Transition Waste Optimization for Coded Elastic Computing

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

***Abstract*—Distributed computing, in which a resource-intensive task is divided into subtasks and distributed among different machines, plays a key role in solving large-scale problems. *Coded computing* is a recently emerging paradigm where redundancy for distributed computing is introduced to alleviate the impact of slow machines (stragglers) on the completion time. We investigate coded computing solutions over elastic resources, where the set of available machines may change in the middle of the computation. This is motivated by recently available services in the cloud computing industry (e.g., EC2 Spot, Azure Batch) where low-priority virtual machines are offered at a fraction of the price of the on-demand instances but can be preempted on short notice. Our contributions are three-fold. We first introduce a new concept called *transition waste* that quantifies the number of tasks existing machines must abandon or take over when a machine joins/leaves. We then develop an efficient method to minimize the transition waste for the cyclic task allocation scheme recently proposed in the literature (Yang *et al.* ISIT'19). Finally, we establish a novel solution based on finite geometry achieving *zero* transition wastes given that the number of active machines varies within a fixed range.**

I. Introduction

In the era of Big Data, massive computational tasks, e.g. in large-scale machine learning and data analytics, are often carried out in *distributed systems* like Apache Spark [1] and Hadoop [2], which can efficiently process terabytes or even petabytes of data. However, it has been observed in such systems that slow machines, or *stragglers*, which may run 6x-8x slower than a median one, may significantly affect the performance of the whole distributed system [3]–[5].

Coded distributed computing [6]–[8], built upon algorithmic fault tolerance [9], is a recently emerging paradigm where computation redundancy is employed to tackle the straggler effect. As a toy example [6], to perform a matrix-vector multiplication $\boldsymbol{A}\boldsymbol{x}$, a master machine first partitions the matrix $\boldsymbol{A}$ into two equal-sized submatrices $\boldsymbol{A}_1$ and $\boldsymbol{A}_2$ and then distributes $\boldsymbol{A}_1$, $\boldsymbol{A}_2$, and $\boldsymbol{A}_1 + \boldsymbol{A}_2$ to three worker machines, respectively. These machines also receive the vector $\boldsymbol{x}$ and perform three multiplications $\boldsymbol{A}_1\boldsymbol{x}$, $\boldsymbol{A}_2\boldsymbol{x}$, and $(\boldsymbol{A}_1 + \boldsymbol{A}_2)\boldsymbol{x}$ in parallel. Clearly, $\boldsymbol{A}\boldsymbol{x}$ 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 through a substantial body of work in the literature, e.g., [10]–[14]. Recent breakthroughs have shown that this paradigm works for not only linear/bilinear operations but also general nonlinear operations such as polynomial evaluation [15] or even any function that can be represented by a deep network [16], [17].

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* [18] and *Microsoft Azure Batch* [19], 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 standard 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.* [20] 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 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. In other words, their solution allows an elastic task allocation: when a new machine joins, existing machines can share some of their workload with the newcomer, reducing the number of tasks allocated to them; likewise, when a machine leaves, existing machines must cover extra tasks left over. The elastic coded computing scheme proposed in [20] 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

Son Hoang Dau and Zahir Tari are with the School of Computing Technologies, RMIT University. Emails: {sonhoang.dau, zahir.tari}@rmit.edu.au. Ryan Gabrys is with the University of California, San Diego. Email: gabrys@ucsd.edu. Yu-Chih Huang is with the Institute of Communications Engineering, National Yang Ming Chiao Tung University. Email: jerryhuang@nycu.edu.tw). Chen Feng is with the School of Engineering, British Columbia University (Okanagan Campus). Email: chen.feng@ubc.ca. Quang-Hung Luu is with the Department of Computing Technologies, School of Science, Computing and Engineering Technologies, Swinburne University of Technology, and also with the Department of Civil Engineering, Monash University. Emails: hluu@swin.edu.au/hung.luu@monash.edu. Eidah Alzahrani is with Albaha University. Email: ejalzahrani1@bu.edu.sa. Part of this work was presented at the IEEE International Symposium on Information Theory, 2020.

linear regression by up to 46% compared to ordinary coded computing schemes. The elastic setting was later extended to cover machines with heterogeneous computation speeds and storage capacities in [21]–[23]. A combination with hierarchical coding was also proposed in [24], which allows finer task allocations to speed up the completion time.

Relaxing the cyclic task allocation proposed in [20], we investigate a more general *elastic task allocation* problem, which we believe may find applications not just in coded distributed computing but also in a much broader context where a set of tasks is distributed to an elastic set of participants (e.g., virtual machines), which frequently leave and join. More specifically, 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? These combinatorial constraints can be easily satisfied, e.g., by using the cyclic scheme employed in [20] or its generalization.
- *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 have to abandon or take over when one machine joins or leaves, minus the necessary amount? As an optimization problem, this is a much more challenging question compared to task allocation and is the focus in our work.

