Optimizing the Transition Waste in Coded Elastic Computing

Hoang Dau, Ryan Gabrys, Yu-Chih Huang, Chen Feng, Quang-Hung Luu, Eidah Alzahrani, Zahir Tari

Abstract—Motivated by recently available services in the cloud computing industry, e.g., EC2 Spot or Azure Batch, where spare/low-priority virtual machines are offered at a fraction of the price of the on-demand instances but can be preempted on short notice, we investigate coded computing solutions over elastic resources, where the set of available machines may change in the middle of the computation. Our contributions are two-fold: We first propose an efficient method to minimize the transition waste, a newly introduced concept quantifying the total number of tasks that existing machines have to abandon or take on anew when a machine joins or leaves, for the cyclic elastic task allocation scheme recently proposed in the literature (Yang et al. ISIT’19). We then proceed to generalize such a scheme and introduce new task allocation schemes based on finite geometry that achieve zero transition wastes as long as the number of active machines varies within a fixed range. The proposed solutions can be applied on top of existing coded computing schemes tolerating stragglers.

I. INTRODUCTION

Coded distributed computing [1]–[3], built upon algorithmic fault tolerance [4], is a recently emerging paradigm where computation redundancy is employed to tackle the straggler effect - one slow machine may become a bottleneck of the whole system. As a toy example [1], to perform a matrix-vector multiplication $Ax$, a master machine first partitions the matrix $A$ into two equal-sized submatrices $A_1$ and $A_2$ and then distributes $A_1$, $A_2$, and $A_1 + A_2$ to three worker machines, respectively. These machines also receive the vector $x$ and perform three multiplications $A_1x$, $A_2x$, and $(A_1 + A_2)x$ in parallel. Clearly, $Ax$ can be recovered by the master from the outcomes of any two workers. Thus, this coded scheme can tolerate one straggler. The potential of coded distributed computing has been extensively investigated in the literature, e.g. [5]–[11].

Most of the research in the literature of coded distributed computing, however, assumes that the set of available worker machines remains fixed. This critical limitation renders current coded computing schemes inapplicable in an environment where low-cost elastic resources are readily available. In fact, major cloud computing providers, very recently, started offering spare virtual machines at a price up to 90% cheaper than that of the on-demand machines, e.g. Amazon EC2 Spot [12] and Microsoft Azure Batch [13], albeit at the cost of low priority in the sense that these machines can be preempted (removed) for a higher-priority customer under a short notice (e.g. two minutes in the case of Amazon Spot). This new development in the cloud computing industry provides customers with an opportunity to have large computing resources at a fraction of the cost of the normal on-demand service. Realizing this opportunity, however, requires the user to develop much more flexible distributed computing paradigms in order to efficiently exploit elastic resources where low-cost machines can leave and join at any time during the computation cycle.

Recently, Yang et al. [14] proposed an elegant technique extending coded computing to deal with elastic resources. Their key idea is to couple a cyclic task allocation scheme, which works for any number of machines, with a coded computing scheme to guarantee that a) as long as there are

| $S^2_1$ | $S^2_2$ | $S^2_3$ | $S^2_4$ | $S^2_5$ |
|---|---|---|---|---|
| 0 → 3 | 0 → 3 | 0 → 3 | 0 → 3 | 0 → 3 |
| 4 → 7 | 4 → 7 | 4 → 7 | 4 → 7 | 4 → 7 |
| 8 → 11 | 8 → 11 | 8 → 11 | 8 → 11 | 8 → 11 |
| 12 → 15 | 12 → 15 | 12 → 15 | 12 → 15 | 12 → 15 |
| 16 → 19 | 16 → 19 | 16 → 19 | 16 → 19 | 16 → 19 |

(a) Cyclic task allocation for five machines [14].

| $S^3_1$ | $S^3_2$ | $S^3_3$ | $S^3_4$ |
|---|---|---|---|
| 0 → 4 | 0 → 4 | 0 → 4 | 0 → 4 |
| 5 → 9 | 5 → 9 | 5 → 9 | 5 → 9 |
| 10 → 14 | 10 → 14 | 10 → 14 | 10 → 14 |
| 15 → 19 | 15 → 19 | 15 → 19 | 15 → 19 |

(b) Cyclic task allocation for four machines [14]. The transition waste from five to four machines is 12 tasks.

| $S^4_1$ | $S^4_2$ | $S^4_3$ | $S^4_4$ |
|---|---|---|---|
| 2 → 6 | 2 → 6 | 2 → 6 | 2 → 6 |
| 7 → 11 | 7 → 11 | 7 → 11 | 7 → 11 |
| 12 → 16 | 12 → 16 | 12 → 16 | 12 → 16 |

(c) Our proposed shifted cyclic task allocation for four machines that results in an optimal transition waste among all cyclic schemes (zero in this case).

