices, to the machines to minimize the overall computation time with the constraint that each computation is assigned to $L = 3$ machines.

In time step 1, there are no preempted machines which means that $\mathcal{N}_1 = \{1, \dots, 6\}$ and $N_1 = 6$. We assign fractions of the rows to the machines defined by the computation load

vector $\mu = [\frac{1}{3}, \frac{1}{3}, \frac{1}{2}, \frac{1}{2}, \frac{2}{3}, \frac{2}{3}]$, such that machines 1 and 2 are assigned $\frac{1}{3}$, machines 3 and 4 are assigned $\frac{1}{2}$ and machines 5 and 6 are assigned $\frac{2}{3}$ of the rows of their respective cs-matrices. We define μ such that it sums to $L = 3$ and each row can be assigned to 3 machines. Furthermore, based on the machine computation speeds, the machines finish at the same time to minimize the overall computation time. In Section IV, we outline the systematic approach to determine μ . Next, given μ , the rows of the cs-matrices must be assigned. We define row sets, $\mathcal{M}_1, \mathcal{M}_2, \mathcal{M}_3$, and \mathcal{M}_4 which are assigned to $F = 4$ sets of $L = 3$ cs-matrices $\mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3$, and \mathcal{P}_4 . Since each machine n stores one cs-matrix \tilde{X}_n , it is equivalent to say that we assign computations to machines. In other words $\mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3$, and \mathcal{P}_4 represent the machines assigned to compute $\mathcal{M}_1, \mathcal{M}_2, \mathcal{M}_3$, and \mathcal{M}_4 , respectively. These sets are depicted in Fig. 1(a) where, for example, \mathcal{M}_1 contains the first $\frac{1}{3}$ of the rows assigned to machines $\mathcal{P}_1 = \{1, 5, 6\}$. Moreover, \mathcal{M}_2 contains the next $\frac{1}{3}$ of the rows assigned to machines $\mathcal{P}_2 = \{2, 3, 4\}$, \mathcal{M}_3 contains the next $\frac{1}{6}$ of the rows assigned to machines $\mathcal{P}_3 = \{3, 5, 6\}$ and \mathcal{M}_4 contains the final $\frac{1}{6}$ of the rows assigned to machines $\mathcal{P}_4 = \{4, 5, 6\}$. In Section V, we present Algorithm 1 to determine the computation assignment for general μ . By this assignment, the fraction of rows assigned to machine n sums to $\mu[n]$ and each row is assigned to $L = 3$ machines to recover the entirety of y_1 .

In time step 2, we find $N_2 = 5$ because machine 4 is preempted and no longer available to perform computations. Therefore, the computations must be re-assigned among $\mathcal{N}_2 = \{1, 2, 3, 5, 6\}$. First, we obtain $\mu = [\frac{2}{5}, \frac{2}{5}, \frac{3}{5}, 0, \frac{4}{5}, \frac{4}{5}]$, which sums to $L = 3$ and minimizes the overall computation time. Given μ , we then use Algorithm 1 (see Section V) to assign computations to a machine with the least number of remaining rows to be assigned and $L - 1 = 2$ machines with the most number of remaining rows to be assigned. For example, in the first iteration, \mathcal{M}_1 is defined to contain the first $\frac{2}{5}$ of the rows and is assigned to machines $\mathcal{P}_1 = \{1, 5, 6\}$. After this iteration, machines 2, 5 and 6 require $\frac{2}{5}$ of the total rows to still be assigned to them and machine 3 requires $\frac{3}{5}$ of the total rows. In the next iteration, \mathcal{M}_2 contains the next $\frac{1}{5}$ of the rows and is assigned to $\mathcal{P}_2 = \{2, 3, 6\}$. Note that, only $\frac{1}{5}$ of the rows could be assigned in this iteration otherwise there would only be two machines, 3 and 5, which still require assignments and therefore, the remaining rows cannot be assigned to 3 machines. In the final two iterations, \mathcal{M}_3 and \mathcal{M}_4 contain $\frac{1}{5}$ of the previously unassigned rows and are assigned to the machines of $\mathcal{P}_3 = \{2, 3, 5\}$ and $\mathcal{P}_4 = \{3, 5, 6\}$, respectively. These assignments are depicted in Fig. 1(b).

Next, in time step 3, we find $N_3 = 4$ and $\mathcal{N}_3 = \{1, 2, 3, 5\}$ because machines 4 and 6 are preempted. It is ideal to have machines 3 and 5 compute $1.5\times$ and $2\times$ the number of computations, respectively, compared to machines 1 and 2. However, this is impossible since each machine can be assigned at most a number of rows equal to the number of rows of the cs-matrices. In this case, we assign all rows to the fastest machine, machine 5, and assign fractions of the rows to the remaining machines which sum up to 2. As a

result, we let $\mu = [\frac{4}{7}, \frac{4}{7}, \frac{6}{7}, 0, 1, 0]$. Then, Algorithm 1 defines $\mathcal{M}_1, \mathcal{M}_2$ and \mathcal{M}_3 as disjoint sets containing $\frac{3}{7}, \frac{1}{7}$ and $\frac{3}{7}$ of the rows, respectively. Moreover, these row sets are assigned to the machines of $\mathcal{P}_1 = \{1, 3, 5\}$, $\mathcal{P}_2 = \{1, 2, 5\}$ and $\mathcal{P}_3 = \{2, 3, 5\}$, respectively. These assignments are depicted in Fig. 1(c).

Finally, in time step 4, machines 1, 4 and 6 are preempted which means that $\mathcal{N}_4 = \{2, 3, 5\}$ and $N_4 = 3$. To assign all the rows to $L = 3$ machines, each available machine is assigned by all of the rows and $\mu = [0, 1, 1, 0, 1, 0]$. In other words, \mathcal{M}_1 contains all rows and $\mathcal{P}_1 = \{2, 3, 5\}$. This is depicted in Fig. 1(d).

Next, we present Example 2 with heterogeneous storage space and computing speeds. This example uses Algorithm 2, which is a generalization of Algorithm 1 discussed in Example 1.

B. Example 2: CEC With Heterogeneous Computing Speeds and Storage Constraints

Consider the case where $L = 6$ and there are $N = 6$ machines which each have distinct speed-storage pairs. For ease of presentation, we only focus on a single time step (or $t = 1$) and assume there are no preempted machines ($N_1 = 6$). The computing speed of the available machines are defined by $s = [2, 3, 4, 2, 3, 4]$. The total number of stored cs-matrices is $Z_1 = 9$ and the machines store a number of cs-matrices defined by $\sigma = [2, 2, 2, 1, 1, 1]$. For example, machine 1 has a speed of 2 and stores 2 cs-matrices and machine 5 has a speed of 3 and stores 1 cs-matrix. The storage of the cs-matrices at each machine is shown in Fig. 2 where \tilde{X}_i is labeled at the top of each block which represent a cs-matrix. The machines are in descending order based on $\frac{\sigma[n]}{s[n]}$ to use Theorem 1 to find the optimal computation load vector described in Section IV. Based on Theorem 1, the computation load vector is $\mu = [\frac{8}{11}, \frac{12}{11}, \frac{16}{11}, \frac{8}{11}, 1, 1]$. Notice that, different from Example 1, a machine here may have a computation load greater than 1 if it performs computations on more than 1 stored cs-matrices. However, similar to Example 1, machines either have the same computation time or perform computations on all locally stored data. In this example, based on the computation speeds, machines 1 through 4 complete the assigned computing tasks at the same time and machines 5 and 6 compute using the entirety of their one locally available cs-matrix and finish before the other machines.

Next, we need to assign computations that yield the computation load vector, μ . We use Algorithm 2 in Section V, where instead of assigning computations to machines, we assign computations to each cs-matrix at each machine. In this algorithm, we need to first decide how much of each cs-matrix will be computed. For example, consider machine 3 which has a computation load of $\mu[3] = \frac{16}{11}$ and locally stores $\sigma[3] = 2$ cs-matrices. There is a choice of how much machine 3 will compute each of its cs-matrices. A solution which simplifies the assignment is for machine 3 to compute the entirety of one cs-matrix and a $\frac{5}{11}$ fraction of the other. Similarly, machine 2 will compute the entirety of one cs-matrix and $\frac{1}{11}$ fraction of the other. In general, when $\sigma[n] > 1$ and machine n

Algorithm 2 Computation Assignment: Heterogeneous Storage Capacity and Computing Speeds

Input: μ^* , N_t , L , q , $\tilde{Q}_1, \dots, \tilde{Q}_{N_t}$ and $\hat{\theta}$ % μ^* is solution to first sub-problem (7)-(8). $\tilde{Q}_1, \dots, \tilde{Q}_{N_t}$ and $\hat{\theta}$ include pre-chosen cs-matrices based on μ^* .

1: **for** $n \in [N_t]$ **do** % determine the computation load for each partially computed cs-matrix.

2: $\hat{\mu}[n] \leftarrow \mu^*[n] - \lfloor \mu^*[n] \rfloor$

3: **end for**

4: $\hat{L} \leftarrow \sum_{n=1}^{N_t} \hat{\mu}[n]$ % \hat{L} is the sum of the computation load over partially computed cs-matrices.
% Use Algorithm 1 next to find computation assignment for partially computed matrices.