We illustrate in a toy example (Fig. 1) the *transition waste* concept and explain why the cyclic elastic task allocation scheme in [20] is *suboptimal* with respect to this new metric. We consider the computation of $\boldsymbol{A}\boldsymbol{x}$ where $\boldsymbol{A}$ can be partitioned into 40 equal-sized sub-matrices $\boldsymbol{A}_1, \ldots, \boldsymbol{A}_{40}$. We first group these sub-matrices into 20 groups, e.g., $\{\boldsymbol{A}_1, \boldsymbol{A}_2\}$, $\{\boldsymbol{A}_3, \boldsymbol{A}_4\}$, and so forth. Then each group is assigned a *task index* from 0 to 19. Task 0, for instance, corresponds to the computation of $\{\boldsymbol{A}_1\boldsymbol{x}, \boldsymbol{A}_2\boldsymbol{x}\}$. Task 0 is *encoded* into five subtasks: $\boldsymbol{A}_1\boldsymbol{x}$, $\boldsymbol{A}_2\boldsymbol{x}$, $(\boldsymbol{A}_1 + \boldsymbol{A}_2)\boldsymbol{x}$, $(\boldsymbol{A}_1 + 2\boldsymbol{A}_2)\boldsymbol{x}$, and $(\boldsymbol{A}_1 + 3\boldsymbol{A}_2)\boldsymbol{x}$. *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 requirement can be met by using the cyclic scheme in [20]: 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 each, 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 $S_1^5 = \{0, 1, \ldots, 11\} = \{0, \ldots, 3\} \cup \{4, \ldots, 7\} \cup \{8, \ldots, 11\}$, Machine 2 works on $S_2^5 = \{4, 5, \ldots, 15\}$, and so on (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.

S_1^5	S_2^5	S_3^5	S_4^5	S_5^5
$0 \to 3$			$0 \to 3$	$0 \to 3$
$4 \to 7$	$4 \to 7$			$4 \to 7$
$8 \to 11$	$8 \to 11$	$8 \to 11$		
	$12 \to 15$	$12 \to 15$	$12 \to 15$	
		$16 \to 19$	$16 \to 19$	$16 \to 19$

(a) Cyclic task allocation for five machines [20].

S_1^4	S_2^4	S_3^4	S_4^4
$0 \to 4$		$0 \to 4$	$0 \to 4$
$5 \to 9$	$5 \to 9$		$5 \to 9$
$10 \to 14$	$10 \to 14$	$10 \to 14$	
	$15 \to 19$	$15 \to 19$	$15 \to 19$

(b) Cyclic task allocation for four machines [20]. The *transition waste* from five to four machines is *12 tasks*.

S_1^4	S_2^4	S_3^4	S_4^4
$17 \to 1$		$17 \to 1$	$17 \to 1$
$2 \to 6$	$2 \to 6$		$2 \to 6$
$7 \to 11$	$7 \to 11$	$7 \to 11$	
	$12 \to 16$	$12 \to 16$	$12 \to 16$

(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 [20] with respect to the *transition waste* when Machine 5 leaves. Here, we use $a \to b$ to denote the set $\{a, a+1, \ldots, b\} \pmod F$, where $F = 20$.

In Fig. 1 (b), only four machines are available, each of which takes 15 tasks. As Machine 5 has left, it is necessary now that each of the four available machines must take $3 = 15 - 12$ more tasks. Ideally, when the transition from five machines to four machines occurs, each machine continues their existing tasks and works on three new tasks. This is true for Machine 1 because $S_1^5 \subset S_1^4$. 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 $(0 \to 4)$. 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

$$(0+3-3)+(1+4-3)+(2+5-3)+(3+6-3) = 12 \text{ (tasks)},$$

Therefore, sticking to the cyclic allocation scheme of [20], 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 $S_n^5 \subset S_n^4$, 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^5 and S_n^4, $n = 1, \ldots, 4$.

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 elastic task allocation scheme introduced by Yang *et al.* [20] 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 optimal transition waste of a shifted cyclic scheme is, in general, 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.

While the cyclic schemes are simple to implement and efficient when there are many tasks and many machines, the schemes with zero-waste transitions are more suitable when there are a moderate number of machines and tasks but each task is resource-intensive. We will discuss this further in Sections II.

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 (see Section II-C). The readers who are familiar with the *parity declustering* technique in redundant disk arrays (RAID) [25]–[27] may recognize the analogy between a coded computing scheme and a stripe unit and between a task allocation scheme and a data layout (in the terminology of [26]).

The paper is organized as follows. The concepts of elastic task allocation and transition waste are defined and discussed in Section II. Section III is devoted for the cyclic task allocation scheme and our proposed shifted version with optimal transition wastes. We develop elastic task allocation schemes that admit zero transition wastes in Section IV, perform simulations and experiments in Section V, and conclude the paper in Section VI.

II. Preliminaries

A. Elastic Task Allocation Scheme

We define in this section the *elastic task allocation scheme*, which generalizes the cyclic scheme originally proposed by Yang *et al.* [20], and the new concept of the *transition waste*. Frequently used notations can also be found in Appendix VII-E.

We henceforth 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 $[[F]] \triangleq \{0, 1, \dots, F-1\}$. We assume that all tasks consume an equal amount of resources (storage, memory, CPU). We use $[F]$ to denote the set $\{1, 2, \dots, F\}$ and $[A, B]$ to denote the set $\{A, A+1, \dots, B\}$. We also use $2^{[[F]]}$ to denote the power set of the set $\{0, 1, \dots, F-1\}$ and $(2^{[[F]]})^N = 2^{[[F]]} \times 2^{[[F]]} \times \cdots \times 2^{[[F]]}$ to denote the N-ary Cartesian power of $2^{[[F]]}$.