Fig. 1: Illustration of the sub-optimality of the cyclic task allocation scheme proposed in [14] with respect to the transition waste when Machine 5 leaves. Here, we use $a \to b$ to denote the set $\{a, a + 1, \ldots, b\}$ (mod $F$), where $F = 20$. a sufficient number of machines working, the scheme can tolerate stragglers, and b) the workload at each machine is inversely proportional to the number of available machines. The elastic coded computing scheme proposed in [14] was evaluated in the multi-tenancy cluster at Microsoft using the Apache REEF Elastic Group Communication framework, and shown to reduce the completion time of matrix-vector multiplication and linear regression by up to 46% compared to ordinary coded computing schemes.

Relaxing the cyclic task allocation proposed in [14], we investigate a more general elastic task allocation problem, in which we seek to address the following key questions.

- **Task allocation:** given a set of tasks and a set of machines, how do we assign tasks to machines so that all machines are assigned an equal number of tasks (workload balance) and every task is covered by the same number of machines? This can be solved, e.g. by using the cyclic scheme employed in [14] or its generalizations developed in this work.

- **Transition reallocation:** when an elastic event occurs (machines leaving/joining), how do we reallocate the tasks to the new set of machines to minimize the transition waste, i.e. the total number of tasks that existing machines must abandon or take over when one machine joins/leaves? This is a much more challenging question and is our focus in this work.

We illustrate in a toy example (Fig. 1) the concept of transition waste and explain why the cyclic elastic task allocation scheme in [14] is suboptimal with respect to this new metric. We consider the computation of $Ax$ where $A$ can be partitioned row-wise into 40 equal-sized sub-matrices.

978-1-7281-6432-8/20/$31.00 ©2020 IEEE 174 ISIT 2020
We first group these sub-matrices into 20 groups, e.g. \(\{A_1, A_2\}, \{A_3, A_4\}\), and so forth. Then each group is assigned a task index from 0 to 19. Task 0, for instance, corresponds to the computation of \(\{A_1, x, A_2, x\}\). Task 0 is encoded into five subtasks: \(A_1 x, A_2 x, (A_1 + A_2)x, (A_1 + 2A_2)x, \) and \((A_1 + 3A_2)x\). A machine taking Task 0 means it computes one of these five subtasks. Similar to the earlier discussion, any three out of five subtasks/machines form a coded computing group that can recover Task 0 given one straggler.

Hence, abstracting away the underlying coded computing scheme, which can be designed independently of the task allocation scheme in consideration, given \(F = 20\) tasks, we require that each task must be covered by precisely \(L = 3\) machines. This can be met by using the cyclic scheme in [14]: each of the \(N\) machines is preloaded with a set of \(F\) tasks, which is then divided into \(N\) equal consecutive subsets of size \(F/N\), and each machine works on tasks in the union of \(L\) consecutive such subsets. For instance, when \(N = 5\), Machine 1 works on the set of tasks \(S_1^0 = \{0, 1, \ldots, 11\} = \{0, \ldots, 3\} \cup \{4, \ldots, 7\} \cup \{8, \ldots, 11\}\), Machine 2 works on \(S_2^0 = \{4, 5, \ldots, 15\}\), and so forth (see Fig. 1 (a)). Note that each machine takes 12 tasks and due to the cyclic task allocation scheme, each task is covered by three machines.

In Fig. 1 (b), four machines are available, each of which takes 15 tasks. As Machine 5 has left, it is necessary that each of the four machines must take \(3 = 15 - 12\) more tasks. Ideally, when the transition from five machines to four machines occurs, each machine continues its existing tasks and works on three new tasks. This is true for Machine 1 because \(S_1^0 \subset S_1^1\). However, it is not the case for other machines. For instance, Machine 3 has to abandon two tasks (8 and 9) and takes over five new tasks (0 $\rightarrow$ 4). The transition waste at Machine 3 is \((2 + 5) - 3 = 4\) tasks. Note that three is the necessary increase in the number of tasks each machine must take and so we subtract that amount. The transition wastes at other machines can be computed in a similar manner. The total transition waste is \((0 + 3 - 3) + (1 + 4 - 3) - 2 + (5 - 3) + (3 + 6 - 3) = 12\) tasks.

Therefore, sticking to the cyclic allocation scheme of [14], we waste 12 tasks. However, it turns out that the transition waste can be reduced to zero if we use the allocation scheme in Fig. 1 (c) instead. In this case, as \(S_n^0 \subset S_n^1\), the transition wastes at all four machines are zero. The trick is to shift the cyclic task allocation by a right amount (−3 in this case) to maximize the overlaps between \(S_n^0\) and \(S_n^1\), \(n = 1, \ldots, 4\).

Our main contributions are summarized below.

- We first introduce the new concept of transition waste of an elastic task allocation scheme, which quantifies the total number of tasks that existing machines must abandon or take over when one machine joins or leaves, less the necessary amount. A reduction in transition waste implies lower computation and communication costs (Remark 2).

- We then compute explicitly the transition waste incurred in the cyclic task allocation scheme introduced in [14] when machines leave and join (Theorems 1, 2) and propose shifted cyclic schemes that minimize the transition waste among all cyclic schemes (Theorems 3, 4). The transition waste of a shifted cyclic scheme is generally greater than zero.

- Lastly, we show that there exists a zero-waste transition when a machine leaves if and only if there exists a perfect matching in a certain bipartite graph, using the famous Hall’s marriage theorem. Based on this new insight, we construct several novel task allocation schemes based on finite geometry that achieve zero transition wastes when the number of active machines varies within a fixed range.

