A New Design Framework for Heterogeneous Uncoded Storage Elastic Computing

Mingyue Ji[1], Xiang Zhang[1], and Kai Wan[2]
[1]University of Utah,
[2]Technische Universität Berlin
Email: {mingyue.ji@utah.edu, xiang.zhang@utah.edu, kai.wan@tu-berlin.de}

***Abstract*—Elasticity is one important feature in modern cloud computing systems and can result in computation failure or significantly increase computing time. Such elasticity means that virtual machines over the cloud can be preempted under a short notice (e.g., hours or minutes) if a high-priority job appears; on the other hand, new virtual machines may become available over time to compensate the computing resources. Coded Storage Elastic Computing (CSEC) introduced by Yang et al. in 2018 is an effective and efficient approach to overcome the elasticity and it costs relatively less storage and computation load. However, one of the limitations of the CSEC is that it may only be applied to certain types of computations (e.g., linear) and may be challenging to be applied to more involved computations because the coded data storage and approximation are often needed. Hence, it may be preferred to use uncoded storage by directly copying data into the virtual machines. In addition, based on our own measurement, virtual machines on Amazon EC2 clusters often have heterogeneous computation speed even if they have exactly the same configurations (e.g., CPU, RAM, I/O cost). In this paper, we introduce a new optimization framework on Uncoded Storage Elastic Computing (USEC) systems with heterogeneous computing speed to minimize the overall computation time. Under this framework, we propose optimal solutions of USEC systems with or without straggler tolerance using different storage placements. Our proposed algorithms are evaluated using power iteration applications on Amazon EC2.**

I. Introduction

Coded Storage Elastic Computing (CSEC) system introduced by Yang et al. in [1] is an effective approach to overcome the elasticity of modern cloud computing system, where elasticity means that Virtual Machines (VMs) on the cloud systems, e.g., instances on Amazon EC2, can be preempted under a short notice (e.g., hours or minutes) if a high-priority job appears; on the other hand, new VMs may become available over time to compensate the computing resources. Such elasticity can result in computation failure or significantly increase computing time.

In [1], using a Maximum Distance Separable (MDS) coded storage placement, the authors proposed a *cyclic* computation assignment scheme such that no redundant computation is needed when the number of available VMs N_t is between L and N where N is the maximum number of VMs in the systems and L is the smallest number of VMs in the system. In [2], the authors introduced a new metric, called transition waste, which is defined as the difference between the total number of changes and the number of necessary changes of the computation assignment if some VMs become preempted during one computation or time step. This problem is combinatorial and is challenging to be solved in general. The authors proposed new algorithms using *shifted cyclic* task allocation to reduce the transition waste and showed it is optimal under some parameter settings. In [3], the authors proposed two hierarchical schemes that can further speed up the CSEC system by effectively allocating tasks among available nodes while the encoding and decoding complexity may be increased. Some important limitations of [1]–[3] include the assumption that all available VMs have the same computing speed or the proposed schemes do not consider the heterogeneous computing speed among machines, and all VMs have the homogeneous storage constraint. In practice, based on our own measurement [4], the computing speed among VMs can be significantly different even if they have exactly the same configurations, e.g., same CPU, RAM and I/O cost. In addition, during the entire computing process (e.g., power iteration application [5] and see Section V), we observe that the computing speed among machines stays approximately the same, while the proposed algorithms in [4] takes measurements of the computing speeds of virtual machines very frequently, i.e., the time scale of computing speed measurement is much smaller than the overall computation time. Hence, we assume that the computing speed of all virtual machines do not change over one time step (see Section II) in [4] and in this paper. This is in contrast to the works that model the computing speeds/times as random variables for long term analysis (e.g., [6], [7]), which consider a very large number of virtual machines for applications that run over an extensive amount of time.

In [8], the authors considered the elastic computing systems with heterogeneous computing speed and homogeneous storage constraint, and formulated a new CSEC framework, that is to minimize the overall computation time, using a combinatorial optimization approach. In addition, one exact optimal solution is provided and can be achieved using the *filling algorithm*, which is a low-complexity iterative algorithm that can complete within N_t iterations, where N_t is the number of available VMs at time step t. Later, in [9], the authors considered the CSEC system with both heterogeneous computing speed and heterogeneous storage constraint, and formulated a new combinatorial optimization framework based on the result

ISBN 978-3-903176-49-2 © 2022 IFIP

in [8] and designed algorithms to achieve the optimal computation time. Under the assumption of heterogeneous computing speed, in [4], the authors made preliminary attempts to study the scenario where both elasticity and stragglers are present and proposed new algorithms using the idea of the filling algorithm.[1] An achievable trade-off between computation time and straggler tolerance was established. In addition, the authors in [4] implemented the proposed algorithms for heterogeneous CSEC systems using real applications on Amazon EC2 and demonstrated that large gain in terms of the computation time can be achieved by the proposed algorithms.