5: $\hat{F}, \hat{\mathcal{M}}_1, \dots, \hat{\mathcal{M}}_F$ and $\hat{\mathcal{P}}_1, \dots, \hat{\mathcal{P}}_F \leftarrow$ Output of **Algorithm 1** with $\hat{\mu}$, N_t , \hat{L} , and q as input

6: $F \leftarrow \hat{F}$

7: **for** $f \in [F]$ **do** % use the output row sets from Algorithm 1 as final row sets.

8: $\mathcal{M}_f \leftarrow \hat{\mathcal{M}}_f$

9: $\mathcal{P}_f \leftarrow \bigcup_{n \in \hat{\mathcal{P}}_f} \hat{\theta}[n] \cup \bigcup_{n \in [N_t]} \tilde{Q}_n$ % combine outputs of Algorithm 1 with fully computed cs-matrices
% to obtain the final cs-matrices assignment

10: **end for**

Output: $F, \mathcal{M}_1, \dots, \mathcal{M}_F$ and $\mathcal{P}_1, \dots, \mathcal{P}_F$

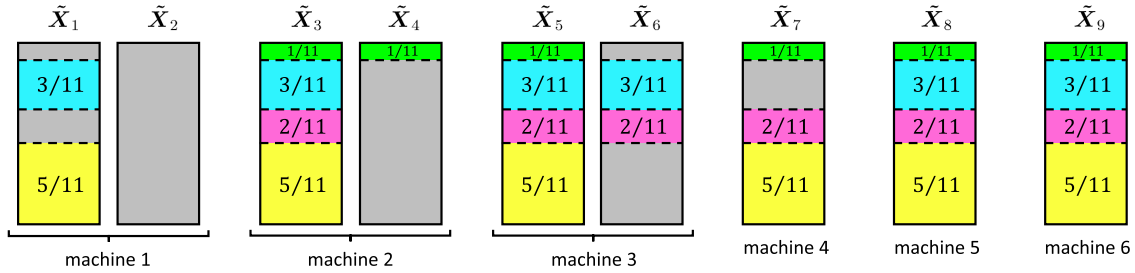


Fig. 2. Example 2: Optimal computation assignments for a CEC network with machines that have heterogeneous storage requirements and varying computations speeds. Here, we have $N = 6$, $Z = 9$, $L = 6$. The machines have varying computation speeds $s = [2, 3, 4, 2, 3, 4]$ and storage capacity $\sigma = [2, 2, 2, 1, 1, 1]$. Machines 1 to 3 each stores 2 cs-matrices, and machines 4 to 6 each stores only 1 cs-matrix. The machines are ordered in the decreasing order of SCR $\frac{\sigma[n]}{s[n]}$. There are 4 row sets \mathcal{M}_1 (green) \mathcal{M}_2 (blue), \mathcal{M}_3 (magenta), \mathcal{M}_4 (yellow), each is assigned to $L = 6$ cs-matrices. For instance, \mathcal{M}_1 is assigned to cs-matrices $\mathcal{P}_1 = \{3, 4, 5, 7, 8, 9\}$. The optimal computation load vector is $\mu = [\frac{8}{11}, \frac{12}{11}, \frac{16}{11}, \frac{8}{11}, 1, 1]$. For instance, machine 3 computes one full cs-matrix \tilde{X}_5 and partially computes $\frac{3}{11} + \frac{2}{11} = \frac{5}{11}$ of \tilde{X}_6 , which adds up to a computation load of $\mu[3] = 1 + \frac{5}{11} = \frac{16}{11}$.

stores more than one cs-matrices, it will compute $\lfloor \mu[n] \rfloor$ whole cs-matrices and a $\mu[n] - \lfloor \mu[n] \rfloor$ fraction of the remaining cs-matrix.

The final computation assignments are shown in Fig. 2. There are $F = 4$ matrix row sets \mathcal{M}_1 through \mathcal{M}_4 which contain a $\frac{1}{11}$, $\frac{3}{11}$, $\frac{2}{11}$ and $\frac{5}{11}$ fraction of rows, respectively. Each row set \mathcal{M}_f is assigned to a cs-matrix set \mathcal{P}_f that contains $L = 6$ cs-matrices. These are given by cs-matrix sets $\mathcal{P}_1 = \{3, 4, 5, 7, 8, 9\}$, $\mathcal{P}_2 = \{1, 3, 5, 6, 8, 9\}$, $\mathcal{P}_3 = \{3, 5, 6, 7, 8, 9\}$, and $\mathcal{P}_4 = \{1, 3, 5, 7, 8, 9\}$.

IV. FIRST SUB-PROBLEM: OPTIMAL COMPUTATION LOAD VECTOR

We decompose the optimization problem (4) into two sub-problems. In this section, we present the first sub-problem by introducing a relaxed convex optimization problem to find the optimal computation load vector μ^* and its corresponding computation time $\hat{c}^* = c(\mu^*)$ without considering an explicit computation assignment $(\mathcal{M}_t, \mathcal{P}_t)$. Due to the relaxed constraints, we have $\hat{c}^* \leq c^*$. Next, in Section V, we will present the second sub-problem where we show a computation assignment $(\mathcal{M}_t, \mathcal{P}_t)$ exists that achieves the μ^* found in the

first step. Hence, there is no gap between the relaxed convex optimization problem and (4) and we have $c^* = \hat{c}^*$.

A. The Proposed Relaxed Convex Optimization Problem

Given a computation speed vector s and storage vector σ , we let the optimal computation load vector μ^* be the solution to the following convex optimization problem:

$$\underset{\mu}{\text{minimize}} \quad c(\mu) = \max_{n \in [N_t]} \frac{\mu[n]}{s[n]} \quad (5a)$$

$$\text{subject to:} \quad \sum_{n \in [N_t]} \mu[n] = L, \quad (5b)$$

$$0 \leq \mu[n] \leq \sigma[n], \quad \forall n \in [N_t], \quad (5c)$$

$$\mu[n] \in \mathbb{R}^+, \quad \forall n \in [N_t] \quad (5d)$$

which can be shown to be a convex optimization problem. While computation assignments, $(\mathcal{M}_t, \mathcal{P}_t)$, are not explicitly considered in (5), we note that the key constraint of $\sum_{n \in [N_t]} \mu[n] = L$ is a relaxed version of the requirement on the computation assignment that each row set should be assigned to L cs-matrices. The analytical solution to the optimization problem (5) can be explicitly found. When $Z_t = L$, it can be seen that this optimal solution is given by $\mu^* = \sigma$.

When $Z_t > L$, the analytical optimal solution to (5) is presented in the following theorem.

Theorem 1: Assume that $Z_t > L$ and the machines are labeled in the decreasing order of the storage capacity to computation speed ratio (SCR)

$$\frac{\sigma[1]}{s[1]} \geq \frac{\sigma[2]}{s[2]} \geq \dots \geq \frac{\sigma[N_t]}{s[N_t]}. \quad (6)$$

The optimal solution μ^* to the optimization problem of (5) must take the following form

$$\mu^*[n] = \begin{cases} \hat{c}^* s[n] & \text{if } 1 \leq n \leq k^* \\ \sigma[n] & \text{if } k^* + 1 \leq n \leq N_t, \end{cases} \quad (7)$$

where k^* is the largest integer in $[N_t]$ such that

$$\begin{aligned} \frac{\sigma[k^* + 1]}{s[k^* + 1]} &< \hat{c}^* = \frac{L - \sum_{n=k^*+1}^{N_t} \sigma[n]}{\sum_{n=1}^{k^*} s[n]} \\ &\leq \frac{\sigma[k^*]}{s[k^*]}, \quad \text{if } k^* < N_t, \end{aligned} \quad (8)$$

otherwise, $k^* = N_t$. Here, $\hat{c}^* = c(\mu^*)$ is the maximum computation time among the N_t machines given the computation load assignment μ^* .

B. Proof of Theorem 1

In the following, we first present two Claims that will lead to the proof of Theorem 1.

Claim 1: If $\mu^*[n] < \hat{c}^* s[n]$, then $\mu^*[n] = \sigma[n]$. Thus, in this case the optimal computation load assigned to machine n is equivalent to its storage.