We emphasize that our task allocation schemes are designed separately from the underlying coded computing scheme and hence can be applied on top of existing coded computing schemes. Missing proofs can be found in our full paper [15].

**II. Preliminaries**

We define in this section the elastic task allocation scheme, which generalizes the cyclic scheme originally proposed by Yang et al. [14], and the new concept of the transition waste.

We use \(N\) for the number of available machines, \(F\) for the common number of pre-loaded tasks at each machine, and \(L\) as minimum number of available machines so that the scheme still works (\(L \leq N\)). Each task is represented by a label from \([F] = \{0, 1, \ldots, F - 1\}\). We assume that all tasks consume an equal amount of resources (storage, memory, CPU). Let \([F] = \{1, 2, \ldots, F\}\) and \([A, B] = \{A, A + 1, \ldots, B\}\). Let \(2[F]\) be the power set of the set \([0, 1, \ldots, F - 1]\) and \((2[F])^N = 2[F]^N \times 2[F]^N \times \cdots \times 2[F]^N\) the \(N\)-ary Cartesian power of \(2[F]\).

**Definition 1** (Task allocation scheme). An ordered list of sets \(S^N = (S^N_1, \ldots, S^N_n) \in (2[F])^N\), where \(S^N_n \subset [F]\), \(n \in [N]\), is referred to as an \((N, L, F)\) task allocation scheme \((N, L, F)\)-TAS if it satisfies the following two properties.

- (L-Redundancy) each element in \([F]\) is included in precisely \(L\) sets in \(S^N\), and
- (Load-Balance) \(|S^N_n| = LF/N\) for all \(n \in [N]\). Here we assume that \(LF/N \in \mathbb{Z}\).

When a machine leaves/joins, we must reallocate tasks to a new set of machines. Thus, we must extend the notion of a task allocation scheme (TAS) to that of an elastic task allocation scheme (ETAS). We explain in [15, Appendix VLA] how to couple an ETAS and a coded computing scheme to obtain a coded elastic computing scheme that tolerates stragglers.

**Definition 2** (Elastic task allocation). A pair \((S_n^0, T)\) is referred to as an \((N_0, L, F)\) elastic task allocation scheme \((N_0, L, F)\)-ETAS if \(S_n^0\) is the initial \((N_0, L, F)\)-TAS and \(T\) is an algorithm that reallocates tasks when machines leave and join so that the new scheme remains a TAS. More specifically,

\[T: (2[F])^N \times [-1, 1] \times [N] \rightarrow (2[F])^{N - 1} \cup (2[F])^{N + 1}\]

takes as input an \((N, L, F)\)-TAS \(S^N\), where \(L \leq N \leq LF\), a variable \(b \in [-1, 1]\), which represents the elastic event of one machine leaving \((b = -1)\) or joining \((b = 1)\), and an index \(n^* \in [N]\), which indicates the index of the machine that leaves when \(b = -1\) (when \(b = 1\), \(n^*\) is ignored). Moreover, \(T\) returns an output \(S^N\), which is another \((N', L, F)\)-TAS, where \(N' = N + b\). In other words, moving from a set of \(N\) machines to a new set of \(N' = N + b\) machines, \(T\) updates the list of task sets \(S^N\) to obtain \(S^{N'}\), which remains a TAS.

A few remarks are in order. First, we make a simplifying assumption in Definition 2 that each elastic event corresponds to one machine leaving/joining only. In other words, we assume that machines leave and join one after another and not at the same time. Second, while in general we allow \(N\) to take any value in the range \([L, LF]\), it is more tractable to limit \(N\) within a fixed range \([L, N_{\text{max}}]\). We also assume, by adding dummy tasks if necessary, that \(F\) is divisible by any number within this range. Third, when Machine \(n^* \in [N]\) leaves, we index the remaining machines by the set \([N - 1] = \{1, \ldots, N - 1\}\). However, when comparing with the previous TAS, we often use \(\{1, \ldots, n^* - 1, n^* + 1, \ldots, N\}\), instead of \([N - 1]\), so that the same machine is given the same index in the previous and in the current task allocation schemes.
Cyclic elastic task allocation scheme [14]. A simple way to construct an ETAS is to let $T$ depend only on the number of machines and not on the current TAS. More specifically, whenever there are $N$ machines available as the result of an elastic event, we usually use a fixed $(N, L, F)$-TAS

$$S_N^N = (S_N^1, \ldots, S_N^N),$$

$$S_n^N = \left[\frac{(n-1)F}{N} - \frac{(n-1)F}{N} + \frac{LF}{N} - 1\right] \mod F, \quad (1)$$

for $n \in [N]$, where $[A, B] \mod F$ is obtained from $[A, B]$ by applying the modulo operation element-wise. We also assume here that $F/N \in \mathbb{Z}$.