Definition 1 (Task allocation scheme)**.** *An ordered list of N sets $\mathcal{S}^N = (S_1^N, \dots, S_N^N) \in (2^{[[F]]})^N$, 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 $\mathcal{S}^N$, and*
- *(Load Balancing) $|S_n^N| = LF/N$ for all $n \in [N]$. Here we assume that $LF/N \in \mathbb{Z}$.*

Note that we can relax the Load Balancing property and require that $S_n^N \in \{\lfloor LF/N \rfloor, \lceil LF/N \rceil\}$ and hence can lift the requirement that N divides LF. To simplify the exposition, however, we assume $LF/N \in \mathbb{Z}$. In practice, padding of dummy tasks can be employed to achieve this property. The L-Redundancy property is tied to the underlying coded computing scheme (see Section II-C).

An (N, L, F)-TAS $\mathcal{S}^N = (S_1^N, \dots, S_N^N)$ can also be represented by its *incidence matrix* $\boldsymbol{B} = (b_{f,n})_{F \times N}$, where $b_{f,n} = 1$ if and only if $f \in S_n^N$. The rows and columns of $\boldsymbol{B}$ represent tasks and machines, respectively. Clearly, $\boldsymbol{B}$ has row weight L and column weight LF/N. In other words, each row of $\boldsymbol{B}$ has precisely L ones while each column has precisely LF/N ones. Thus, a TAS simply corresponds to a binary matrix with constant row and column weights.

Example 1. For $N = 3, L = 2, F = 6$, the list of sets

$$\mathcal{S}^3 = (\{0, 1, 2, 3\}, \{2, 3, 4, 5\}, \{4, 5, 0, 1\})$$

is a $(3, 2, 6)$-TAS as each member set has size $4 = 2 \times 6/3$ and each element $f \in [[6]] = \{0, 1, \dots, 5\}$ belongs to precisely $L = 2$ such sets. The incident matrix of $\mathcal{S}^3$, given by (1), has column weight four and row weight two.

$$\begin{array}{c} \\ \\ \text{Task } 0 \\ \text{Task } 1 \\ \text{Task } 2 \\ \text{Task } 3 \\ \text{Task } 4 \\ \text{Task } 5 \end{array} \begin{array}{c} \begin{array}{ccc} \text{Machine 1} & \text{Machine 2} & \text{Machine 3} \\ S_1^3 & S_2^3 & S_3^3 \end{array} \\ \left(\begin{array}{ccc} 1 & 0 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \\ 0 & 1 & 1 \end{array} \right) \end{array}. \tag{1}$$

When a machine leaves or joins, we need to 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 Section II-C how to couple an ETAS and a coded computing scheme to achieve a coded elastic computing scheme that tolerates stragglers. Note that the parameter L in Definition 2 is specified by the underlying coded computing scheme and considered fixed in an elastic task allocation scheme while the number of machines N and the number of tasks F can vary.

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

$$\mathcal{T} \colon (2^{[[F]]})^N \times \{-1, 1\} \times [N] \to (2^{[[F]]})^{N-1} \cup (2^{[[F]]})^{N+1}$$

takes as input an (N, L, F)-TAS $\mathcal{S}^N$, 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, $\mathcal{T}$ returns an output $\mathcal{S}^{N'}$, 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, $\mathcal{T}$ updates the list of task sets $\mathcal{S}^N$ to obtain $\mathcal{S}^{N'}$, 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 and joining only. In other words, we assume that machines leave and join one after another and not at the same time. This not only allows us to avoid complex mathematical notations but also covers the case of multiple machines leaving/joining: we can treat that case as a series of independent transitions in each of which only one machine leaves or joins. *Second*, while in general we allow N to take any value in the range $[L, LF]$, it is more practical to limit N within a fixed range $[L, N_{\mathsf{max}}]$. Moreover, we often assume that F is divisible by any number within this range. These assumptions allow us to achieve concrete results and are also practically reasonable. For instance, we can use padding, i.e., adding dummy tasks, to make F satisfy the aforementioned property. *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 [20]. A simple way to construct an ETAS is to let $\mathcal{T}$ 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 always use a fixed (N, L, F)-TAS

$$\begin{aligned} \mathcal{S}^N_{\mathsf{cyc}} &= (S_1^N, \ldots, S_N^N), \\ S_n^N &= \left[(n-1)\frac{F}{N}, (n-1)\frac{F}{N} + \frac{LF}{N} - 1\right] \pmod{F}, \end{aligned} \tag{2}$$

for every $n \in [N]$, where $[A, B] \pmod{F}$ is obtained from $[A, B]$ by applying the modulo operation on every element of this set. We also assume here that $F/N \in \mathbb{Z}$.

It is straightforward to verify that each $\mathcal{S}^N_{\mathsf{cyc}}$ satisfies the Load Balancing and the L-Redundancy properties, and therefore, is indeed an (N, L, F)-TAS. The reallocation algorithm is trivial: $\mathcal{T}(\mathcal{S}^N_{\mathsf{cyc}}, 1) = \mathcal{S}^{N+1}_{\mathsf{cyc}}$ and $\mathcal{T}(\mathcal{S}^N_{\mathsf{cyc}}, -1, n^*) = \mathcal{S}^{N-1}_{\mathsf{cyc}}$ 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 $n^* = 5$.