Proof: We prove Claim 1 by contradiction. Since $\hat{c}^* = \max_{n \in [N_t]} \frac{\mu^*[n]}{s[n]}$, we define two disjoint sets \mathcal{T}_0 and \mathcal{T}_1 , where $\mathcal{T}_0 \cup \mathcal{T}_1 = [N_t]$, as

$$\mathcal{T}_0 = \{n \in [N_t] : \mu^*[n] = \hat{c}^* s[n]\} \quad (9)$$

and

$$\mathcal{T}_1 = \{n \in [N_t] : \mu^*[n] < \hat{c}^* s[n]\}. \quad (10)$$

In the following, we will show that if there exists an $i \notin \mathcal{T}_0$, then we must have $i \in \mathcal{T}_1$ and $\mu^*[i] = \sigma[i]$. In order to do this, we will construct a new solution μ' from the optimal solution μ^* such that $c(\mu') < \hat{c}^*$, which leads to a contradiction that μ^* is an optimal solution. The details are as follows. Assume that there exists some $i \in [N_t]$ such that $i \in \mathcal{T}_1$ and $\mu^*[i] < \sigma[i]$. Define μ' such that

$$\mu'[n] = \begin{cases} \mu^*[n] + \epsilon & \text{if } n = i, \\ \mu^*[n] - \frac{\epsilon}{|\mathcal{T}_0|} & \text{if } n \in \mathcal{T}_0, \\ \mu^*[n] & \text{if } n \in \mathcal{T}_1 \setminus i \end{cases} \quad (11)$$

where $0 < \epsilon < \sigma[i] - \mu^*[i]$ and ϵ is sufficiently small such that

$$\frac{\mu'[i]}{s[i]} = \frac{\mu^*[i] + \epsilon}{s[i]} < \hat{c}^*, \quad (12)$$

and for all $n \in \mathcal{T}_0$

$$\mu^*[n] - \frac{\epsilon}{|\mathcal{T}_0|} > 0. \quad (13)$$

One can verify that we have $\frac{\mu'[n]}{s[n]} < \hat{c}^*$ for any $n \in [N_t]$ and thus we obtain $c(\mu') < \hat{c}^*$. This contradicts with the assumption that μ^* is optimal. Thus, it follows that if $n \notin \mathcal{T}_0$, then we must have $n \in \mathcal{T}_1$ and $\mu^*[n] = \sigma[n]$. ■

Claim 2: If $j \in \mathcal{T}_0$ and $i \in \mathcal{T}_1$, then $\frac{\sigma[j]}{s[j]} > \frac{\sigma[i]}{s[i]}$.

Proof: This claim follows directly from

$$\frac{\mu^*[i]}{s[i]} = \frac{\sigma[i]}{s[i]} < \hat{c}^* = \frac{\mu^*[j]}{s[j]} \leq \frac{\sigma[j]}{s[j]}. \quad (14)$$

Proof of Theorem 1: Combining Claims 1 and 2, the optimal solution must take the form of

$$\mu^*[n] = \begin{cases} \hat{c}_k^* s[n] & \text{if } 1 \leq n \leq k, \\ \sigma[n] & \text{if } k+1 \leq n \leq N_t, \end{cases} \quad (15)$$

where $k = |\mathcal{T}_0|$. Next, we will optimize k such that \hat{c}_k^* is minimized. Since $\sum_{n \in [N_t]} \mu^*[n] = L$, by using (15), we obtain (8) because

$$\begin{aligned} L = \sum_{n=1}^{N_t} \mu^*[n] &= \sum_{n=1}^k \mu^*[n] + \sum_{n=k+1}^{N_t} \sigma[n] \\ &= \hat{c}_k^* \sum_{n=1}^k s[n] + \sum_{n=k+1}^{N_t} \sigma[n], \end{aligned} \quad (16)$$

and

$$\hat{c}_k^* = \frac{L - \sum_{n=k+1}^{N_t} \sigma[n]}{\sum_{n=1}^k s[n]}. \quad (17)$$

The right-most inequality of (8) follows from $k \in \mathcal{T}_0$ and $\mu^*[k] \leq \sigma[k]$. The left-most inequality of (8) follows from $k+1 \in \mathcal{T}_1$ and $\mu^*[k+1] = \sigma[k+1]$. Since $\frac{\sigma[n]}{s[n]}$ is a decreasing sequence, we see from (8) that \hat{c}_k^* is minimized when k is chosen to be k^* , the largest value in $[N_t]$ such that (8) is satisfied.

C. Discussions on Theorem 1

From (6), (7) and (8), we can observe that the optimal solution μ^* to the optimization problem (5) is always rational due to the fact that s are rational numbers and σ are integers. Hence, it is achievable for large enough q if the computation assignment exists. The following corollary presents the solution of the optimal computation load vector when the storage among machines is homogeneous, i.e., each machine stores exactly one cs-matrix. This storage design is equivalent to that used in the original CEC work of [30], but here the machines have varying speeds as opposed to the homogeneous setting of [30].

Corollary 1: When $\sigma = [1, 1, \dots, 1]$, we find

$$\mu^*[n] = \begin{cases} \hat{c}^* s[n] & \text{if } 1 \leq n \leq k^* \\ 1 & \text{if } k^* + 1 \leq n \leq N_t, \end{cases} \quad (18)$$

where k^* is the largest integer in $[N_t]$ such that

$$\frac{1}{s[k^* + 1]} < \hat{c}^* = \frac{L - N_t + k^*}{\sum_{n=1}^{k^*} s[n]} \leq \frac{1}{s[k^*]}. \quad (19)$$

Proof: Corollary 1 is proved by substituting $\sigma[n] = 1$ for $n \in [N_t]$ in equations (7) and (8) and ordering the machines by speed in ascending order. ■

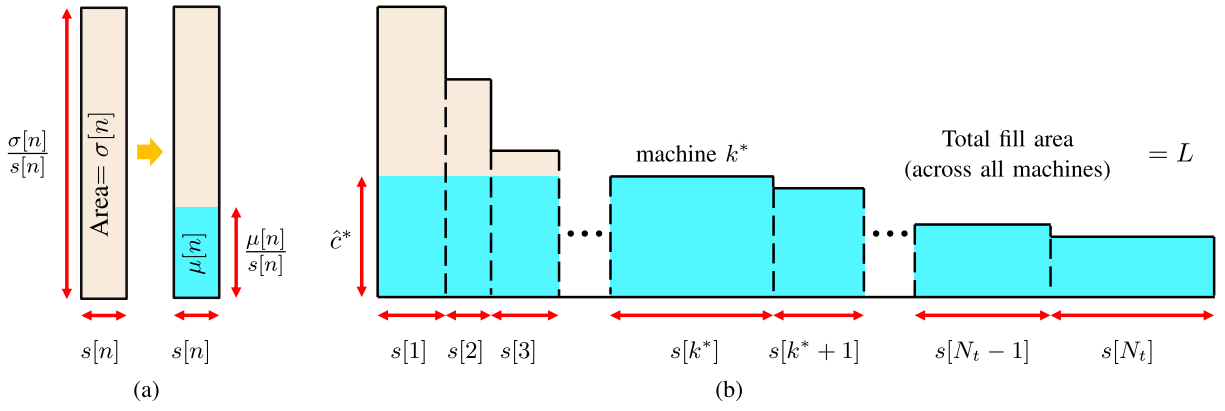


Fig. 3. A water-filling like representation of the storage and computation load of (a) for machine n only and (b) for a set of available machines with an optimal computation load vector that solves the optimization of (5) with the solution of Theorem 1. Machines in (b) are ordered in the decreasing order of SCR, $\frac{\sigma(n)}{s(n)}$. The storage of machine n is represented by the area of the full rectangle (in peach color). The computation load $\mu[n]$ assigned to machine n is represented by the area of the filled blue region within the n -th rectangle. The height of the each filled blue region represents $\frac{\mu[n]}{s[n]}$, which is the computation time of machine n . The maximum height of these filled regions represent the computation time $\hat{c}^* = c(\mu^*)$.

Remark 5: The two cases in (7) are determined by whether a machine n satisfies $\mu^*[n] = \hat{c}^*s[n]$ or $\mu^*[n] < \hat{c}^*s[n]$. For the first case when $1 \leq n \leq k^*$, the equality is achieved and we must have $0 < \mu^*[n] \leq \sigma[n]$. Among these k^* machines, the computation load $\mu^*[n]$ is proportional to the computation speed $s[n]$. For the second case when $k^* + 1 \leq n \leq N$, we have the strict inequality and $\mu^*[n] = \sigma[n]$. The computation load $\mu^*[n]$ equals (thus is limited by) the storage $\sigma[n]$. The equality in (8) ensures that $\sum_{n=1}^{N_t} \mu^*[n] = L$; the right-most inequality ensures that $\mu^*[n] \leq \mu^*[k^*] = \hat{c}^*s[k^*] \leq \sigma[n]$, for any $1 \leq n \leq k^*$; the left-most inequality ensures that for any $k^* + 1 \leq n \leq N$, we have $\mu^*[n] < \hat{c}^*s[n]$. Hence, the computation time \hat{c}^* is equal to the local computation time of any of the k^* machines with the largest SCR.

Since the optimization problem of (5) aims to minimize a convex function on a closed and convex set, the existence of an optimal solution is guaranteed. This ensures the existence of some $k^* \in [N_t]$ such that (8) is satisfied. In the following, we provide a numerical procedure to find k^* . First, it is straightforward to verify that if the right-hand-side (RHS) inequality “ \leq ” of (8) is violated for $k^* = i$, then the left-hand-side (LHS) inequality “ $<$ ” of (8) must hold for $k^* = i - 1$. In other words, for any $i \in [N_t]$,

$$\text{if } \hat{c}_i^* > \frac{\sigma[i]}{s[i]}, \text{ then } \frac{\sigma[i]}{s[i]} < \hat{c}_{i-1}^*. \quad (20)$$

where \hat{c}_i^* (and \hat{c}_{i-1}^*) are defined by (17) for different values of k^* . We first check $k^* = N_t$. If the RHS of (8) holds, then we have $k^* = N_t$. Otherwise, it follows from (20) that the LHS of (8) must hold for $k^* = N_t - 1$. If the RHS of (8) also hold for $k^* = N_t - 1$, then we have $k^* = N_t - 1$. Otherwise, it follows from (20) that the LHS of (8) must hold for $k^* = N_t - 2$. We continue this process by decreasing k^* until we find one value of k^* for which both sides of (8) hold. This process is guaranteed to terminate before reaching $k^* = 1$ for which the RHS of (8) always hold. Hence, this establishes the procedure to find k^* directly using (8).