It is straightforward to verify that $S_N^N$ satisfies the Load-Balance and the $L$-Redundancy properties, and therefore, is indeed an $(N, L, F)$-TAS. The reallocation algorithm is trivial: $T(S_{\text{cyc}}^N, 1) = S_{\text{cyc}}^{N+1}$ and $T(S_{\text{cyc}}^N, \{-1, n\}^*) = S_{\text{cyc}}^{N-1}$ for every $n^* \in [N]$. Fig. 1 (a) and (b) illustrate the cyclic ETAS when $N = 5$ and when $N = 4$, $L = 3$, $F = 20$, and $n^* = 5$.

Transition waste. We now define the transition waste during an elastic event when one machine leaves or joins and demonstrate this new concept via a few examples.

Definition 3 (Necessary load change). For a transition from an $(N, L, F)$-TAS $S_N^N$ to another $(N', L, F)$-TAS $S_{N'}^N$, $\Delta_{N,N'} = |LF/N - LF'/N'|$ is referred to as the necessary load change. When $N' = N$, we have $\Delta_{N,N+1} = LF/(N(N+1))$.

The necessary load change, $\Delta_{N,N'} = |S_N^N| - |S_{N'}^N|$, reflects the necessary increase or decrease in the number of tasks each machine must take when one machine leaves or joins, respectively. For instance, when $L = 3$, $F = 20$, if there are $N = 5$ machines, the Load-Balance property requires that each machine runs $LF/N = 12$ tasks, while if there are $N' = 4$ machines due to the removal of one, then each machine runs $LF/N' = 15$ tasks. Therefore, each of the four machines must take $3 = 15 - 12$ more tasks to react to this event. The necessary load change is $3$ in this case.

Definition 4 (Transition waste for one machine). The transition waste incurred at Machine $n$ when transitioning from a set of tasks $S_n^N$ to another set of tasks $S_{n'}^N$ is defined as

$$W(S_n^N \rightarrow S_{n'}^N) = |S_{n'}^N \Delta S_n^N| - \Delta_{N,N'},$$

where $\Delta_{N,N'}$ is the necessary load change (Definition 3) and $A \Delta B$ denotes the symmetric difference between $A$ and $B$. We also use $W_n(S_n^N \rightarrow S_{n'}^N)$ for the case Machine $n$’s leaves.

Remark 1. In Definition 4, we assume that each existing machine keeps its current index in the new TAS, that is, $S_n^N$ and $S_{n'}^N$ refer to the task sets assigned to the same machine.

Remark 2. Note that $|S_n^N \Delta S_{n'}^N| = |S_n^N \setminus S_{n'}^N| + |S_{n'}^N \setminus S_n^N|$ corresponds to the number of scheduled tasks Machine $n$ must abandon (tasks in $S_n^N$ but not in $S_{n'}^N$) and take anew (tasks in $S_{n'}^N$ but not in $S_n^N$). Thus, the transition waste $W(S_n^N \rightarrow S_{n'}^N)$ in Definition 4 measures the maximum number of tasks wasted at Machine $n$ when another machine leaves or joins. As some tasks may have been already completed before the transition, one should abandon as few existing tasks as possible. Likewise, taking on fewer new tasks will decrease the downloading traffic (if the protocol requires new tasks to be downloaded). Thus, a low-waste transition saves computation and network resources and hence reduces the completion time.

The transition waste of a TAS is defined as the total transition wastes at all machines.

Definition 5 (Transition waste). When Machine $N+1$ joins, the transition waste of the transition from an $(N, L, F)$-TAS $S_N^N$ to an $(N+1, L, F)$-TAS $S_{N+1}^N$ is defined as

$$W(S_N^N \rightarrow S_{N+1}^N) \equiv \sum_{n \in [N]} W(S_n^N \rightarrow S_{n+1}^N).$$

When Machine $n^*$ leaves, the transition waste of the transition from an $(N, L, F)$-TAS $S_N^N$ to an $(N-1, L, F)$-TAS $S_{N-1}^N$ is

$$W_{n^*}(S_n^N \rightarrow S_{n-1}^N) \equiv \sum_{n \in [N] \setminus \{n^*\}} W_n(S_n^N \rightarrow S_{n-1}^N).$$

Here, $W(S_n^N \rightarrow S_{n+1}^N)$ and $W_{n^*}(S_n^N \rightarrow S_{n-1}^N)$ denote the transition waste incurred at Machine $n$ (Definition 4).

We demonstrated in the Introduction (Fig. 1a, b, c) two different transitions from a $(5, 3, 20)$-TAS to a $(4, 3, 20)$-TAS, i.e. one machine removed. These have transition wastes 12 and zero, respectively. Another example is given below.

Example 1. Let $L = 2, F = 6, N = 3, N' = 4$. We can verify that $S^3 = \{(0, 1, 2), (0, 1, 2), (0, 1, 2), (3, 4, 5), (0, 1, 2), (0, 1, 2), (0, 1, 2), (3, 4, 5), (3, 4, 5)\}$ is a $(3, 2, 6)$-TAS and $S^4 = \{(0, 1, 2), (0, 1, 2), (0, 1, 2), (3, 4, 5), (3, 4, 5)\}$ is a $(4, 2, 6)$-TAS. The necessary load change when going from three to four machines is $\Delta_{3,4} = |S^3| - |S^4| = 1$. The waste when transitioning from $S^3$ to $S^4$ is computed as $W(S^3 \rightarrow S^4) = \sum_{n=1}^{3}(S^3|S^4| - \Delta_{3,4}) = (1 - 1) + (3 - 1) + (3 - 1) = 6$.