B. 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 $\mathcal{S}^N$ to another (N', L, F)-TAS $\mathcal{S}^{N'}$, $\Delta_{N,N'} \triangleq |LF/N - LF/N'|$ is referred to as the necessary load change. When $N' = N \pm 1$, we have $\Delta_{N,N\pm1} = LF/(N(N \pm 1))$.*

The *necessary load change*, $\Delta_{N,N'} = ||S_n^N| - |\mathcal{S}_n^{N'}||$, 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-Balancing 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 has to take $3 = 15 - 12$ more tasks to react to this event. The necessary load change is *three* 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 $\mathcal{S}_n^{N'}$ is defined as*

$$W(S_n^N \to \mathcal{S}_n^{N'}) = |S_n^N \Delta \mathcal{S}_n^{N'}| - \Delta_{N,N'},$$

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 $W_{n^*}(S_n^N \to \mathcal{S}_n^{N'})$ for the case Machine n^* 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 $\mathcal{S}_n^{N'}$ 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 has to 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 $W(S_n^N \to \mathcal{S}_n^{N'})$ in Definition 4 measures the maximum number of tasks wasted at Machine n when another machines 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 $\mathcal{S}^N$ to an $(N+1, L, F)$-TAS $\mathcal{S}^{N+1}$ is defined as*

$$W(\mathcal{S}^N \to \mathcal{S}^{N+1}) \triangleq \sum_{n \in [N]} W(S_n^N \to S_n^{N+1}).$$

When Machine n^* leaves, the transition waste of the transition from an (N, L, F)-TAS $\mathcal{S}^N$ to an $(N-1, L, F)$-TAS $\mathcal{S}^{N-1}$ is defined as

$$W_{n^*}(\mathcal{S}^N \to \mathcal{S}^{N-1}) \triangleq \sum_{n \in [N] \setminus \{n^*\}} W_{n^*}(S_n^N \to S_n^{N-1}).$$

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

We demonstrated in Fig. 1(a), (b), (c)) in Section I two different transitions from a $(5, 3, 20)$-TAS to a $(4, 3, 20)$-TAS, i.e., one machine removed. The first transition has a transition waste of 12 tasks, while the second one has a zero waste. Another example, built upon Example 1, is given below.

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

$$\begin{aligned} W(\mathcal{S}^3 \to \mathcal{S}^4) &= \sum_{n=1}^{3}(|S_n^3 \Delta S_n^4| - \Delta_{3,4}) \\ &= (1-1) + (5-1) + (3-1) = 6. \end{aligned}$$

C. Coupling an Elastic Task Allocation Scheme and a Coded Computing Scheme

We now explain how to couple an elastic task allocation scheme (ETAS) and a coded computing scheme (CCS) to achieve a coded elastic computing scheme, which allows

- *straggler tolerance*: at most E slow machines do not affect the completion time of the system,
- *load balancing*: every available machine is assigned the same workload,
- *elasticity*: the workload of available machines can be flexibly adjusted when machines leave and join.

The general method is to first partition the problem instance into F *independent* sub-instances and then apply a CCS to *each* sub-instance. Each task $f \in [[F]]$ refers to the computation task performed over the fth sub-instance. Suppose that throughout the computation the number of available machines varies from L to N_{max}. For each task, a CCS generates N_{max} sub-tasks, which are distributed to maximum N_{max} machines so that the completion of any $L - E$ sub-tasks leads to the completion of the task ($L - E$ is referred to as the *recovery threshold*). Each of the N available machines must be loaded with the corresponding sub-tasks of *all* F sub-instances so as to be ready to work on any new tasks when machines leave or join. However, each machine only works on the sub-tasks of the tasks assigned to it by the TAS. More specifically, if an (N, L, F)-TAS $\mathcal{S}^N = \{S_1^N, \ldots, S_n^N\}$ is used then Machine n only works on tasks indexed by S_n^N.

The L-Redundancy of the TAS guarantees that any task f is worked on by precisely L different machines among N. As the CCS allows the recovery of Task f from any $L-E$ outputs, the coded elastic computing scheme, which couples a TAS and a CCS, can tolerate E stragglers. The Load Balancing property of the TAS guarantees that every available machine is assigned the same workload. When a machine joins or leaves, a new TAS constructed by the transition algorithm $\mathcal{T}$ of the ETAS is applied, which preserves the straggler tolerance and the load balancing property. We discuss below how to define the tasks for the *matrix-vector multiplication* problem. For other related problems such as matrix-matrix multiplication, linear regression, and multivariate polynomial evaluation please refer to Appendix VII-A.

Matrix-Vector Multiplication. We aim to compute $\boldsymbol{Ax}$, where $\boldsymbol{A}$ is a matrix and $\boldsymbol{x}$ is a vector of matching dimension, in a way that tolerates any E stragglers ($0 \le E < L$), and with a varied number of available machines N ($L \le N \le N_{\mathsf{max}}$).

Assuming that the number of rows of $\boldsymbol{A}$ is divisible by F (padding if necessary), we partition $\boldsymbol{A}$ row-wise into F equal-sized sub-matrices $\boldsymbol{A}_0, \boldsymbol{A}_1, \ldots, \boldsymbol{A}_{F-1}$. The pair $(\boldsymbol{A}_f, \boldsymbol{x})$ forms the fth sub-instance of the original instance $(\boldsymbol{A}, \boldsymbol{x})$ and the computation of $\boldsymbol{A}_f\boldsymbol{x}$ is referred to as Task f. A known CCS for matrix-vector multiplication (e.g., [6]) can then be used to generate N_{max} sub-tasks for each Task f, each of which is then distributed to the corresponding machine (machines joining later download later). Clearly, the completion of all tasks $f \in [[F]]$ gives us the desired product $\boldsymbol{Ax}$.