Despite clear advantages of the CSEC systems such as less storage overhead, it can only be applied to certain types of computations (e.g., linear) and may be challenging to be applied to more involved computations (e.g., deep learning) due to the coded data storage. In this case, approximation is often needed. Hence, it may be preferred to use uncoded storage by just copying the data into the virtual machines since computations can be operated directly over the original data in this case. We refer to such systems as Uncoded Storage Elastic Computing (USEC) systems. In this paper, we first introduce a new optimization framework on USEC with heterogeneous computing speed to minimize the overall computation time. Then, we propose optimal low-complexity solutions to USEC systems with or without straggler tolerance using different storage placements.

Our contributions are summarized as follows:

1) When there is no straggler tolerance requirement, given the storage placement and the heterogeneous computing speed of VMs, we formulate a new USEC framework as a convex optimization problem which can be solved using typical convex optimization solvers. Further, we investigate the performance in terms of computation time using different uncoded storage placements.
2) We incorporate straggler tolerance into the above problem formulation and formulate it as a combinatorial optimization problem. In addition, we design a low-complexity algorithm to achieve the optimal solution of the proposed optimization problem given the uncoded storage placement.
3) We perform experiments using the proposed USEC framework with heterogeneous computing speed, and using the power iteration application under a simple setup. We demonstrate that about 20% gain in terms of computation time can be achieved using the proposed algorithms by taking the advantage of heterogeneous computing speed.

The rest of this paper is organized as follows. Section II introduces the network model and presents the problem formulation. Section III presents some motivating examples to the proposed problem and illustrate our proposed solutions. In Section IV, we propose the new USEC design in general. Some experimental results over Amazon EC2 are illustrated in Section V.

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

II. Network Model and Problem Formulation

We consider a set of N virtual machines jointly store an uncoded data matrix $\mathbf{X}$ with dimension $q \times r$, which is row-wise partitioned as follows.

$$\mathbf{X} = [\mathbf{X}_1; \mathbf{X}_2; \cdots ; \mathbf{X}_G].$$

With a slight abuse of notation, $\mathbf{X}_g, g \in [G]$ denotes both the *row sets* and *sub-matrices* of $\mathbf{X}$. In particular, the number of rows in each $\mathbf{X}_g, g \in [G]$ is q/G and we index them as $[q/G]$. Each $\mathbf{X}_g$ is placed into J virtual machines. Let $\mathcal{N}_g = \{n : \mathbf{X}_g \in \mathcal{Z}_n\}$ denote the set of virtual machines that stores $\mathbf{X}_g$ and $\mathcal{Z}_n$ be the storage placement for VM n. The set of the storage placements for all virtual machines is denoted by $\boldsymbol{\mathcal{Z}} = \{\mathcal{Z}_n, n \in [N]\}$.

Similar to [1], the virtual machines collectively perform matrix-vector computations over multiple steps. In a given time step only a subset of the N VMs are available to perform matrix computations. More specifically, in computation step t, a set of available VMs $\mathcal{N}_t \subseteq [N]$ with $|\mathcal{N}_t| = N_t$ aims to compute

$$\boldsymbol{y}_t = \boldsymbol{X}\boldsymbol{w}_t, \tag{1}$$

where $\boldsymbol{w}_t$ is some vector of length r. The VMs of $[N] \setminus \mathcal{N}_t$ are preempted.

The VMs in $\mathcal{N}_t$ do not compute $\boldsymbol{y}_t$ directly. Instead, each machine $n \in \mathcal{N}_t$ computes $\mathbf{X}_{\mathcal{S}_n}\boldsymbol{w}_t$, where $\mathcal{S}_n \subset \mathbf{X}_g, \mathbf{X}_g \in \mathcal{Z}_n$ denotes a row set in the sub-matrix $\mathbf{X}_g \in \mathcal{Z}_n$. Then the results from VMs will be sent to the master machine to obtain $\boldsymbol{y}_t$. Let $\mathcal{T}_{g,n}$ denote the row set of sub-matrix $\mathbf{X}_g$ computed at machine $n \in \mathcal{N}_t$.

Definition 1: **(Computation load)** Let the computation load matrix be $\boldsymbol{M}$ and each entry of $\boldsymbol{M}$, $[\boldsymbol{M}]_{g,n} = \mu[g, n]$, is the computation load of sub-matrix $\mathbf{X}_g$ at machine n defined as

$$\mu[g, n] \overset{\Delta}{=} \frac{|\mathcal{T}_{g,n}|}{q/G}. \tag{2}$$

If $\mathbf{X}_g \notin \mathcal{Z}_n$, $\mu[g, n] = 0$. The computation load vector for N machines, $\boldsymbol{\mu} = [\mu[1], \cdots, \mu[n]]$, is defined as

$$\mu[n] = \sum_{g \in [G]} \mu[g, n], \ \ \forall n \in \mathcal{N}_t, \tag{3}$$

which is the sum of the fractions of rows of the corresponding stored sub-matrices computed by machine n at time step t.