Finally, we note that the solution in Theorem 1 for the optimization problem of (5) has a “water-filling” like visualization as shown in Fig. 3. In Fig. 3(a), the storage of machine n is represented by the area of the full rectangle (shaded with the peach color) that it corresponds to. We make the width of the rectangle $s[n]$ because if we “fill” part of the rectangle with an area of $\mu[n]$ (shaded in blue), then the height of the filled area, which is the water level at that rectangle, represents the computation time of machine n . Note that, this filled area does not represent a specific computation assignment, but only the total computations assigned to machine n . In Fig. 3(b), in accordance to (6), we arrange the available machines in descending order of the rectangle height, which is $\frac{\sigma[n]}{s[n]}$ for machine n . Then, following (5b), we “fill” all the available machines with a total area of L . First, notice that machines with larger rectangle width, or speed, will have larger filled area, or more computation load, until they are completely filled. Machines $k^* + 1, \dots, N_t$ with smaller rectangle height fill completely and have a computation time strictly less than \hat{c}^* . Machines $1, \dots, k^*$ with larger rectangle height, all have the same computation time of \hat{c}^* . Note that one or more of these machines may be completely filled such as machine k^* in Fig. 3(b), but will have the same “water level” as machines $1, \dots, k^*$.

D. Computation Load of Example 1 and Example 2

Homogeneous storage: We return to Example 1 presented in Section III-A with homogeneous storage but heterogeneous computing speeds and explain how to find the optimal computation load vector. In this example, each machine stores exactly one cs-matrix. When $t = 1$, we have $N_1 = 6$ and $L = 3$. Given $s = [2, 2, 3, 3, 4, 4]$, the largest k^* that satisfies (8) is $k^* = 6$, and thus $\hat{c}^* = 1/6$, $\mu^* = \hat{c}^*s = [\frac{1}{3}, \frac{1}{3}, \frac{1}{2}, \frac{1}{2}, \frac{2}{3}, \frac{2}{3}]$. Similarly, for $t = 2$, since machine 4 is preempted, we have now $N_2 = 5$, $\mathcal{N}_2 = \{1, 2, 3, 5, 6\}$ and $s = [2, 2, 3, 4, 4]$ (we ignore any preempted machines). In this case, we have $k^* = 5$, and thus $\hat{c}^* = 1/5$, $\mu^* = \hat{c}^*s = [\frac{2}{5}, \frac{2}{5}, \frac{3}{5}, \frac{4}{5}, \frac{4}{5}]$.

Similarly, for $t = 3$, we have $N_3 = 4$, $\mathcal{N}_3 = \{1, 2, 3, 5\}$ and $\mathbf{s} = [2, 2, 3, 4]$ because machines 4 and 6 preempt. Here, we have $k^* = 3$, $\hat{c}^* = 2/7$, and $\boldsymbol{\mu}^* = [\frac{4}{7}, \frac{4}{7}, \frac{6}{7}, 1]$.⁴

Heterogeneous storage: We illustrate this case using Example 2 presented in Section III-B with $L = N = 6$ and no preempted machines. In this case, we order the machines in a descending order by $\frac{\sigma[n]}{s[n]}$, where $\mathbf{s} = [2, 3, 4, 2, 3, 4]$ and $\boldsymbol{\sigma} = [2, 2, 2, 1, 1, 1]$. Next, we need to determine k^* , and we start by checking $k^* = 6$. However, we can observe that (8) does not hold since

$$\frac{L}{\sum_{n=1}^{N_t} s[n]} = \frac{1}{3} > \frac{1}{4} = \frac{\sigma[6]}{s[6]}. \quad (21)$$

Similarly, if we try $k^* = 5$, we see that (8) does not hold since

$$\frac{L - \sigma[6]}{\sum_{n=1}^5 s[n]} = \frac{5}{14} > \frac{1}{3} = \frac{\sigma[5]}{s[5]}. \quad (22)$$

Finally, we see that $k^* = 4$ is the solution that satisfies (8) because

$$\frac{\sigma[5]}{s[5]} = \frac{1}{3} < \frac{L - \sigma[5] - \sigma[6]}{\sum_{n=1}^4 s[n]} = \frac{4}{11} \leq \frac{1}{2} = \frac{\sigma[4]}{s[4]}. \quad (23)$$

It follows that $\hat{c}^* = 4/11$ and by using (7), we obtain $\boldsymbol{\mu}^* = [\frac{8}{11}, \frac{12}{11}, \frac{16}{11}, \frac{8}{11}, 1, 1]$.

In Section V, we will show that there always exists a computation assignment $(\mathcal{M}_t, \mathcal{P}_t)$ whose computation load vector equals $\boldsymbol{\mu}^*$ and the assignment pair can be found using the proposed Algorithm 1 (for homogeneous storage) and Algorithm 2 (for heterogeneous storage) in no more than N_t iterations.

V. SECOND SUB-PROBLEM: OPTIMAL COMPUTATION ASSIGNMENT

In this section, we present a computation assignment $(\mathcal{M}_t, \mathcal{P}_t)$ that solves the optimization problem of (4). First, we show the existence of a computation assignment that yields the computation load vector $\boldsymbol{\mu}^*$ and computation time \hat{c}^* . This shows that there is no gap between the combinatorial optimization problem (4) and the *relaxed* convex optimization problem of (5). Then, we provide a low-complexity iterative algorithm that outputs such an assignment in at most N_t iterations. In the following, we start with the case of homogeneous storage and heterogeneous computing speeds, where each machine stores exactly one cs-matrix, then we move to the case of heterogeneous storage and computing speed requirements where each machine may store any integer number of cs-matrices.

A. Homogeneous Storage With Heterogeneous Computing Speeds

Here, we focus on the case where $\sigma[n] = 1$ for $n \in [N_t]$ such that each available machine stores exactly one cs-matrix. Our goal is to assign computations among the

machines such that each row set in \mathcal{M}_t is assigned to L machines and the assignments satisfy the $\boldsymbol{\mu}^*$ given by (18). Interestingly, we find that once $\boldsymbol{\mu}^*$ is given, we can adapt the *filling problem (FP)* introduced in [32] for private information retrieval (PIR) to solve our second sub-problem of finding the computation assignment for CEC networks. Note that our proposed formulation of the computation assignments based on row sets, together with the two-step approach to solve the proposed combinatorial optimization problem, are important to allow successful adaptation of the FP problem [32] to the CEC setting.

Definition 5: $(\boldsymbol{\mu}, L)$ -Filling Problem (FP): Let $\mathcal{B} = \{\mathbf{b}_1, \mathbf{b}_2, \dots, \mathbf{b}_{|\mathcal{B}|}\}$ be a basis containing all $\{0, 1\}$ -vectors of length N_t with exactly L 1's. Find a set of non-negative elements $\alpha_1, \alpha_2, \dots, \alpha_{|\mathcal{B}|} \in \mathbb{R}^+$ such that $\sum_{f=1}^{|\mathcal{B}|} \alpha_f \mathbf{b}_f = \boldsymbol{\mu}$.⁵ We are interested in a solution to the $(\boldsymbol{\mu}^*, L)$ -FP, because we can define $\mathcal{M}_1, \mathcal{M}_2, \dots, \mathcal{M}_{|\mathcal{B}|}$ where \mathcal{M}_f contains an α_f fraction of the rows. Then, we assign \mathcal{M}_f to machines $\mathcal{P}_f = \{n : \mathbf{b}_f[n] = 1\}$ defined by the indices of the non-zero elements of \mathbf{b}_f . This precisely yields a computation load of $\boldsymbol{\mu}^*$ and each computation is assigned to L machines. Note that, we ensure each row is assigned since $L \sum_f \alpha_f = \sum_n \mu^*[n] = L$ and the elements of $\alpha_1, \alpha_2, \dots, \alpha_{|\mathcal{B}|}$ sum to 1. Also, $\mathcal{M}_1, \dots, \mathcal{M}_{|\mathcal{B}|}$ will span all rows if we define these sets to be disjoint. The question remains whether a $(\boldsymbol{\mu}^*, L)$ -FP solution exists for general $\boldsymbol{\mu}^*$ from the computation load optimization problem. In particular, we refer to the following lemma (Theorem 2 in [32]).

Lemma 1: Given $\boldsymbol{\mu}^* \in \mathbb{R}_+^{N_t}$ and $L \in \mathbb{Z}^+$, a $(\boldsymbol{\mu}^*, L)$ -FP solution exists *if and only if*

$$\mu^*[n] \leq \frac{\sum_{i=1}^{N_t} \mu^*[i]}{L} \quad (24)$$

for all $n \in [N_t]$. \square

There are two important consequences of Lemma 1. First, in our problem setting, we have $\sum_{i=1}^{N_t} \mu^*[i] = L$ and $\mu^*[n] \leq 1$ for all $n \in [N_t]$. Therefore, by using Lemma 1, an optimal computation assignment exists. Second, Lemma 1 provides a guide to develop an iterative algorithm to find such an assignment. In other words, if we define an iteration to assign a computation to a set of L machines, then effectively we create a new FP after each iteration. We can design each iteration to ensure the conditions of Lemma 1 are satisfied. Moreover, by adapting Algorithm 1 in [32], we obtain an equivalent Algorithm 1 (see pseudo-codes of Algorithm 1 for detailed operations) to explicitly provide an optimal computation assignment, $(\mathcal{M}_t, \mathcal{P}_t)$ where machine n stores and performs computations on its stored cs-matrix $\tilde{\mathbf{X}}_n$.