### III. SHIFTED CYCLIC ELASTIC TASK ALLOCATION SCHEMES WITH OPTIMAL TRANSITION WASTES

We first compute explicitly the transition waste of the cyclic elastic task allocation scheme introduced by Yang et al. [14] and then propose a shifted cyclic scheme that achieves the optimal transition waste among all such cyclic schemes. We assume that the number of machines $N$ lies in a predetermined interval $[L, N_{\text{max}}]$ and $N(N+1)F$ for every $L \leq N < N_{\text{max}}$.

A. Transition Waste of the Cyclic Elastic Task Allocation

First, we consider the case of one machine joining.

**Theorem 1.** The transition waste when transitioning from a cyclic $(N, L, F)$-TAS $S_N^N$ to a cyclic $(N+1, L, F)$-TAS $S_{N+1}^N$ (defined in (1)) is given below (assuming $N > L$).

$$W(S_N^N \rightarrow S_{N+1}^N) = (N - 1)F/(N + 1).$$

Now suppose that Machine $n^* \in [N]$ leaves the computation. We assume the system transitions to the cyclic TAS $S_{N-1}^N = \{S_N^N - 1, S_{n^*}^{n-1}, \ldots, S_{n^*}^{n-1}, \ldots, S_N^N\}$, where for $n < n^*$,

$$S_N^N = \left[\frac{(n-1)F}{N} - \frac{(n-1)F}{N} + \frac{LF}{N} - 1\right] \mod F,$$

and for $n > n^*$,

$$S_N^N = \left[\frac{(n-2)F}{N} - \frac{(n-2)F}{N} + \frac{LF}{N} - 1\right] \mod F.$$

**Theorem 2.** The transition waste when Machine $n^* \in [N]$ leaves and the system transitions from a cyclic $(N, L, F)$-TAS $S_N^N$ to a cyclic $(N-1, L, F)$-TAS $S_{N-1}^N$ (defined in (1)) is given as follows (assuming $N > L + 1$).

If $n^* < N - L$, $W_{n^*}(S_N^N \rightarrow S_{N-1}^N)$ is

$$\left((n^*-1)(n^*-2) + (N - L - n^*)((N - L - n^* + 1)\right) \frac{F}{N(N - 1)}.$$

If $n^* \geq N - L$, $W_{n^*}(S_N^N \rightarrow S_{N-1}^N)$ is

$$\left(n^* - 1)(n^* - 2) + \frac{N(N - 1)}{3N}\right) \frac{F}{N(N - 1)}.$$

Averaging $\alpha$ over $[N]$, the averaged transition waste when one machine leaves in the cyclic ETAS is

$$\left(\frac{N - 2}{3N} + \frac{(N - 2)(N - 1)(N - L)(N - L + 1)}{3(N - 1)N^2}\right) F.$$

176
B. Shifted Cyclic Scheme Achieving Optimal Transition Waste

From Theorem 1 and Theorem 2, the transition waste incurred across all existing machines in the cyclic ETAS proposed in [14] is $\frac{N-1}{N}F \approx F$ or $(\frac{N+1}{N} + \cdots)F \approx \frac{F}{N}$ tasks when a machine joins or leaves, respectively. In this section, we show that by applying a calculated shift, we can significantly reduce the transition waste of the cyclic ETAS.

**Definition 6 (Shifted cyclic task allocation).** For $\delta \in [F]$, a $\delta$-shifted cyclic $(N, L, F)$-TAS is given as follows.

$$S_{N,cyc}^0 = (S_1^0, \ldots, S_N^0),$$

where for $n \in [N]$,

$$S_n^0 = \left( \frac{n-1}{N}F + \delta, \frac{n-1}{N}F + \frac{LF}{N} - 1 + \delta \right) \pmod{F}.$$  

Given that the system transitions from a $\delta'$-shifted cyclic $(N, L, F)$-TAS to a $\delta$-shifted cyclic $(N', L, F)$-TAS, the question of interest is to determine $\delta$ that leads to a minimum transition waste. Our contribution is to derive the explicit formula of an optimal shift, which results in the minimum waste among certain shifted schemes.

**Theorem 3.** The transition waste when transitioning from a $\delta'$-shifted cyclic $(N, L, F)$-TAS $S_{N,cyc}^0$ to a $\delta$-shifted cyclic $(N + 1, L, F)$-TAS $S_{N+1,cyc}^0$ with $\delta = \delta' + \left(\frac{N+1}{N}\right)$ is

$$W(S_{N,cyc}^0 \rightarrow S_{N+1,cyc}^0) = \left(\frac{(N-L-1)(N-L-1)F}{2N(N+1)}\right), \text{ for odd } N - L,$$

$$\left(\frac{(N-L-1)^2F}{2N(N+1)}\right), \text{ for even } N - L.$$  

We note that the transition waste of the proposed shifted cyclic TAS is improved over that of the ordinary cyclic TAS ([14]) by a considerable factor of approximately $\frac{2N^2}{(N-L)}$, which is 8X when $L \approx N/2$. The improvement becomes even more significant when $L$ gets closer to $N$, e.g. in the order of $N^2$ when $N - L$ is small.