◇

Note that $\mathcal{T}_{g,n}$, $\boldsymbol{M}$ and $\boldsymbol{\mu}$ may change with each time step, but reference to t is omitted for ease of disposition. Moreover, the machines have varying computation speed defined by the

[1] Stragglers are often referred to as the machines with abnormally slower speed.

strictly positive vector, $\boldsymbol{s}$, which is known for each time step and defined as follows.

Definition 2: **(Computation Speed)** The computation speed vector $\boldsymbol{s}$ is a length-N vector with elements $s[n]$, $n \in [N]$, where $s[n]$ is the speed of machine n measured as the inverse of the time it takes machine n to compute all rows of one of its assigned sub-matrix.[2]

$\Diamond$

The computation time is dictated by VMs taking the most time to perform its assigned tasks, and defined as follows.

Definition 3: **(Computation Time)** The computation time in a particular time step is defined as

$$\begin{aligned} c(\boldsymbol{M}) = c(\boldsymbol{\mu}) &\triangleq \max_{n \in \mathcal{N}_t} \frac{\mu[n]}{s[n]} \\ &= \max_{n \in \mathcal{N}_t} \frac{\sum_{g \in [G]} \mu[g,n]}{s[n]}. \end{aligned} \tag{4}$$

$\Diamond$

A. USEC without straggler tolerance

We first formulate the optimization framework for USEC systems without straggler tolerance. For a fixed storage placement $\boldsymbol{\mathcal{Z}}$, we can formulate the following optimization problem.

$$\underset{\mathcal{T}_{g,n}}{\text{minimize}} \quad c(\boldsymbol{M}) \tag{5a}$$

$$\text{subject to:} \bigcup_{n \in \mathcal{N}_t : \mathbf{X}_g \in \mathcal{Z}_n} \mathcal{T}_{g,n} = \left[\frac{q}{G}\right], \forall g \in [G]. \tag{5b}$$

It can be shown that the optimization problem (5) is equivalent to the following convex optimization problem.

$$\underset{\boldsymbol{M}}{\text{minimize}} \quad c(\boldsymbol{M}) = \max_{n \in \mathcal{N}_t} \frac{\sum_{g \in [G]} \mu[g,n]}{s[n]} \tag{6a}$$

$$\text{subject to:} \sum_{n \in \mathcal{N}_t : \mathbf{X}_g \in \mathcal{Z}_n} \mu[g,n] = 1, \forall g \in [G], \tag{6b}$$

$$\mu[g,n] = 0, \forall \mathbf{X}_g \notin \mathcal{Z}_n, n \in \mathcal{N}_t, \tag{6c}$$

$$0 \leq \mu[g,n] \leq 1, \forall n \in \mathcal{N}_t. \tag{6d}$$

It can be seen that by solving (6), we can obtain the optimal computation assignment $\boldsymbol{M}^\star$, which can be used to find the corresponding $\mathcal{T}_{g,n}$ straightforwardly since each row in $\mathbf{X}_g$ is computed only once (see Section III for examples).

B. USEC with straggler tolerance

When straggler tolerance is incorporated into the USEC framework, we use the *redundant task assignment* approach, meaning that each row in $\mathbf{X}$ can be computed $1+S$ times in order to tolerate at most S stragglers. This implies that the computation can be recovered when any S machines, denoted by $\mathcal{S}$, of the available machines $\mathcal{N}_t$ become stragglers and $\mathcal{S}$ is not known a priori. Hence, this problem becomes a combinatorial optimization problem. In particular, a computation assignment within $\mathbf{X}_g$ is defined by F_g disjoint sets of rows in $\mathbf{X}_g$, i.e., $\boldsymbol{\mathcal{M}}_g = \{\mathcal{M}_{g,1}, \ldots, \mathcal{M}_{g,F_g}\}$ such that $\bigcup_{f \in [F_g]} \mathcal{M}_{g,f} = \left[\frac{q}{G}\right]$. Then, F_g sets of machines, $\boldsymbol{\mathcal{P}}_g = \{\mathcal{P}_{g,1}, \ldots, \mathcal{P}_{g,F_g}\}$, which store and perform computation over $\mathbf{X}_g$, are defined such that $\mathcal{P}_{g,f} \subseteq \{n \in \mathcal{N}_t : \mathbf{X}_g \in \mathcal{Z}_n\}$, $|\mathcal{P}_{g,f}| = 1 + S, \forall f \in [F_g]$ and machines in $\mathcal{P}_{g,f}$ computes the row set $\mathcal{M}_{g,f}$ in $\mathbf{X}_g$. Note that $\mathcal{T}_{g,n} = \bigcup_{f \in [F_g] : n \in \mathcal{P}_{g,f}} \mathcal{M}_{g,f}$. The sets $\boldsymbol{\mathcal{M}}_g$, $\boldsymbol{\mathcal{P}}_g$ and F_g may vary with each time step based on machines' availability.