Remark 6: In line 10 of Algorithm 1, we consider the conditions of Lemma 1 to ensure Algorithm 1 yields a computation assignment with computation load $\boldsymbol{\mu}^*$. In lines 14 through 16, we update \mathbf{m} which tracks the remaining amount of assignments for each machine after the f -th iteration. \mathbf{m} must satisfy the conditions of Lemma 1 so that a (\mathbf{m}, L) -FP exists. After line 16, the machine with the most remaining computations to be assigned could be machine

⁴Note that, as in the optimization problem of (5), the computation load of the preempted machines are ignored since they are simply 0, presenting a slight difference between the optimal computation load vectors presented in Section III.

⁵ ℓ is an N' -length vector and $0 < m[\ell[1]] \leq m[\ell[2]] \leq \dots \leq m[\ell[N']]$.

⁶This is the condition obtained by using Lemma 1.

f	α_f	$m[1]$	$m[2]$	$m[3]$	$m[4]$	$m[5]$	$m[6]$	L'
1	$2/5$	$2/5$	$2/5$	$3/5$	0	$4/5$	$4/5$	3
2	$1/5$	0	$2/5$	$3/5$	\vdots	$2/5$	$2/5$	$9/5$
3	$1/5$	\vdots	$1/5$	$2/5$	\vdots	$2/5$	$1/5$	$6/5$
4	$1/5$	\vdots	0	$1/5$	\vdots	$1/5$	$1/5$	$3/5$
-		\downarrow	\downarrow	0	\downarrow	0	0	0

Fig. 4. Computation assignment following Algorithm 1 for Example 1 at time $t = 2$. Here, $N = 6$, $L = 3$, $F = 4$. f is the iteration index; Each row corresponds to an iteration f , where α_f is to the fraction of rows assigned to row set \mathcal{M}_f ; $m[n]$ denotes the remaining computation load for machine n at iteration f . The three red arrows from the first row to the second row represent that a fraction of $\alpha_1 = \frac{2}{5}$ rows are assigned to \mathcal{M}_1 , which are computed by machines (or equivalently, cs-matrices) $\mathcal{P}_1 = \{1, 5, 6\}$. L' represents the remaining total computation load at iteration f . At $f = 1$, $L' = L = 3$. After one iteration, we have $L' = 3 - 3 \cdot \alpha_1 = \frac{9}{5}$. Note that α_f is determined by lines 9 to 12 of Algorithm 1. At each iteration f , \mathcal{P}_f includes the machine with the smallest remaining $m[n]$ and $L - 1 = 2$ machines with the largest remaining $m[n]$.

$\ell[N' - L + 1]$. This machine had the most computations to be assigned before the f -th iteration but is not assigned any computations during the f -th iteration. After the f -th iteration, we must ensure

$$m[\ell[N' - L + 1]] \stackrel{(a)}{\leq} \frac{\sum_{i=1}^{N_t} m[i]}{L} \stackrel{(b)}{=} \frac{L' - \alpha_f L}{L} = \frac{L'}{L} - \alpha_f. \quad (25)$$

Here, (a) ensures the conditions of Lemma 1. Also, (b) holds because L' is the sum of m before the f -th iteration and the total computations to be assigned is reduced by $\alpha_f L$ after the f -th iteration. From (25) we find $\alpha_f \leq \frac{L'}{L} - m[\ell[N' - L + 1]]$ which is enforced in line 10.

Remark 7: Using a similar approach in the proof of Lemma 2 in [32], we can show that $F \leq N_t$ such that Algorithm 1 needs at most N_t iterations to complete.

B. An Example of Algorithm 1 for Homogeneous Storage and Heterogeneous Computing Speed

We return to Example 1 presented in Sections III-A and IV-D and use Algorithm 1 to derive the computation (rows in each element of \mathcal{M}_t) assignments for $t = 2$, where machine 4 is preempted. The steps of the algorithm are shown in Fig. 4. In this case, we showed that $\mu^* = [\frac{2}{5}, \frac{2}{5}, \frac{3}{5}, \frac{4}{5}, \frac{4}{5}]$ in Section IV-D. In the first iteration, $f = 1$, we have $L' = L$ and $m = \mu^*$ as no computations have been assigned yet. Rows of the respective cs-matrices are assigned to machine 1, 5, and 6 because among all machines, machine 1 has the least remaining computations to be assigned, and machines 5 and 6 have with the most remaining computations to be assigned. Note that

$$m[1] = \frac{2}{5} \leq \frac{L'}{L} - m[3] = 1 - \frac{3}{5} = \frac{2}{5}, \quad (26)$$

where machine 3 is the machine with the most remaining rows to be assigned that is not included in $\mathcal{P}_1 = \{1, 5, 6\}$. Therefore, a $\alpha_1 = \frac{2}{5}$ fraction of the rows are assigned to machines 1, 5, 6. Then, m is adjusted to reflect the remaining computations to be assigned and $L' = 3 - 3\alpha_1 = \frac{9}{5}$. For iteration 2, the condition of (26) relating to line 10 of Algorithm 1 is motivated by the necessary and sufficient conditions for the existence of a (m, L) -FP solution given in Lemma 1. Here, we are interested in the FP solution for m which is updated after each iteration. In other words, for each iteration of a row set assignment in Algorithm 1, we ensure there exists a set of succeeding row assignments such that a final FP solution is obtained.

In the second iteration, $f = 2$, machine 2 is a machine with the least remaining rows to be assigned. Computations are assigned to machine 2 and machines 3 and 6 which are a pair of machines with the most remaining computations to be assigned. Ideally, we would like to assign all the remaining rows to machine 2. However, since

$$m[2] = \frac{2}{5} > \frac{L'}{L} - m[5] = \frac{3}{5} - \frac{2}{5} = \frac{1}{5}, \quad (27)$$

assigning $\frac{2}{5}$ of the rows to machine 2 in this iteration will violate the condition of (24) in Lemma 1 and as a consequence, there will be no valid filling solution going forward. Therefore, we set $\alpha_2 = \frac{1}{5}$ instead and after this iteration m and L' are adjusted accordingly.

In the third iteration, $f = 3$, since

$$m[2] = \frac{1}{5} \leq \frac{L'}{L} - m[6] = \frac{2}{5} - \frac{1}{5} = \frac{1}{5}, \quad (28)$$

we assign $\alpha_3 = \frac{1}{5}$ of the rows to machines 2, 3, 5 and m and L' are adjusted accordingly. Finally, in the fourth iteration, $f = 4$, only three machines 3, 5, 6 have non-zero computation assignment left and each is assigned $\alpha_4 = \frac{1}{5}$ of the rows. In this example, the algorithm completes in $F = 4$ (fewer than $N_2 = 5$) iterations. The resulting computing assignment is shown in Fig. 1(b).

C. Heterogeneous Storage Capacity and Computing Speeds

When both storage capacity and computing speeds among machines are heterogeneous, machines may store more than cs-matrices. In this case, machine n will pick $\lfloor \mu^*[n] \rfloor$ cs-matrices to compute entirely. Then, it will pick a remaining cs-matrix and compute a $\mu^*[n] - \lfloor \mu^*[n] \rfloor$ fraction of that cs-matrix. We will show this strategy requires $F \leq N_t$ iterations using Algorithm 2 and the number of computation assignments F is at most equal to the number of available machines N_t . Overall, the assignment consists of two steps. In the first step, those cs-matrices that are computed entirely are put into in the cs-matrix sets of \mathcal{P}_t . In the second step, we assign row sets to the cs-matrices which are not entirely computed so that each row set in \mathcal{M}_t is guaranteed to be computed across L cs-matrices. Next, we demonstrate that we can re-use Algorithm 1 for the second step of the computation assignment under a modified procedure described in Algorithm 2.

To explain the computation assignment process we introduce the following notations. For $n \in [N_t]$, let $\tilde{Q}_n \subseteq Q_n$ contain the indices of $\lfloor \mu^*[n] \rfloor$ arbitrarily chosen cs-matrices in Q_n that machine n computes entirely. Note that $|\tilde{Q}_n| = \lfloor \mu^*[n] \rfloor$. If $\mu[n] < 1$, then \tilde{Q}_n is empty. Next, machine n arbitrarily chooses one cs-matrix from $Q_n \setminus \tilde{Q}_n$ to compute partially which we label as $\hat{\theta}[n]$. Note that when $\mu[n]$ is an integer, $\hat{\theta}[n]$ is simply a dummy variable and is never referenced, i.e. $\hat{\theta}[n] = \emptyset$. In the following, we denote $\hat{\theta} = (\hat{\theta}[1], \hat{\theta}[2], \dots, \hat{\theta}[N_t])$ as the cs-matrices partially computed by the machines. Then we define the partial computation vector $\hat{\mu} \in \mathbb{R}_+^{N_t}$ such that

$$\hat{\mu}[n] = \mu^*[n] - \lfloor \mu^*[n] \rfloor, \quad \forall n \in [N_t]. \quad (29)$$

Hence, machine n will entirely compute each cs-matrix \tilde{X}_i for $i \in \tilde{Q}_n$ and compute a $\hat{\mu}[n]$ fraction of the cs-matrix $\tilde{X}_{\hat{\theta}[n]}$.