D. Storage, Communication, and Computation Overhead of an ETAS

As proposed in [20], each machine stores all F tasks but only runs a subset of those tasks based on the specific allocation. In this way, when switching to a new TAS, each existing machine doesn't have to download new data. When coupling with a coded computing scheme (see Section II-C), each machine actually stores only a $1/(L-E)$-fraction of the input data, e.g., the matrix $\boldsymbol{A}$ if we are computing $\boldsymbol{Ax}$, where $E < L$ is the number of stragglers the scheme can tolerate.

Every machine joining the system has to download its portion of data once, which constitutes the most costly, but necessary, communication overhead of the system. From a practical perspective, letting a machine *joining* in the middle of the computation might be costly as it must download its allocated tasks from existing servers or from the master (decoding/re-encoding may even be needed). Machines leaving, on the other hand, would not cause any issue in terms of communication overhead because the active ones already stored all needed data and are ready to transition to any new sets of tasks. The communication between a master machine, which coordinates the task allocation, and the worker machines, is negligible.

The master has to run an algorithm to find a new TAS whenever a machine leaves or joins. If a cyclic or a shifted cyclic ETAS (see Section III-B) is used, the computation overhead is negligible. If a zero-waste transition (see Section IV) is insisted, the complexity of the search is polynomial in N, L, and F (basically, it runs a network flow algorithm). A zero-waste transition will be particularly beneficial when there are a moderate number of tasks while each task is resource-intensive. In that case, the benefit of a zero-waste transition will offset the time spent for finding one. Note that we have total control of the number of tasks F when defining tasks (only requiring that F satisfies some divisibility condition). For example, when computing the matrix-matrix multiplication $\boldsymbol{AB}$ (see Appendix VII-A), we can partition both $\boldsymbol{A}$ and $\boldsymbol{B}$ into a desirable number F of sub-matrices $\boldsymbol{A}_f$ and $\boldsymbol{B}_f$ (row-wise for $\boldsymbol{A}$ and column-wise for $\boldsymbol{B}$). Then, Task f corresponds to $\boldsymbol{A}_f\boldsymbol{B}_f$, $f = 0, 1, \ldots, F-1$.

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.* [20] (Theorems 1, 2) and then propose a shifted cyclic scheme that achieves the optimal transition waste among all such cyclic schemes (Theorems 3, 4, 5). We assume that the number of machines N lies in a predetermined interval $[L, N_{\mathsf{max}}]$ and $N(N+1)$ divides F for every $L \leq N < N_{\mathsf{max}}$.

A. Transition Waste of the Cyclic Elastic Task Allocation

The following lemma is useful in determining the symmetric difference between two sets in $[[F]]$.

Lemma 1. *Let $S = [a, b] \pmod F$ and $T = [c, d] \pmod F$. Assume that $0 \leq a \leq c < F$, and moreover, $0 < |S| < F$ and $0 < |T| < F$. The following statements hold.*

(a) *If $c - a < |S| < (c-a) + |T| < F$ then*

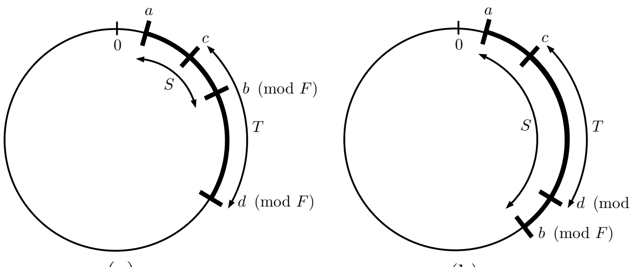
$$|S \Delta T| = 2(c-a) + |T| - |S|.$$

(b) *If $|S| \geq (c-a) + |T|$ then $T \subset S$ and*

$$|S \Delta T| = |S| - |T|.$$

Proof. **(a)** Suppose that $c - a < |S| < (c-a) + |T| < F$. If we travel along the circle of integers mod F (see Fig. 2 (a)) clockwise from a, we first see c, then $b \pmod F$ (because $c - a < |S|$), then $d \pmod F$ (because $|S| < (c-a) + |T|$), before we reach a again (because $(c{-}a){+}|T| < F$). Therefore,

$$\begin{aligned} |S \Delta T| &= |S \setminus T| + |T \setminus S| \\ &= (c-a) + (|T| - (|S| - (c-a))) \\ &= 2(c-a) + |T| - |S|. \end{aligned}$$



(a) (b)

Fig. 2: Illustrations of the two sets $S = [a, b] \pmod F$ and $T = [c, d] \pmod F$ on the circle of integers mod F.

(b) Suppose that $|S| \geq (c-a) + |T|$. This clearly implies that $T \subset S$ and hence $|S \Delta T| = |S| - |T|$ (see Fig. 2 (b)). ■

Lemma 2 identifies the case where the zero waste is achieved, which is obvious by the definition of the transition waste.

Lemma 2. *The transition waste incurred at Machine n when transitioning from a set of tasks S_n^N to another set of tasks $\mathcal{S}_n^{N'}$ is zero if and only if $S_n^N \subset \mathcal{S}_n^{N'}$ or $S_n^N \supset \mathcal{S}_n^{N'}$.*

In the next corollary, we show that when there are $N = L + 1$ machines and one machine leaves or when there are $N = L$ machines and one machine joins, the transition waste is trivially zero, no matter which TASs the system are employing.