In a given time step t, our goal is to design the task assignments, $\boldsymbol{\mathcal{M}}_g, \boldsymbol{\mathcal{P}}_g, g \in [G]$, such that the computation $\boldsymbol{y}_t = \boldsymbol{X}\boldsymbol{w}_t$ can be recovered when some VMs are stragglers that do not provide their assigned computations to the master machine.

Then, we aim to design the computation assignment that minimizes the computation time of (4) resulting from the computation load matrix defined in (2). In time step t, given $\boldsymbol{\mathcal{Z}}$, $\mathcal{N}_t$ and $\boldsymbol{s}$, the optimal computation time, $c^\star$, is the minimum of computation times defined by all possible task assignments, such that S stragglers can be tolerated and the computation can be recovered. In particular, $c^\star$ is the optimal value of the following combinatorial optimization problem.

$$\underset{\boldsymbol{\mathcal{M}}_g, \boldsymbol{\mathcal{P}}_g}{\text{minimize}} \; c(\boldsymbol{M}) \tag{7a}$$

$$\text{s.t.} \bigcup_{f \in [F_g]} \mathcal{M}_{g,f} = \left[\frac{q}{G}\right], \forall g \in [G], \tag{7b}$$

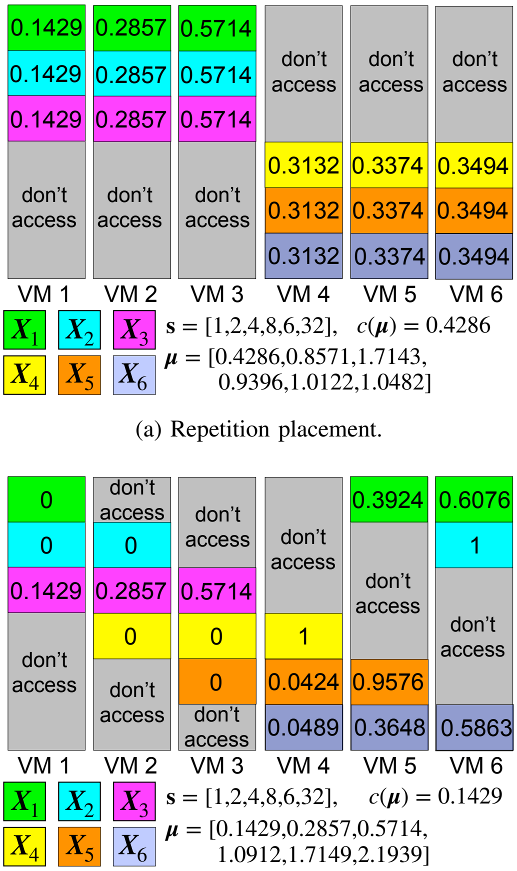
$$\begin{aligned} &|\mathcal{P}_{g,f} \setminus \mathcal{S}| \geq 1, \forall g \in [G], \mathcal{P}_{g,f} \in \boldsymbol{\mathcal{P}}_g, \\ &\qquad \forall \mathcal{S} \subset \mathcal{N}_t, |\mathcal{S}| = S. \end{aligned} \tag{7c}$$

The optimization problem (7) is combinatorial and the optimal solution is challenging. In the following, we will propose a novel low-complexity algorithm to achieve the optimal solution for this problem. Interestingly, the *filling algorithm* introduced in the CSEC framework with heterogeneous computing speed [9] or the heterogeneous storage-constrained private information retrieval problem [10] can be applied here with small modifications to obtain the proposed optimal solution for (7). We consider this as one of the main theoretical contributions of this paper.

III. Examples

In this section, we will illustrate two examples of the proposed USEC framework with and without straggler tolerance, respectively, under the homogeneous storage constraint. We consider two commonly used uncoded storage schemes, which are *fractional repetition placement* (referred to as repetition placement hereafter) and *cyclic placement*, which are widely used in the distributed storage and gradient coding literatures (e.g., [11]–[14]). In particular, we consider a USEC system with $N = 6$ VMs and the speed vector is $\mathrm{s} = [1, 2, 4, 8, 16, 32]$. The data matrix $\mathbf{X}$ is partitioned into $G = 6$ sub-matrices, each placed into $J = 3$ machines. Fig. 1 shows this system with repetition placement (Fig.1a) and with cyclic placement (Fig. 1b), respectively. Let $N = N_t$, all $\mu[g,n], g \in [6], n \in [N]$ are computed by solving the convex optimization problem (6). In Fig. 1,

[2] As mentioned in the Section I, based on our measurement and the duration of one time step, it is appropriate to model the computing speed of a virtual machine does not vary during one time step.