Finally, we define the sum of the partial computation load vector $\hat{\mu}$ as

$$\hat{L} \triangleq \sum_{n=1}^{N_t} \hat{\mu}[n]. \quad (30)$$

Note that \hat{L} is an important parameter because it represents the number of cs-matrices that each row set needs to be assigned to, excluding those cs-matrices that are entirely computed. In other words, since the elements in μ sum to L , there are $L - \hat{L}$ cs-matrices that are entirely computed by the machines. Therefore, in order to assign each row computation (or row set) in \mathcal{M}_t to L cs-matrices, we assign each row set to \hat{L} cs-matrices that are only partially computed. The detailed description of the proposed algorithm is given in Algorithm 2.

In Algorithm 2, we will perform Algorithm 1 on $\hat{\mu}$, which iteratively fills some computations at \hat{L} machines in each iteration. Following similar arguments as before, we need to ensure that such a filling problem solution exists and can be found using the proposed algorithm. From Lemma 1, we see that a $(\hat{\mu}, \hat{L})$ -FP solution exists because $\hat{\mu}[n] < 1$ for all $n \in [N_t]$ and $\hat{L} = \sum_{n=1}^{N_t} \hat{\mu}[n]$. Then, similar to the previous analysis, it can be seen that Algorithm 1 will yield a $(\hat{\mu}, \hat{L})$ -FP solution with $F \leq N_t$ iterations. This means that, in order to use Algorithm 1, instead of inputting μ^* and L , we input $\hat{\mu}$ and \hat{L} . Then, we label the output of Algorithm 1 as \hat{F} , $\hat{\mathcal{M}}_1, \dots, \hat{\mathcal{M}}_F$ and $\hat{\mathcal{P}}_1, \dots, \hat{\mathcal{P}}_F$. These variables represent the computation assignments at the cs-matrices that are partially computed, but computations are only assigned to \hat{L} , instead of L , cs-matrices. Note that, due to the one-to-one mapping between partially computed cs-matrices and machines, each $\hat{\mathcal{P}}_f$ represents the set of machines that are assigned to compute rows in \mathcal{M}_f of the cs-matrix it is partially computing. To complete the computation assignment, we must include the $L - \hat{L}$ cs-matrices that are entirely computed. Therefore,

$$\mathcal{P}_f = \bigcup_{n \in \hat{\mathcal{P}}_f} \hat{\theta}[n] \cup \bigcup_{m \in [N_t]} \tilde{Q}_m, \quad \forall f \in [F], \quad (31)$$

where $|\bigcup_{n \in \hat{\mathcal{P}}_f} \hat{\theta}[n]| = \hat{L}$ and $|\bigcup_{m \in [N_t]} \tilde{Q}_m| = L - \hat{L}$.

Since Algorithm 1 assigns computations to machines,⁵ $\hat{\theta}$ is needed to identify the cs-matrices that the machines partially

⁵Algorithm 1 is designed with the assumption that machine $n, \forall n \in [N]$ stores one cs-matrix, \tilde{X}_n .

compute. In addition, the number of computation assignments remains the same and $\hat{F} = F$. The row sets also remain the same, $\mathcal{M}_f = \hat{\mathcal{M}}_f$ for all $f \in [F]$.

Algorithm 2 will be illustrated using the following example.

D. An Example of Algorithm 2 for Heterogeneous Storage and Computing Speed

We consider Example 2 presented in Sections III-B and IV-D, where we have $L = 6$ and $\sigma = [2, 2, 2, 1, 1, 1]$. This means that machines 1, 2, and 3 stores two cs-matrices while machines 4, 5, and 6 stores one cs-matrix, respectively. We assume no preempted machines at $t = 1$. In this case, the optimal computation load vector is found to be $\mu^* = [\frac{8}{11}, \frac{12}{11}, \frac{16}{11}, \frac{8}{11}, 1, 1]$ in Section IV-D. Since $\mu^*[n] \geq 1, n \in \{2, 3, 5, 6\}$, it can be seen that machines 2, 3, 5, and 6 will compute all the rows sets of \mathcal{M}_1 for one cs-matrix (see Fig. 2). Next, machines 1, 2, 3 and 4 have one cs-matrix to be partially computed, and each of them will compute a fraction of that cs-matrix. Note that, since $\mu^*[5] = 1$ and $\mu^*[6] = 1$ are integers, by the algorithm design, no computations will be assigned to cs-matrices partially computed by machines 5 and 6. In other words, based on the optimal computation load vector μ^* , machines 5 and 6 only entirely compute cs-matrices. Using (29), we can obtain the partial computation load vector as $\hat{\mu} = [\frac{8}{11}, \frac{1}{11}, \frac{5}{11}, \frac{8}{11}, 0, 0]$, whose elements sum to $\hat{L} = 2$. Our goal is to assign computations to cs-matrices partially computed by machines 1 through 4, where we assign the computations corresponding to each row set of \mathcal{M}_1 to $\hat{L} = 2$ cs-matrices at a time. This will be done using Algorithm 1.

In particular, let the indexes of the cs-matrices stored at the machines be $Q_1 = \{1, 2\}$, $Q_2 = \{3, 4\}$, $Q_3 = \{5, 6\}$, $Q_4 = \{7\}$, $Q_5 = \{8\}$ and $Q_6 = \{9\}$. Each machine n picks a set of $\lfloor \mu[n] \rfloor$ stored cs-matrices to be computed entirely which could be that of $\tilde{Q}_1 = \emptyset$, $\tilde{Q}_2 = \{3\}$, $\tilde{Q}_3 = \{5\}$, $\tilde{Q}_4 = \emptyset$, $\tilde{Q}_5 = \{8\}$ and $\tilde{Q}_6 = \{9\}$. Moreover, each machine selects an index of a stored cs-matrix to be partially computed, which are denoted as $\hat{\theta} = [1, 4, 6, 7, 0, 0]$.

In the first iteration of Algorithm 1 inside Algorithm 2 (line 5), we aim to assign some computations to the cs-matrix $\tilde{X}_{\hat{\theta}[2]}$, since $\hat{\mu}[2]$ is the smallest non-zero element in $\hat{\mu}$. $\tilde{X}_{\hat{\theta}[2]}$ will be partially computed by machine 2. We also assign this computation to machine 4 because it is a machine with the largest remaining computations to be assigned (line 8 in Algorithm 1). Therefore, we assign a $\alpha_1 = \frac{1}{11}$ fraction of rows to the cs-matrices partially computed by machines 2 and 4 (line 10 in Algorithm 1). After this iteration $\mathbf{m} = [\frac{8}{11}, 0, \frac{5}{11}, \frac{7}{11}, 0, 0]$ (line 15 in Algorithm 1), and machines 1 and 3 are the machines with the most and least, respectively, remaining computations to be assigned. From line 10 in Algorithm 1, since

$$m[3] = \frac{5}{11} > \frac{3}{11} = \frac{\hat{L}'}{\hat{L}} - m[4], \quad (32)$$

we assign a $\alpha_2 = \frac{3}{11}$ fraction of rows to machines 1 and 3. Then, after this iteration, we find $\mathbf{m} = [\frac{5}{11}, 0, \frac{2}{11}, \frac{7}{11}, 0, 0]$. By a similar approach, next we

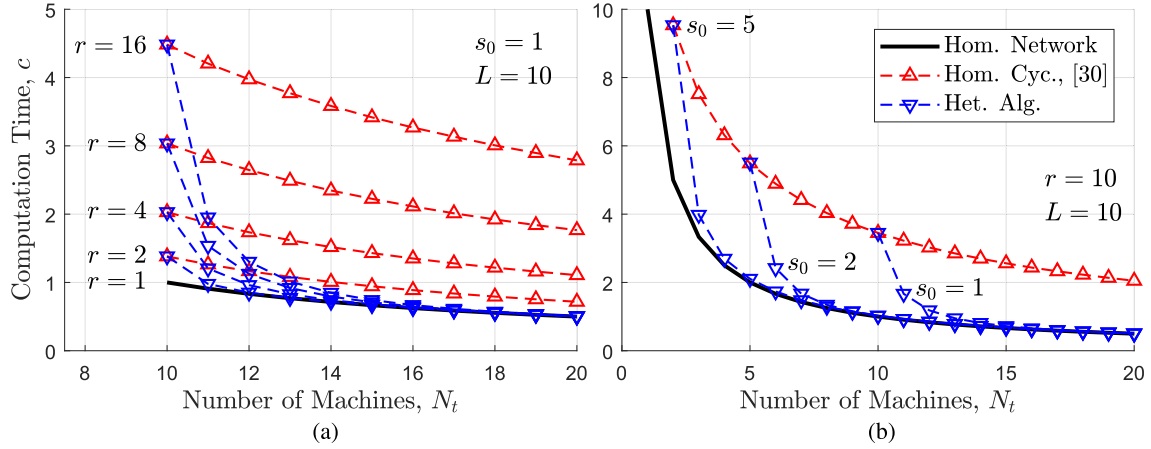


Fig. 5. The simulated mean computation time using the Homogeneous Cyclic design [30] (red curves) and the proposed Heterogeneous Algorithm (blue curves) on heterogeneous networks. The machine speeds are sampled from a uniform distribution with mean 1. The results are compared to the computation time of an equivalent homogeneous network where machines all have a speed of 1. In (a) and (b), the network heterogeneity, r , and storage constraint, s_0 , respectively, are varied to produce multiple curves.

assign a $\alpha_3 = \frac{2}{11}$ fraction of rows to machines 3 and 4 and a $\alpha_4 = \frac{5}{11}$ fraction of rows to machines 1 and 4. After this iteration, we can find that $\mathbf{m} = \mathbf{0}$.