Corollary 1. *The transition waste when transitioning from an (L, L, F)-TAS to an $(L{+}1, L, F)$-TAS and vice versa are zero.*

Proof. Note that for an (L, L, F)-TAS $\mathcal{S}^L = (S_1^L, \ldots, S_L^L)$, we have $S_n^L = [[F]]$ for all $n \in [L]$. Therefore, $S_n^N \supset S_n^{N'}$. By Lemma 2, the corollary follows. ■

We henceforth assume that $N > L$ when one machine joins and $N > L + 1$ when one machine leaves. First, we consider the case of one machine *joining*. Theorem 1 establishes the transition waste of the cyclic task allocation scheme.

Theorem 1. *The transition waste when transitioning from a cyclic (N, L, F)-TAS $\mathcal{S}_{\mathsf{cyc}}^N$ to a cyclic $(N{+}1, L, F)$-TAS $\mathcal{S}_{\mathsf{cyc}}^{N+1}$ (defined in (2)) is given below (assuming $N > L$).*

$$W(\mathcal{S}_{\mathsf{cyc}}^N \to \mathcal{S}_{\mathsf{cyc}}^{N+1}) = \frac{N-1}{N+1} F.$$

Proof. We prove this theorem by performing a direct computation of the symmetric difference of the sets of tasks allocated before and after the transition. Suppose Machine $N+1$ joins the computation. According to (2), we have

$$\mathcal{S}_{\mathsf{cyc}}^N = (S_1^N, \ldots, S_N^N) \text{ and } \mathcal{S}_{\mathsf{cyc}}^{N+1} = (S_1^{N+1}, \ldots, S_{N+1}^{N+1}),$$

where for $n \in [N]$,

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

and for $n \in [N+1]$,

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

We now apply Lemma 1 to find the symmetric difference of S_n^N and S_n^{N+1} for every $n \in [n]$. We write

$$S = S_n^{N+1} = [a, b] \pmod F, \quad T = S_n^N = [c, d] \pmod F$$

and can verify that all assumptions of Lemma 1 (a) are satisfied. Indeed, since $N > L$ and $N \geq n \geq 1$, we have

$$0 \leq a = (n-1)\frac{F}{N+1} \leq c = (n-1)\frac{F}{N} < F,$$

$$0 < |S| = \frac{LF}{N+1} < F, \quad 0 < |T| = \frac{LF}{N} < F,$$

$$c - a = (n-1)\frac{F}{N(N+1)} < \frac{LF}{N+1} = |S|,$$

$$|S| = \frac{LF}{N+1} < (n{-}1)\frac{F}{N(N+1)} {+} \frac{LF}{N} = (c{-}a){+}|T| < F.$$

Therefore, by Lemma 1 (a),

$$\begin{aligned} |S_n^N \Delta S_n^{N+1}| &= 2(c-a) + (|S_n^N| - |S_n^{N+1}|) \\ &= \frac{2(n-1)F}{N(N+1)} + \frac{LF}{N(N+1)} \\ &= \frac{2(n-1)F}{N(N+1)} + \Delta_{N,N+1}. \end{aligned}$$

Thus, the transition waste incurred at Machine n is

$$W(S_n^N \to S_n^{N+1}) = |S_n^N \Delta S_n^{N+1}| - \Delta_{N,N+1} = \frac{2(n-1)F}{N(N+1)}.$$

Finally, the transition waste when transitioning from $\mathcal{S}_{\mathsf{cyc}}^N$ to $\mathcal{S}_{\mathsf{cyc}}^{N+1}$ is

$$\begin{aligned} W(\mathcal{S}_{\mathsf{cyc}}^N \to \mathcal{S}_{\mathsf{cyc}}^{N+1}) &= \sum_{n\in[N]} W(S_n^N \to S_n^{N+1}) \\ &= \sum_{n\in[N]} \frac{2(n-1)F}{N(N+1)} = \frac{N-1}{N+1}F, \end{aligned}$$

as desired. ■

We now turn to the slightly more involved case when one machine leaves the computation. When Machine $n^* \in [N]$ leaves, for the ease of notation, we assume the system transitions to the cyclic TAS

$$\mathcal{S}_{\mathsf{cyc}}^{N-1} = \{S_1^{N-1}, \dots, S_{n^*-1}^{N-1}, S_{n^*+1}^{N-1}, \dots, S_N^{N-1}\},$$

where for $n < n^*$,

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

and for $n > n^*$,

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

Lemma 3. *Suppose that Machine $n^* \in [N]$ leaves and the system transitions from a cyclic (N, L, F)-TAS $\mathcal{S}_{\mathsf{cyc}}^N$ to a cyclic $(N-1, L, F)$-TAS $\mathcal{S}_{\mathsf{cyc}}^{N-1}$ (defined in (2)). The transition waste incurred at Machine n for $n < n^*$ is (assuming $N > L+1$)*

$$W_{n^*}(S_n^N \to S_n^{N-1}) = \frac{2(n-1)F}{N(N-1)}.$$

Proof. The proof is the same as that of Theorem 1, whereby Lemma 1 (a) is applied to $S = S_n^N$ and $T = S_n^{N-1}$. ■

Lemma 4. *Suppose that Machine $n^* \in [N]$ leaves and the system transitions from a cyclic (N, L, F)-TAS $\mathcal{S}_{\mathsf{cyc}}^N$ to a cyclic $(N-1, L, F)$-TAS $\mathcal{S}_{\mathsf{cyc}}^{N-1}$ (defined in (2)). The transition waste incurred at Machine n for $N \geq n \geq n^*+1$ is given below (assuming $N > L+1$).*

If $n^* \geq N-L$ then $W_{n^*}(S_n^N \to S_n^{N-1}) = 0$.