0.1429
0.2857
0.5714
0.1429
0.2857
0.5714
0.1429
0.2857
0.5714
don't access
0.3132
0.3374
0.3494
VM 1
VM 2
VM 3
VM 4
VM 5
VM 6
X_1 X_2 X_3 X_4 X_5 X_6
s = [1,2,4,8,6,32], $c(\boldsymbol{\mu})$ = 0.4286
$\boldsymbol{\mu}$ = [0.4286,0.8571,1.7143, 0.9396,1.0122,1.0482]
0
0
0.1429
0.2857
0.5714
0
0
0
1
0.0424
0.9576
0.0489
0.3648
0.5863
0.3924
0.6076
1
s = [1,2,4,8,6,32], $c(\boldsymbol{\mu})$ = 0.1429
$\boldsymbol{\mu}$ = [0.1429,0.2857,0.5714, 1.0912,1.7149,2.1939]

(a) Repetition placement.

(b) Cyclic placement.

Fig. 1: Illustration of the proposed USEC framework.

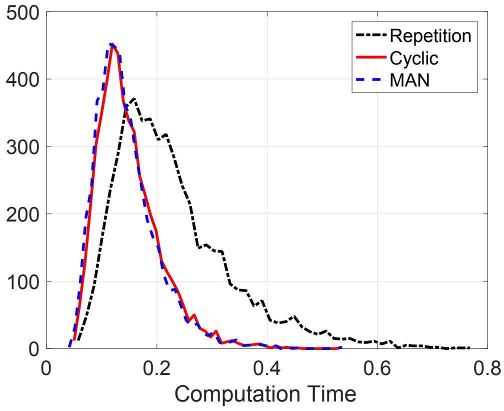


Fig. 2: Comparison of histograms of $C(\boldsymbol{M})$ for repetition, cyclic and MAN storage placements over 5000 realizations of the computing speed vector.

TABLE I: Comparison between MAN, cyclic and repetition placements.

computation time	cyclic	repetition	MAN
mean	0.1492	0.2296	0.1442
variance	0.0033	0.0114	0.0032

the colors represent the storage placement of each sub-matrix and the numbers inside represent the corresponding $\mu[g,n]$ for sub-matrix g and machine n. The computation time for the cyclic placement is $c(\boldsymbol{\mu}) = 0.1429$, which is significantly better than that of the repetition placement $c(\boldsymbol{\mu}) = 0.4286$. However, interestingly, the cyclic placement is not necessarily better than the repetition placement for any speed vector. For example, if machines 3 and 4 are much faster than other VMs, then the repetition placement can be better than the cyclic placement since machines 3 and 4 stores the entire data matrix under the repetition placement. In order to have a better understanding of this phenomenon, we ran an experiment by randomly generating s based on an exponential distribution. By solving the minimum computation time for each s using (5), we obtain the distribution of the computation time for these two storage placements shown in Fig. 2, where the cyclic placement (red) is much better than the repetition placement (black) in most realizations. In particular, there are only 68 cyclic placement realizations out of 5000 worse than repetition placement realizations. Although these results show the promising performance of the cyclic storage placement, it is not optimal in general. For example, using the Maddah-Ali Niesen coded caching (MAN) storage placement scheme [15] to repeat the same experiment, we can obtain slightly better results as shown in Fig. 2 (blue). In particular, out of 5000 realizations, there are only 9 MAN storage realizations worse than repetition placement realizations and 1621 MAN placement realizations worse than cyclic placement realizations. Moreover, the MAN placement indeed achieves the minimum computing time in terms of both mean and variance compared to cyclic and repetition placements (see Table I).

When straggler tolerance is considered, we need to solve problem (7) to obtain the optimal $\boldsymbol{M}^\star$ and then find a feasible computation assignment that meets $\boldsymbol{M}^\star$. Consider an example of a USEC system with homogeneous computing speed. Here, we let $N = N_t = 6$, $J = 3$, $S = 1$, and the repetition placement is used. The optimal $\mu^\star[g,n], g \in [6], n \in [6]$ are shown in Fig. 3 and the optimal $\boldsymbol{\mu}^\star = [2,2,2,3,3]$. The optimal computation time is $c^\star(\boldsymbol{\mu}) = 3$.