Based on above procedure in Algorithm 1, we obtain the output \hat{F} , $\hat{\mathcal{M}}_1, \dots, \hat{\mathcal{M}}_4$ which contain a $\frac{1}{11}$, $\frac{3}{11}$, $\frac{2}{11}$ and $\frac{5}{11}$ fraction of rows, respectively, and the machines assigned to compute row sets \mathcal{M}_f are given by $\hat{\mathcal{P}}_1 = \{2, 4\}$, $\hat{\mathcal{P}}_2 = \{1, 3\}$, $\hat{\mathcal{P}}_3 = \{3, 4\}$ and $\hat{\mathcal{P}}_4 = \{1, 4\}$. For the final solution, the number of assignments stays the same, $F = \hat{F} = 4$, and the row sets stay the same $\mathcal{M}_f = \hat{\mathcal{M}}_f$, $\forall f \in [F]$. However, using $\hat{\mathcal{P}}_f$, $\forall f \in [F]$, we need to define specifically which cs-matrices are being computed for each row set. Note that, $\hat{\mathcal{P}}_f$ is the set of machines that are assigned to compute rows in \mathcal{M}_f of the corresponding cs-matrix partially computed. We use $\hat{\theta}$ to resolve the indexes of the cs-matrices from $\hat{\mathcal{P}}_f$. Then, we also need to include the indexes of all cs-matrices that are entirely computed from $\hat{\mathcal{Q}}_1, \dots, \hat{\mathcal{Q}}_6$. For example, recall that, $\hat{\mathcal{Q}}_1 = \emptyset$, $\hat{\mathcal{Q}}_2 = \{3\}$, $\hat{\mathcal{Q}}_3 = \{5\}$, $\hat{\mathcal{Q}}_4 = \emptyset$, $\hat{\mathcal{Q}}_5 = \{8\}$ and $\hat{\mathcal{Q}}_6 = \{9\}$, we then obtain the cs-matrix sets

$$\mathcal{P}_1 = \{\hat{\theta}[2], \hat{\theta}[4], 3, 5, 8, 9\} = \{3, 4, 5, 7, 8, 9\}. \quad (33)$$

Similarly, we see that $\mathcal{P}_2 = \{1, 3, 5, 6, 8, 9\}$, $\mathcal{P}_3 = \{3, 5, 6, 7, 8, 9\}$ and $\mathcal{P}_4 = \{1, 3, 5, 7, 8, 9\}$.

VI. RESULTS SUMMARY

In this section, we summarize the results of this work and present simulation studies to evaluate the performance of the new heterogeneous CEC design.

Theorem 2: An optimal computation assignment that solves (4) is found by using Algorithm 2 to yield an assignment satisfying the computation load vector $\boldsymbol{\mu}^*$ from Theorem 1.

Proof: Algorithm 2 produces a computation assignment that yields the computation load vector $\boldsymbol{\mu}^*$ and computation time \hat{c}^* , the solution of (5). The optimization problem of (5) is a relaxed version of (4). Therefore, this computation

assignment from Algorithm 2 is a solution to the optimization problem of (4). ■

A. Simulations

Machine computation speeds are sampled from a uniform distribution with mean 1. The variable r defines the ratio between the upper and lower bounds of the distribution and is a quantitative measure of the network heterogeneity. For example, if $r = 2$, then machine speeds are sampled from a uniform distribution ranging from $\frac{2}{3}$ to $\frac{4}{3}$. This distribution has a mean of 1. Furthermore, we define s_0 as the number of cs-matrices that each machine stores.

We perform simulations for various values of r , s_0 , and N_t where $L = 10$ is fixed. In each configuration, the mean computation time of the Homogeneous Cyclic [30] (red curves) and the proposed Heterogeneous Algorithm (blue lines) from 10^5 samples are shown in Fig. 5. The results are compared to those of a purely homogeneous network (black curves, $r = 1$) where each machine has a computation speed of 1.

In Fig. 5(a), each machine has the same storage constraint and stores $s_0 = 1$ cs-matrix. Then, for different heterogeneity measures, r , we vary the number of available machines, N_t from 10 to 20. Focusing on when $N_t = L = 10$, we notice the proposed heterogeneous design and the homogeneous cyclic design [30] have the same performance. This occurs because the number of cs-matrices among the available machines is exactly L and both designs produce the same computation assignment which is that each machine computes the entirety of their local cs-matrix. Also, the computation time, c , increases with r because c equals the time it takes the slowest machine to compute the entirety of its cs-matrix. For both designs, we observe a decrease in c as N_t increases since the computations are spread across more machines, reducing the computation load of any single machine. Note that when $N_t > L$, the proposed heterogeneous CEC design outperforms the homogeneous CEC design [30] because the former spreads the computation load relative to the computation speed of each

available machine. Moreover, c for the proposed heterogeneous CEC design converges to that of a homogeneous network with equivalent cumulative computation speed.

In Fig. 5(b), we fix $r = 10$ and assume that each machine stores s_0 cs-matrices. First, we vary s_0 to produce multiple blue curves for the proposed heterogeneous design. Second, for the homogeneous design [30], only one red curve corresponding to $s_0 = 5$ is presented. Due to the constraint that $Z_t = s_0 N_t \geq L$, choosing the larger $s_0 = 5$ allows the homogeneous design to operate on a wider range of N_t . We also note that for a given N_t that satisfies $Z_t = s_0 N_t \geq L$, the c of the homogeneous design is independent of s_0 because all machines are assigned an equal computation load. In other words, for a given N_t , varying s_0 will not change c . This justifies why only one red curve is shown. In contrast, for the proposed heterogeneous CEC design, when s_0 increases, a smaller c can be achieved. For example, when $N_t = 5$, we have $c \approx 5.5$ for $s_0 = 2$ and $c \approx 2.1$ for $s_0 = 5$. This occurs because s_0 is the upper bound on the computation load for each machine. When s_0 is larger, it is less likely that faster machines reach this upper bound. The faster machines can take on more of the computation load and alleviate the computation load of some slower machines that may be affecting the computation time c . Finally, again we observe that the performance of the proposed heterogeneous CEC converges to that of a homogeneous network with equivalent cumulative computation speed.

VII. CONCLUSION

In this paper, we study the heterogeneous coded elastic computing problem where computing machines store MDS coded data matrices and may have both varying computation speeds and storage capacity. The key of this problem is to design a fixed storage assignment scheme and a computation assignment strategy such that no redundant computations are present and the overall computation time can be minimized as long as there are at least L cs-matrices stored among the available machines. Given a set of available machines with arbitrary relative computation speeds and storage capacity, we first proposed a novel combinatorial min-max problem formulation in order to minimize the overall computation time, which is determined by the machines that need the longest computation time. Based on the MDS coded storage assignment, the goal of this optimization problem is to assign computation tasks to machines such that the overall computation time is minimized. In order to precisely solve this combinatorial problem, we decompose it into a convex optimization problem to determine the optimal computation load of each machine and a computation assignment problem that yields the resulting computation load from the convex optimization problem. Then, we adapt low-complexity iterative algorithms to find the optimal solution to the original combinatorial problems, which require a number of iterations no greater than the number of available machines. The proposed heterogeneous coded elastic computing design has the potential to perform computations faster than the state-of-the-art design which was developed for a homogeneous distributed computing system.