If $n^* < N-L < n$ then $W_{n^*}(S_n^N \to S_n^{N-1}) = 0$.

If $n^* < n \leq N-L$ then

$$W_{n^*}(S_n^N \to S_n^{N-1}) = \frac{2(N-L-n+1)F}{N(N-1)}.$$

Proof. See Appendix VII-B. ■

Theorem 2 dertermines the transition waste for the cyclic task allocation scheme when one machine *leaves*.

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

If $n^* < N-L$, $W_{n^*}(\mathcal{S}_{\mathsf{cyc}}^N \to \mathcal{S}_{\mathsf{cyc}}^{N-1})$ is

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

If $n^* \geq N-L$, $W_{n^*}(\mathcal{S}_{\mathsf{cyc}}^N \to \mathcal{S}_{\mathsf{cyc}}^{N-1})$ is

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

Averaging n^* over $[N]$, the averaged transition waste when one machine leaves in the cyclic ETAS is

$$\begin{aligned} &W_{\mathsf{avg}}(\mathcal{S}_{\mathsf{cyc}}^N \to \mathcal{S}_{\mathsf{cyc}}^{N-1}) \\ &\quad = \left(\frac{N-2}{3N} + \frac{(N-L-1)(N-L)(N-L+1)}{3(N-1)N^2}\right)F. \end{aligned}$$

Proof. If $n^* < N-L$, by Lemma 3 and Lemma 4, we have

$$\begin{aligned} &W_{n^*}(\mathcal{S}_{\mathsf{cyc}}^N \to \mathcal{S}_{\mathsf{cyc}}^{N-1}) \\ &= \sum_{n=1}^{n^*-1} \frac{2(n-1)F}{N(N-1)} + \sum_{n=n^*+1}^{N-L} \frac{2(N-L-n+1)F}{N(N-1)} + \sum_{n=N-L+1}^{N} 0 \\ &= \big((n^*-1)(n^*-2) + (N-L-n^*)(N-L-n^*+1)\big)\frac{F}{N(N-1)}. \end{aligned}$$

Similarly, when $n^* \geq N-L$, we obtain

$$\begin{aligned} W_{n^*}(\mathcal{S}_{\mathsf{cyc}}^N \to \mathcal{S}_{\mathsf{cyc}}^{N-1}) &= \sum_{n=1}^{n^*-1} \frac{2(n-1)F}{N(N-1)} + \sum_{n=n^*+1}^{N} 0 \\ &= (n^*-1)(n^*-2)\frac{F}{N(N-1)}. \end{aligned}$$

Averaging $W_{n^*}(\mathcal{S}_{\mathsf{cyc}}^N \to \mathcal{S}_{\mathsf{cyc}}^{N-1})$ over all $n^* \in [N]$ we obtain the stated formula for $W_{\mathsf{avg}}(\mathcal{S}_{\mathsf{cyc}}^N \to \mathcal{S}_{\mathsf{cyc}}^{N-1})$. ■

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 [20] is $\frac{N-1}{N+1}F \approx F$ or $(\frac{N-2}{3N} + \cdots)F \approx \frac{F}{3}$ 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.

As mentioned earlier, the updated TAS used by the cyclic ETAS [20] (see Section II) only depends on the number of machines available and not on the current TAS, which is one reason that leads to the scheme's poor transition waste. We now generalize the cyclic TAS to *shifted* cyclic TAS in order to allow a more adaptive transition that takes into account the current TAS.

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

$$\mathcal{S}_{\delta\text{-}\mathsf{cyc}}^N = (S_1^N, \dots, S_N^N),$$

where for $n \in [N]$,

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

Note that there are F different shifted TASs possible corresponding to F different values of δ. When $\delta = 0$, the shifted cyclic TAS reduces to an ordinary cyclic TAS (Section II).

Given that the system transitions from an δ'-shifted cyclic (N, L, F)-TAS to a δ-shifted cyclic (N', L, F)-TAS, the question of interest is to determine δ that leads to a minimum transition waste. We note here that the master machine can always exhaustively examine all possible F shifted schemes and find the one with the smallest waste. However, this will take the master roughly $LF^2 = FN\frac{LF}{N}$ operations, which is time-consuming for large F. *Our contribution* is to derive the explicit formula of an *optimal shift*, which results in the *minimum waste* among all F shifted schemes. We first tackle the case of one machine joining and then argue that the case of one machine leaving follows by symmetry.

Theorem 3 computes the transition waste for a particular shifted cyclic task allocation scheme when one machine *joins*. The amount of shift $\delta - \delta'$ given in the theorem will be partially proved to be optimal in Theorem 5.