**Theorem 4.** The transition waste when transitioning from a $\delta'$-shifted cyclic $(N-1, L, F)$-TAS $S_{N-1,cyc}^0$ to a $\delta$-shifted cyclic $(N, L, F)$-TAS $S_{N,cyc}^0$ with $\delta = \delta' + (N-n^*) - \left(\frac{N+L-2}{2}\right)$, where Machine $n^*$ leaves, is

$$W(S_{N-1,cyc}^0 \rightarrow S_{N,cyc}^0) = \left(\frac{(N-L-1)^2F}{2(N-1)}\right), \text{ for odd } N - L,$$

$$\left(\frac{(N-L-1)(N-L-2)F}{2N(N-1)}\right), \text{ for even } N - L.$$  

Although we are able to show the optimality of our shifted cyclic ETASs only when the parameter $\delta$ satisfies a certain divisibility property (Theorem 5), we believe the optimality holds for every $\delta$, which was supported by an exhaustive search over small values of $N$ and $L$.

**Theorem 5.** The transition wastes stated in Theorem 3 and Theorem 4 are optimal among all choices of $\delta$-shifted cyclic TASs where $\frac{F}{N(N-1)}$ divides $\delta - \delta'$ and $\frac{F}{N(N-1)}$ divides $\delta' - \delta + N - n^*$, respectively.

IV. ZERO-WASTE ELASTIC TASK ALLOCATION SCHEMES

The shifted cyclic ETAS developed in Section III-B is easy to implement and has a negligible computation overhead at the master machine. However, in order to maintain the cyclic structure, the transitions incur a nontrivial transition waste, which can be linear in $F$. This drawback of the (shifted) cyclic ETAS motivated us to investigate elastic task allocation schemes with zero transition wastes. Our key findings include a necessary and sufficient condition for the existence of a zero-waste transition from an $(N, L, F)$-TAS to an $(N', L, F)$-TAS based on the famous Hall's marriage theorem and a construction of zero-waste ETAS based on finite geometry.

A. Zero-Waste Transition When One Machine Joins

It turns out that if the elastic events only consist of machines joining than it is easy to achieve zero-waste transitions.

**Proposition 1.** There always exists a zero-waste transition from an $(N, L, F)$-TAS to an $(N+1, L, F)$-TAS.

B. Zero-Waste Transition When One Machine Leaves

The case of one machine leaving, say Machine $n^*$, is more interesting. Theorem 6 provides a necessary and sufficient condition for the existence of a zero-waste transition from an $(N, L, F)$-TAS to an $(N-1, L, F)$-TAS no matter which machine leaves. Hall’s marriage theorem is used in the proof of this theorem. Recall that $\Delta_{N,N-1} = LF/(N-1)$.

**Theorem 6.** There exists a zero-waste transition from an $(N, L, F)$-TAS $S_N = (S_N^0, \ldots, S_N^N)$ to an $(N-1, L, F)$-TAS when Machine $n^*$ leaves for every $n^* \in [N]$ if and only if

$$|\cap_{n \in I} S_n^0| \leq (N - |I|)\Delta_{N,N-1},$$  

for every nonempty set $I \subseteq [N]$. Moreover, such a transition can be found in time $O((N - 1 + \frac{LF}{N})(N-1)(F(1 - \frac{1}{N}))$.

Theorem 6 provides us with an important insight: to make transitions with zero waste possible, we should assign to machines sets of tasks with small overlaps. This will be crucial in our construction of an ETAS with zero transition waste.

C. A Zero-Waste Elastic Task Allocation Scheme

So far we have discussed the case of a single machine leaving or joining. The more challenging question is how to allow (a possibly infinite) chain of such elastic events while guaranteeing zero-waste transitions. More specifically, we are interested in establishing a zero-waste range $[N_{\min}, N_{\max}] \subseteq [L, F]$ where the system can start with any number $N_0$ of machines, $N_0 \in [N_{\min}, N_{\max}]$, and then can transition with zero wastes an arbitrary number of times within this range, one machine leaving or joining at a time. We show the existence of a handful of such ranges in Theorem 7 and Corollary 1. We first need a formal definition of a zero-waste range.

**Definition 7 (Zero-waste range).** Given $L$ and $F$, a range $[N_{\min}, N_{\max}]$, where $L \leq N_{\min} \leq N_{\max} \leq F$ is called an $(L, F)$-zero-waste range $(L, F)$-ZWR if for every $N_0 \in [N_{\min}, N_{\max}]$ there exists an $(N_0, L, F)$-ETAS $(S_N^0, T)$ (see Definition 2) where the transition algorithm $T$ incurs a zero waste whenever the transition is within the range $[N_{\min}, N_{\max}]$.

Note that $N_{\min}$ and $N_{\max}$ are usually functions of $L$ and $F$. Also, the transition algorithm $T$ mentioned in Definition 2 and Definition 7 can be applied repeatedly to enable a chain of transitions within $N_{\min}$ and $N_{\max}$ machines by adding or removing one machine at a time.