REFERENCES

- [1] N. Woolsey, R.-R. Chen, and M. Ji, "Heterogeneous computation assignments in coded elastic computing," 2020, *arXiv:2001.04005*. [Online]. Available: <http://arxiv.org/abs/2001.04005>
- [2] S. Li, M. A. Maddah-Ali, Q. Yu, and A. S. Avestimehr, "A fundamental tradeoff between computation and communication in distributed computing," *IEEE Trans. Inf. Theory*, vol. 64, no. 1, pp. 109–128, Jan. 2018.
- [3] S. Li, M. A. Maddah-Ali, and A. S. Avestimehr, "Compressed coded distributed computing," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Jun. 2018, pp. 2032–2036.
- [4] K. Konstantinidis and A. Ramamoorthy, "Leveraging coding techniques for speeding up distributed computing," 2018, *arXiv:1802.03049*. [Online]. Available: <http://arxiv.org/abs/1802.03049>
- [5] N. Woolsey, R.-R. Chen, and M. Ji, "A new combinatorial design of coded distributed computing," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Jun. 2018, pp. 726–730.
- [6] S. R. Srinivasavaradhan, L. Song, and C. Fragouli, "Distributed computing trade-offs with random connectivity," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Jun. 2018, pp. 1281–1285.
- [7] S. Prakash, A. Reisizadeh, R. Pedarsani, and A. S. Avestimehr, "Coded computing for distributed graph analytics," 2018, *arXiv:1801.05522*. [Online]. Available: <http://arxiv.org/abs/1801.05522>
- [8] N. Woolsey, R.-R. Chen, and M. Ji, "Cascaded coded distributed computing on heterogeneous networks," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Jul. 2019, pp. 2644–2648.
- [9] F. Xu and M. Tao, "Heterogeneous coded distributed computing: Joint design of file allocation and function assignment," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Dec. 2019, pp. 1–6.
- [10] N. Woolsey, R.-R. Chen, and M. Ji, "Coded distributed computing with heterogeneous function assignments," 2019, *arXiv:1902.10738*. [Online]. Available: <http://arxiv.org/abs/1902.10738>
- [11] K. Wan, M. Ji, and G. Caire, "Topological coded distributed computing," 2020, *arXiv:2004.04421*. [Online]. Available: <http://arxiv.org/abs/2004.04421>
- [12] M. Adel Attia and R. Tandon, "Near optimal coded data shuffling for distributed learning," *IEEE Trans. Inf. Theory*, vol. 65, no. 11, pp. 7325–7349, Nov. 2019.
- [13] A. Elmahdy and S. Mohajer, "On the fundamental limits of coded data shuffling," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Jun. 2018, pp. 716–720.
- [14] K. Wan, D. Tuninetti, M. Ji, G. Caire, and P. Piantanida, "Fundamental limits of decentralized data shuffling," *IEEE Trans. Inf. Theory*, vol. 66, no. 6, pp. 3616–3637, Jun. 2020.
- [15] K. Lee, M. Lam, R. Pedarsani, D. Papailiopoulos, and K. Ramchandran, "Speeding up distributed machine learning using codes," *IEEE Trans. Inf. Theory*, vol. 64, no. 3, pp. 1514–1529, Mar. 2018.
- [16] R. Tandon, Q. Lei, A. G. Dimakis, and N. Karampatziakis, "Gradient coding: Avoiding stragglers in distributed learning," in *Proc. Int. Conf. Mach. Learn.*, 2017, pp. 3368–3376.
- [17] S. Dutta, V. Cadambe, and P. Grover, "Short-dot: Computing large linear transforms distributedly using coded short dot products," in *Proc. Adv. Neural Inf. Process. Syst.*, 2016, pp. 2100–2108.
- [18] K. Wan, H. Sun, M. Ji, and G. Caire, "Distributed linearly separable computation," 2020, *arXiv:2007.00345*. [Online]. Available: <http://arxiv.org/abs/2007.00345>
- [19] Q. Yu, M. A. Maddah-Ali, and A. S. Avestimehr, "Straggler mitigation in distributed matrix multiplication: Fundamental limits and optimal coding," *IEEE Trans. Inf. Theory*, vol. 66, no. 3, pp. 1920–1933, Mar. 2020.
- [20] S. Dutta, M. Fahim, F. Haddadpour, H. Jeong, V. Cadambe, and P. Grover, "On the optimal recovery threshold of coded matrix multiplication," *IEEE Trans. Inf. Theory*, vol. 66, no. 1, pp. 278–301, Jan. 2020.
- [21] K. Wan, H. Sun, M. Ji, D. Tuninetti, and G. Caire, "Cache-aided matrix multiplication retrieval," 2020, *arXiv:2007.00856*. [Online]. Available: <http://arxiv.org/abs/2007.00856>
- [22] C. Karakus, Y. Sun, S. Diggavi, and W. Yin, "Straggler mitigation in distributed optimization through data encoding," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 5434–5442.
- [23] R. Bitar, P. Parag, and S. El Rouayheb, "Minimizing latency for secure distributed computing," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Jun. 2017, pp. 2900–2904.
- [24] M. Aliasgari, O. Simeone, and J. Klierer, "Private and secure distributed matrix multiplication with flexible communication load," *IEEE Trans. Inf. Forensics Security*, vol. 15, pp. 2722–2734, 2020.
- [25] H. Sun and S. A. Jafar, "The capacity of private computation," *IEEE Trans. Inf. Theory*, vol. 65, no. 6, pp. 3880–3897, Jun. 2019.

- [26] S. A. Obead, H.-Y. Lin, E. Rosnes, and J. Klierer, "Private function computation for noncolluding coded databases," 2020, *arXiv:2003.10007*. [Online]. Available: <http://arxiv.org/abs/2003.10007>
- [27] Z. Chen, Z. Jia, Z. Wang, and S. A. Jafar, "GCSA codes with noise alignment for secure coded multi-party batch matrix multiplication," 2020, *arXiv:2002.07750*. [Online]. Available: <http://arxiv.org/abs/2002.07750>
- [28] L. Chen, H. Wang, Z. Charles, and D. Papailiopoulos, "Draco: Byzantine-resilient distributed training via redundant gradients," in *Proc. Int. Conf. Mach. Learn.*, 2018, pp. 902–911.
- [29] E. Jonas, Q. Pu, S. Venkataraman, I. Stoica, and B. Recht, "Occupy the cloud: Distributed computing for the 99%," in *Proc. Symp. Cloud Comput.*, Sep. 2017, pp. 445–451.
- [30] Y. Yang, M. Interlandi, P. Grover, S. Kar, S. Amizadeh, and M. Weimer, "Coded elastic computing," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Jul. 2019, pp. 2654–2658.
- [31] H. Dau *et al.*, "Optimizing the transition waste in coded elastic computing," 2019, *arXiv:1910.00796*. [Online]. Available: <http://arxiv.org/abs/1910.00796>
- [32] N. Woolsey, R.-R. Chen, and M. Ji, "An optimal iterative placement algorithm for PIR from heterogeneous storage-constrained databases," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Dec. 2019, pp. 1–6.
- [33] Y. Yang, M. Interlandi, P. Grover, S. Kar, S. Amizadeh, and M. Weimer, "Coded elastic computing," 2018, *arXiv:1812.06411*. [Online]. Available: <http://arxiv.org/abs/1812.06411>



Nicholas Woolsey (Student Member, IEEE) received the B.S. degree in biomedical engineering from the University of Connecticut in 2012 and the M.Eng. degree in bioengineering from the University of Maryland, College Park, MD, USA, in 2015, with a focus on signal processing, imaging and optics. He is currently pursuing the Ph.D. degree with the Department of Electrical and Computer Engineering, University of Utah. From 2014 to 2017, he was an Electrical Engineer with Northrop Grumman Corporation (NGC), Ogden, UT, USA, developing test and

evaluation methods, modernization solutions, and signal processing algorithms for the sustainment of aging aircraft and ground communication systems. He is currently with Trabus Technologies, San Diego, CA, USA. While at NGC, he received the "Outside the Box" Grant to investigate the design of a modern receiver that interfaces aging technology and the 2016 Brent Scowcroft Team Award for performing exceptional systems engineering work. His research interests include combinatorial designs and algorithms for resource allocation, coding, and efficient communications in distributed computing, private and caching networks.



Rong-Rong Chen (Member, IEEE) received the B.S. degree in applied mathematics from Tsinghua University, China, in 1993, and the M.S. degree in mathematics and the Ph.D. degree in electrical and computer engineering from the University of Illinois at Urbana–Champaign, in 1995 and 2003, respectively. From 2003 to 2011, she was an Assistant Professor with the University of Utah. Since 2011, she has been an Associate Professor with the University of Utah. Her main research interests include communication systems and networks, with current emphasis on distributed computing, machine learning, caching networks, statistical signal processing, image reconstructions, and channel coding. She was a recipient of the M. E. Van Valkenburg Graduate Research Award for excellence in doctoral research with the Department of Electronics and Communication Engineering, University of Illinois at Urbana–Champaign, in 2003, and the prestigious National Science Foundation Faculty Early Career Development (CAREER) Award in 2006. She was rated among the Top 15% Instructors of the College of Engineering, University of Utah, in 2017 and 2018. She has served as an Associate Editor for IEEE TRANSACTIONS ON SIGNAL PROCESSING and a Guest Editor for IEEE JOURNAL ON SELECTED TOPICS IN SIGNAL PROCESSING. She has also served on the technical program committees for leading international conferences in wireless communication and networks.



Mingyue Ji (Member, IEEE) received the B.E. degree in communication engineering from the Beijing University of Posts and Telecommunications, China, in 2006, the M.Sc. degrees in electrical engineering from the Royal Institute of Technology, Sweden, and the University of California, Santa Cruz, CA, USA, in 2008 and 2010, respectively, and the Ph.D. degree from the Ming Hsieh Department of Electrical Engineering, University of Southern California, Los Angeles, CA, in 2015. He subsequently was a Staff II System Design Scientist with

Broadcom Corporation (Broadcom Ltd.) from 2015 to 2016. He is currently an Assistant Professor with the Department of Electrical and Computer Engineering and an Adjunct Assistant Professor with the School of Computing, University of Utah. He received the IEEE Communications Society Leonard G. Abraham Prize for the Best IEEE JSAC Paper in 2019, and the Best Paper Award from the IEEE ICC 2015 Conference, the IEEE European Wireless 2010 Conference, and the USC Annenberg Fellowship from 2010 to 2014. He is currently an Associate Editor of IEEE TRANSACTIONS ON COMMUNICATIONS. His research interests include information theory, coding theory, concentration of measure and statistics with the applications of caching networks, wireless communications, distributed storage and computing systems, federated learning, and (statistical) signal processing.