IV. Proposed USEC Design

The proposed USEC design with straggler tolerance is given by Algorithm 1, which is obtained by solving the combinatorial optimization (7) in a similar fashion as in [8] (line 6 in Algorithm 1). The proposed design is adaptive by measuring (line 14 in Algorithm 1) and updating (line 4 in Algorithm 1) the speed vector at time step. Interestingly, this

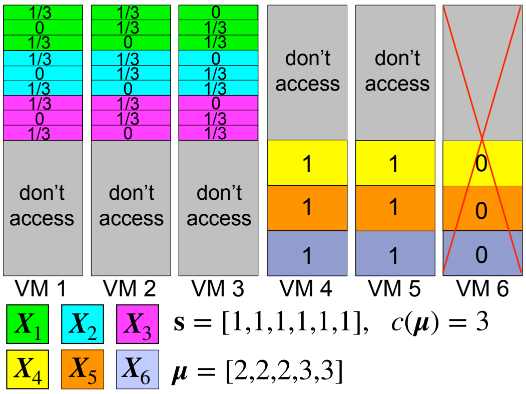


Fig. 3: Illustration of uncoded USEC with straggler tolerance for $S = 1$ using redundant task assignment.

algorithm adapts the previous CSEC (not USEC) computation assignment [8] to assign computations to $1 + S$ machines.

Next we will explain the proposed design. Since the proposed design without straggler tolerance is a special case of the general design with straggler tolerance for the combinatorial optimization problem (7), then we will focus on designing algorithms to solve (7).

Similar to [8], we will solve the combinatorial optimization problem (7) exactly in two steps. In the first step, we solve the following relaxed convex optimization problem to obtain the optimal $\boldsymbol{M}^\star$ without considering whether such a computation assignment exists or not.

$$\underset{\boldsymbol{M}}{\text{minimize}} \quad c(\boldsymbol{M}) = \max_{n \in \mathcal{N}_t} \frac{\sum_{g \in [G]} \mu[g,n]}{s[n]} \tag{8a}$$

$$\text{subject to:} \quad \sum_{n \in \mathcal{N}_t : \mathbf{X}_g \in \mathcal{Z}_n} \mu[g,n] = 1 + S, \forall g \in [G], \tag{8b}$$

$$\mu[g,n] = 0, \forall \mathbf{X}_g \notin \mathcal{Z}_n, n \in \mathcal{N}_t, \tag{8c}$$

$$0 \leq \mu[g,n] \leq 1, \forall n \in \mathcal{N}_t. \tag{8d}$$

The difference between (8) and (6) is to change (6b) from $\sum_{n \in \mathcal{N}_t : \mathbf{X}_g \in \mathcal{Z}_n} \mu[g,n] = 1, \forall g \in [G]$ to $\sum_{n \in \mathcal{N}_t : \mathbf{X}_g \in \mathcal{Z}_n} \mu[g,n] = 1 + S, \forall g \in [G]$ as in (8b). After obtaining the optimal $\mathbf{M}^\star$, we will apply the *filling algorithm* developed in [8] to assign computations for each $\mathbf{X}_g \in \mathcal{Z}_n, n \in \mathcal{N}_t$. Now we will describe the filling algorithm for USEC with homogeneous and heterogeneous computing speed, respectively.

Proposed USEC with homogeneous computation assignment: Consider $\mathcal{N}_g = \{n : \mathbf{X}_g \in \mathcal{Z}_n\}$ with $|\mathcal{N}_g| = N_g$. Then we define a computation assignment with $F_g = N_g$ row sets of $\mathbf{X}_g$. There are N_g disjoint equally-sized row sets that collectively span all rows: $\mathcal{M}_{g,f} = \{1+(f-1)\frac{q}{N_g G}, \ldots, f\frac{q}{N_g G}\}$ for $f \in [N_g]$. Then, define a cyclic assignment such that machine set $\mathcal{P}_{g,f} = \{f\%N_g, \ldots, (f+S)\%N_g\}$ for $f \in [N_g]$, where we define $a\%N_g \triangleq a - \left\lfloor \frac{a-1}{N_g} \right\rfloor N_g$ to facilitate the cyclic design.

Proposed USEC with heterogeneous computation assignment: Given the computation load matrix $\boldsymbol{M}^\star$, we can obtain the computation assignment by applying the assignment algorithm in [8] to assign computations to $1+S$ VMs for each $\mathbf{X}_g$ (line 6 in Algorithm 1). The computation assignment algorithm for $\mathbf{X}_g$ is given by Algorithm 2. The detailed explanation of Algorithm 2 can be found in [9].

Remark 1: For both designs, we observe that the computation time $c(\boldsymbol{M})$ increases with the straggler tolerance, S. This demonstrates a trade-off between the computation time and straggler tolerance of the system.