**Lemma 1.** If there exists an $(N_{\max}, L, F)$-ETAS $(S_{N_{\max}}^0, T)$ so that $T$ always incurs a zero transition waste for every possible chain of $N_{\max} - N_{\min}$ transitions from $N_{\max}$ to $N_{\min}$ machines (machines leaving only) then $[N_{\min}, N_{\max}]$ is an $(L, F)$-ZWR.

Based on Lemma 1, we now describe our construction of $(L, F)$-ZWRs based on the so-called symmetric configurations from combinatorial designs.

**Definition 8 (Configuration [16]).** A $(v, b, k, r)$-configuration is an incident structure of $v$ points and $b$ lines such that (a) each line contains $k$ points, (b) each point lies on $r$ lines, and (c) two different points are connected by at most one line. If $v = b$ and, hence, $r = k$, the configuration is symmetric, denoted by $(v, k)$-configuration.
For example, the famous Fano plane is a $(7,3)$-configuration with seven points $\{1,2,3,\ldots,7\}$ and seven lines: $\{1,2,3\}$, $\{1,4,5\}$, $\{1,6,7\}$, $\{2,4,6\}$, $\{2,5,7\}$, $\{3,5,6\}$, and $\{3,4,7\}$.

We first show that an $(\Nmax, L)$-configuration can be used to construct an $(\Nmax, L, F)$-TAS with small pairwise overlaps and then present a method to establish an $[\Nmin, \Nmax]$-zero-waste range from such a TAS. Essentially, points correspond to tasks while lines correspond to sets of tasks. As there are $\Nmax$ points and $F$ tasks, it is natural to associate each point with $F/\Nmax$ tasks.

**Construction 1.** Suppose that $\Nmax$ divides $F$ and $B = \{B_1, \ldots, B_{\Nmax}\}$ is the set of $\Nmax$ lines of an $(\Nmax, L)$-configuration. An $(\Nmax, L, F)$-TAS $S^{\Nmax}$ can be constructed as follows. First, partition $[F] = \{0, \ldots, F-1\}$ into $\Nmax$ equal-sized parts $F_1, \ldots, F_{\Nmax}$. Then for each $n \in [\Nmax]$ we assign to Machine $n$ the index of the parts $F_p$'s corresponding to all points $p$ in the line $B_n$. In other words, we set $S^{\Nmax}_n := \cup_{p \in B_n} F_p$, for every $n \in [\Nmax]$. Fig. 2: A $(7,3,14)$-TAS constructed from the Fano plane. The table rows/columns correspond to the plane points/lines.

For instance, when there are $\Nmax = 7$ machines, $L = 3$, and $F = 14$ tasks, we first partition $[F]$ in seven parts: $F_1 = \{0,1\}$, $F_2 = \{2,3\}$, $F_3 = \{4,5\}$, $F_4 = \{6,7\}$, $F_5 = \{8,9\}$, $F_6 = \{10,11\}$, $F_7 = \{12,13\}$. Then, using the $(7,3,14)$-configuration (the Fano plane) in Construction 1, we obtain a $(7,3,14)$-TAS, represented by Fig. 2. For instance, Machine 1 is allocated the task set $S^1_1 = \{0,1,\ldots,5\} = F_1 \cup F_2 \cup F_3$, while Machine 2 has the task set $S^2_2 = \{0,1,6,7,8,9\} = F_1 \cup F_4 \cup F_5$. Clearly, each task is performed by $L = 3$ machines and each machine performs $LF/\Nmax = 6$ tasks.

**Lemma 2.** Construction 1 produces an $(\Nmax, L, F)$-TAS where every two task sets intersect at most $F/\Nmax$ tasks.

By Lemma 2, Construction 1 produces an initial $(\Nmax, L, F)$-TAS with small pairwise set overlaps. To show that $R$ machines can be removed one by one from this TAS with zero transition wastes, we first show that the pairwise intersections of the sets of intermediate TASs do not increase too much. Then, by using the pairwise intersection as an upper bound on the intersection of any set $I$ of task sets, $|I| \leq L$, we can guarantee that the intersections still satisfy the Hall-like condition in Theorem 6. By Lemma 1, zero-waste transitions will be possible within the range $[\Nmax - R, \Nmax]$.

**Theorem 7.** If there exists an $(\Nmax, L)$-configuration then there exists an $(\Nmax, L, F)$-TAS $S^{\Nmax}$ where $S^{\Nmax}_n \cap S^{\Nmax}_n' \subseteq F/\Nmax$, for every $n, n' \in [\Nmax]$, $n \neq n'$. This leads to the existence of an $(L, F)$-zero-waste range $[\Nmax - R, \Nmax]$ where

$$
R = 1 + \left\lfloor \frac{3L\Nmax - 2N_{max} - 2L + 1 - \sqrt{\Delta}}{4L - 2} \right\rfloor, \quad (3)
$$

$$
\Delta = L\Nmax(\Nmax^2 + 8L^2 - 16L + 6) + (2L - 1)^2. \quad (4)
$$

We assume here that $N \mid F$ for every $N \in [\Nmax - R, \Nmax]$. Equipped with Theorem 7, we now present a few explicit zero-waste ranges based on known results on configurations from the literature of combinatorial designs (see more in [15]).