Algorithm 1 Adaptive Straggler Tolerant Uncoded Storage Elastic Computing

```
Input: $\hat{\boldsymbol{s}}$, $\gamma$, $S$, $T$, $\boldsymbol{w}_1$
1:  $\boldsymbol{\nu} \leftarrow \hat{\boldsymbol{s}}$: same for all worker VMs
2:  for $t \in [T]$ do
3:      At Master Machine:
4:          $\hat{\boldsymbol{s}} \leftarrow \gamma\boldsymbol{\nu} + (1-\gamma)\hat{\boldsymbol{s}}$ (update estimate of speed vector).
5:          $\mathcal{N}_t \leftarrow$ list of available machines
6:          $\{F_g, \boldsymbol{\mathcal{M}}_g, \boldsymbol{\mathcal{P}}_g : \forall g \in [G]\} \leftarrow$ Results of computation assignment algorithm for $\mathbf{X}_g$ with straggler tolerance of $S$ for available machines $\mathcal{N}_t$ with speeds of $\hat{\boldsymbol{s}}$
7:          Send $\boldsymbol{w}_t$ and $\{F_g, \boldsymbol{\mathcal{M}}_g, \boldsymbol{\mathcal{P}}_g : \forall g \in [G]\}$ to worker VMs
8:      At Worker VMs:
9:          $n \leftarrow$ index of worker VM
10:         $\mu[n] \leftarrow$ total computation load of worker VM $n$
11:         $\tau_1 \leftarrow$ current time
12:         Perform assigned computations based on $\{F_g, \boldsymbol{\mathcal{M}}_g, \boldsymbol{\mathcal{P}}_g : \forall g \in [G]\}$
13:         $\tau_2 \leftarrow$ current time
14:         $\nu[n] \leftarrow \mu[n]/(\tau_2 - \tau_1)$ (calculate speed based on current time step)
15:         Send computations and $\nu[n]$ to Master Machine
16:     At Master Machine: after receiving results from at most $N_t - S$ workers.
17:         $\boldsymbol{w}_{t+1} \leftarrow$ Combine worker results
18: end for
Output: $\boldsymbol{w}_T$
```

V. Evaluations on Amazon EC2

We evaluate the proposed algorithm using power iteration applications on Amazon EC2 instances. The goal is to compare the performance difference in terms of computation time between the homogeneous and heterogeneous task assignments.

Power Iteration [5]: The power iteration algorithm computes the largest eigenvalue and the corresponding eigenvector of a large matrix $\mathbf{X}$. In particular, it starts with a vector $\mathbf{b}_0$, which may be an approximation to the dominant eigenvector or a random vector. The method is described by the recursive relation, $\mathbf{b}_{k+1} = \frac{\mathbf{X}\mathbf{b}_k}{\|\mathbf{X}\mathbf{b}_k\|}$. The sequence $\mathbf{b}_k$ converges to an eigenvector associated with the dominant eigenvalue. It can be seen that at each iteration, we can directly apply Algorithm 1. In particular, a dense $6,000$-by-$6,000$ symmetric matrix is row-wise split into $G = 6$ sub-matrices which will

Algorithm 2 Computation Assignment for $\mathbf{X}_g$ for Heterogeneous Computing Speed

```
Input: $\boldsymbol{\mu}_g^\star$, $q$, $\boldsymbol{\mathcal{Z}}$ and $\mathcal{N}_g = \{1, \cdots, N_g\}$.
$\boldsymbol{m} \leftarrow \boldsymbol{\mu}_g^\star$
$f \leftarrow 0$
while $\boldsymbol{m}$ contains a non-zero element do
    $f \leftarrow f + 1$
    $L' \leftarrow \sum_{i=1}^{N_g} m[i]$
    $N' \leftarrow$ number of non-zero elements in $\boldsymbol{m}$
    $\boldsymbol{\ell} \leftarrow$ indices that sort the non-zero elements of $\boldsymbol{m}$ from smallest to largest[3]
    $\mathcal{P}_{g,f} \leftarrow \{\ell[1], \ell[N' - L + 2], \dots, \ell[N']\}$
    if $N' \geq L + 1$ then
        $\alpha_{g,f} \leftarrow \min\left(\frac{L'}{L} - m[\ell[N' - L + 1]], m[\ell[1]]\right)$[4]
    else
        $\alpha_{g,f} \leftarrow m[\ell[1]]$
    end if
    for $n \in \mathcal{P}_{g,f}$ do
        $m[n] \leftarrow m[n] - \alpha_{g,f}$
    end for
end while
$F \leftarrow f$
Partition rows $[\frac{q}{G}]$ of $\mathbf{X}_g$ into $F$ disjoint row sets $\mathcal{M}_{g,1}, \dots, \mathcal{M}_{g,F}$ of size $\frac{\alpha_1 q}{G}, \dots, \frac{\alpha_F q}{G}$ rows, respectively
Output: $F$, $\{\mathcal{M}_{g,1}, \dots, \mathcal{M}_{g,F}\}$ and $\{\mathcal{P}_{g,1}, \dots, \mathcal{P}_{g,F}\}$
```

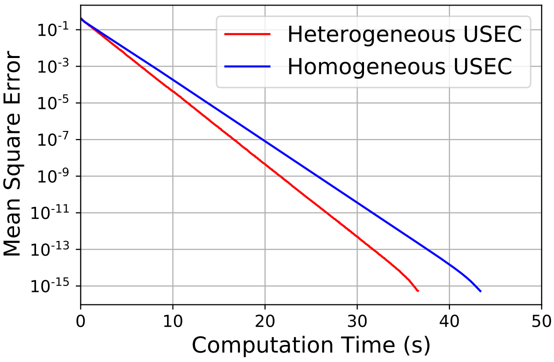


Fig. 4: **Power Iteration**: Results using USEC designs on Amazon EC2 without stragglers (top) and with 2 stragglers each iteration (bottom). The y-axis represents the normalized mean square error between the true dominant eigenvector and the estimated eigenvector.

be stored at each VM. We apply the repetition placement. A vector of length $6,000$ is updated by performing a matrix-vector multiplication in a distributed manner on the available virtual machines. The master machine combines the results and normalizes the vector. This process is repeated such that the vector converges to the eigenvector associated with the largest eigenvalue.

The network has one `t2.x2large` master machine with 8 vCPUs and 32 GiB of memory. The worker virtual machines consist of 3 `t2.large` instances, each with 2 vCPUs and 8 GiB of memory, and 3 `t2.xlarge` instances, each with 4 vCPUs and 16 GiB of memory. Similar to [4], we observed that all virtual machines have very different computing speed. For simplicity, we let $N = N_t$ and $S = 0$ in order to show the advantage of the heterogeneous task assignment over the homogeneous task assignment. The result is shown in Fig. 4, where the gain of Algorithm 1 is about 20% in terms of the computation time.

VI. Conclusions and Future Directions

In this paper, we introduce the concept of USEC and propose a new optimization framework on USEC with heterogeneous computing speed to minimize the overall computation time. In particular, we consider the USEC systems under different uncoded storage placements and with or without straggler tolerance. For both scenarios, we propose optimal algorithms given the storage placements. These algorithms are evaluated using real applications on Amazon EC2 to demonstrate their gains in terms of computation time compared to the designs using the homogeneous computing speed assumption. Due to the advantages of USEC systems, we believe this is one of the important future research directions in the area of elastic computing.

One obvious open problem in USEC is to find the optimal storage placement. From Table I, it can be seen that the MAN storage placement can achieve the minimum computation time while it is unclear whether it is optimal in general. In addition, the combinatorial optimization problems (5) and (7) are just one way of formulating the USEC problem. It is unclear whether there are better ways of formulating this problem to minimize the computation time. Another important future research direction is to implement the proposed algorithm for other applications possibly in machine learning and data mining and investigate the performance gains in terms of computation time.

Acknowledgment

This work was supported by National Science Foundation (NSF) CAREER Award 2145835.

References

[1] 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.

[2] 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.

[3] 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.

[4] 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.

[3] $\boldsymbol{\ell}$ is an N'-length vector and $0 < m[\ell[1]] \leq m[\ell[2]] \leq \cdots \leq m[\ell[N']]$.
[4] This is the condition obtained by using Lemma 1 in [9].

[5] R. V. Mises and H. Pollaczek-Geiringer, "Praktische verfahren der gleichungsauflösung.," *ZAMM-Journal of Applied Mathematics and Mechanics/Zeitschrift für Angewandte Mathematik und Mechanik*, vol. 9, no. 1, pp. 58–77, 1929.

[6] D. Wang, G. Joshi, and G. Wornell, "Efficient task replication for fast response times in parallel computation," in *The 2014 ACM international conference on Measurement and modeling of computer systems*, 2014, pp. 599–600.

[7] A. Behrouzi-Far and E. Soljanin, "Efficient replication for straggler mitigation in distributed computing," *arXiv preprint:2006.02318*, 2020.

[8] N. Woolsey, R.-R. Chen, and M. Ji, "Heterogeneous computation assignments in coded elastic computing," in *2020 IEEE International Symposium on Information Theory (ISIT)*, 2020, pp. 168–173.

[9] 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.

[10] N. Woolsey, R.-R. Chen, and M. Ji, "Uncoded placement with linear sub-messages for private information retrieval from storage constrained databases," *IEEE Transactions on Communications*, vol. 68, no. 10, pp. 6039–6053, 2020.

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

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

[13] S. Wang, J. Liu, and N. Shroff, "Fundamental limits of approximate gradient coding," *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, vol. 3, no. 3, pp. 1–22, 2019.

[14] N. Raviv, I. Tamo, R. Tandon, and A. G. Dimakis, "Gradient coding from cyclic mds codes and expander graphs," *IEEE Transactions on Information Theory*, vol. 66, no. 12, pp. 7475–7489, 2020.

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