**Corollary 1.** The following zero-waste ranges exist for all relevant $F$, that is, $F$ is divisible by $N(N - 1)$ for every $N \in [\Nmin, \Nmax]$.

1. $L = 3$, $\Nmax \geq 7$, $\Nmin = \Nmax = \frac{7\Nmax - 5 - \sqrt{\Delta}}{10} - 1$, where $\Delta = 9N_{max}^2 + 90N_{max} + 25$.
2. $L = 4$, $\Nmax \geq 13$, $\Nmin = \Nmax = \frac{10\Nmax - 7 - \sqrt{\Delta}}{14} - 1$, where $\Delta = 16N_{max}^2 + 280N_{max} + 49$.
3. $L = q$, $\Nmax = q^2$, for every prime power $q$.

Applying Corollary 1 to the case $L = 3$ and $\Nmax = 7$, we obtain a $(3, F)$-ZWR [5, 7] where the $(7,3,F)$-TAS corresponds to the Fano plane. In other words, zero-waste transitions are possible between five and seven machines when $L = 3$. Similarly, when applying the corollary to the case $L = 4$ and $\Nmax = 13$, we obtain a $(4, F)$-ZWR [9, 13], which implies that zero-waste transitions are possible between nine and thirteen machines. When $L = q$ and $\Nmax = q^2$, we obtain a $(q, F)$-ZWR $[\Nmin, \Nmax]$ where $\Nmin = \Theta(\Nmax^2/2)$, for every prime power $q$. Ideally, we would like to expand these ranges to $[\Nmin = L, \Nmax]$, which remains an open question.

**Acknowledgement**

We thank Yaoqing Yang for helpful discussions.

**References**

[1] K. Lee, M. Lam, R. Pellarini, D. Papailiopoulos, and K. Ramchandran, “Speeding up distributed machine learning using codes,” IEEE Trans. Inform. Theory, vol. 64, no. 3, pp. 1514–1529, 2018.

[2] S. Li, M. A. Maddah-Ali, and A. S. Avestimehr, “Coded MapReduce,” in Proc. Ann. Allerton Conf. Comm. Control Comput., 2015, pp. 964–971.

[3] 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.

[4] K. Huang and J. Abrahami, “Algorithm-based fault tolerance for matrix operations,” IEEE Trans. Comput., vol. C-33, no. 6, pp. 518–528, 1984.

[5] S. Dutta, V. Cadambe, and P. Grover, “Short-hot: Computing large linear transforms distributedly using coded short dot products,” in Adv. Neural Inform. Process. Syst., 2016, pp. 2100–2108.

[6] Q. Yu, M. A. Maddah-Ali, and A. S. Avestimehr, “Polyomino codes: an optimal design for high-dimensional coded matrix multiplication,” in Adv. Neural Inform. Process. Syst., 2017, pp. 4403–4413.

[7] C. Karakus, Y. Sun, S. Diggavi, and W. Yin, “Straggler mitigation in distributed optimization through data encoding,” in Adv. Neural Inform. Process. Syst., 2017, pp. 5434–5442.

[8] S. Li, M. A. Maddah-Ali, Q. Yu, and A. S. Avestimehr, “A fundamental tradeoff between computation and communication in distributed computing,” IEEE Trans. Inform. Theory, vol. 64, no. 1, pp. 109–128, 2018.

[9] A. Mallik, M. Chaudhari, and G. Joshi, “Rateless codes for near-perfect load balancing in distributed matrix-vector multiplication,” 2018. [Online]. Available: http://arxiv.org/abs/1804.10331

[10] J. Kosaraju, K. V. Rashmi, and S. Venkataraman, “Learning a code: Machine learning for approximate non-linear coded computation,” 2018. [Online]. Available: http://arxiv.org/abs/1806.01259

[11] Q. Yu, S. Li, N. Raviv, S. Kulan, M. Soljanin, and S. Avestimehr, “Lagrange coded computing: Optimal design for resiliency, security, and privacy,” in Proc. Mach. Learn. Res., vol. 89, 2019, pp. 1215–1225.

[12] Amazon EC2 Spot Instances. https://aws.amazon.com/ec2/spot/.

[13] Microsoft Azure Batch. https://azure.microsoft.com/en-au/services/batch/.

[14] Y. Yang, P. Grover, and S. Kar, “Coded elastic computing,” in Proc. IEEE Int. Symp. Inform. Theory, 2019, pp. 2654–2658.

[15] H. Dau, R. Gabrys, Y.-C. Huang, C. Feng, Q.-H. Luan, E. Alizahrahi, and Z. Tari, “Optimizing the transition waste in coded elastic computing,” 2019. [Online]. Available: https://arxiv.org/abs/1910.00796

[16] C. J. Colbourn and J. H. Dinitz, Handbook of Combinatorial Designs, Second Edition (Discrete Math. & Its Appl.). CRC Press, 2006.