Theorem 3. *The transition waste when transitioning from a δ'-shifted cyclic (N, L, F)-TAS $\mathcal{S}^N_{\delta'\text{-cyc}}$ to a δ-shifted cyclic $(N+1, L, F)$-TAS $\mathcal{S}^{N+1}_{\delta\text{-cyc}}$ with $\delta = \delta' + \lfloor \frac{N+L-1}{2} \rfloor \frac{F}{N(N+1)}$ is*

$$W(\mathcal{S}^N_{\delta'\text{-cyc}} \to \mathcal{S}^{N+1}_{\delta\text{-cyc}}) = \begin{cases} \frac{(N-L-1)(N-L+1)F}{2N(N+1)}, & \textit{for odd } N-L, \\ \frac{(N-L)^2 F}{2N(N+1)}, & \textit{for even } N-L. \end{cases}$$

Proof of Theorem 3. See Appendix VII-C. ■

By comparing the formulas derived in Theorem 1 and Theorem 3, we deduce that the transition waste of the proposed shifted cyclic TAS when a machine joins is improved over that of the ordinary cyclic TAS ([20]) by a considerable factor of approximately $\frac{2N^2}{(N-L)^2}$, 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 determines the transition waste for a particular shifted cyclic task allocation scheme when one machine *leaves*. The amount of shift $\delta - \delta'$ given in the theorem will be proved to be optimal in Theorem 5 under a divisibility condition.

Theorem 4. *The transition waste when transitioning from a δ'-shifted cyclic (N, L, F)-TAS $\mathcal{S}^N_{\delta'\text{-cyc}}$ to a δ-shifted cyclic $(N-1, L, F)$-TAS $\mathcal{S}^{N-1}_{\delta\text{-cyc}}$ with $\delta = \delta' + \big((N - n^*) - \lfloor \frac{(N+L-2)}{2} \rfloor\big) \frac{F}{N(N-1)}$, where Machine n^* leaves, is*

$$W(\mathcal{S}^N_{\delta'\text{-cyc}} \to \mathcal{S}^{N-1}_{\delta\text{-cyc}}) = \begin{cases} \frac{(N-L-1)^2 F}{2N(N-1)}, & \textit{for odd } N-L, \\ \frac{(N-L)(N-L-2)F}{2N(N-1)}, & \textit{for even } N-L. \end{cases}$$

Proof. The proof works by symmetry. By treating Machine n^* that leaves as the machine that joins the system in Theorem 3 and replacing N by $N-1$, we obtain the claimed formula for the transition wastes. Note that because the task sets can be cyclically shifted along the circle of integers mod F, the index of the machine that leaves does not matter. This phenomenon, however, does not apply to the ordinary cyclic ETAS. ■

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

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

Proof. By symmetry, we just need to prove this for the case of machines joining. We first derive a formula of the transition waste for every δ and then show that it is minimized within the specified range of δ. See Appendix VII-D for more details. ■

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. Indeed, to coordinate a transition, the master just needs to inform each machine its updated index, the number of active machines, and the amount of shift required. However, to maintain the cyclic structure, the transitions incur a nontrivial transition waste. This quantity can scale linear in F, which is the maximum number of tasks each machine can take, and hence may significantly increase the computation overhead at each machine. Moreover, high transition wastes also mean more new tasks than necessary must be downloaded if each machine does not already store all the tasks from the beginning, which leads to higher communication overhead.

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

By Lemma 2, the transition waste incurred at Machine n when transitioning from the set of tasks S^N_n to another set $\mathcal{S}^{N'}_n$ is zero if and only if $S^N_n \subset S^{N'}_n$ or vice versa. It turns out that if the elastic events consist of only machines joining then 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.*

Proof. To achieve a zero-waste transition when Machine $N+1$ joins, each existing machine (from 1 to N) can simply choose a subset of $\frac{LF}{N(N+1)}$ tasks to pass to Machine $N+1$, which will then have in total $N\frac{LF}{N(N+1)}$ tasks. The requirement is to have these N sets disjoint. We can achieve this by letting each machine n from 1 to N choose an arbitrary subset of S^N_n of size $\frac{LF}{N(N+1)}$ that does not intersect any sets chosen by previous machines so far. This is always possible because Machine n has enough tasks in its set to do the selection:

$$|S^N_n| = \frac{LF}{N} \geq (n-1)\frac{LF}{N(N+1)} + \frac{LF}{N(N+1)}.$$

This completes the proof. ■

Note that this proposition is a stand-alone result and will not be used in the rest of the paper.

B. Zero-Waste Transition When One Machine Leaves

The case of one machine leaving, say Machine n^*, is more challenging. Note that to achieve a zero-waste transition, due to Lemma 2, it is necessary and sufficient to let other machines keep their current sets of tasks while reallocating the tasks from the leaving machine to them (so that $S^N_n \subset S^{N-1}_n$). Reallocating one task from Machine n^* to a machine n corresponds to selecting one edge in the *transition graph* (Definition 7 below). Note that when the transition happens, both L and F are fixed. This means that each task from the leaving machine needs to be reallocated to exactly *one* active machine to maintain the L-Redundancy, which requires that each task is performed by exactly L machines (see Definition 1). We will see later that reallocating all tasks turns out to correspond to a "matching" in that graph (Lemma 5).

Definition 7. *Given an (N, L, F)-TAS $\mathcal{S}^N = (S_1^N, \dots, S_N^N)$, the transition graph $\mathcal{G}_{n^*}$ is the bipartite graph with vertex set $U_{n^*} \cup V_{n^*}$, where $U_{n^*} = [N] \setminus \{n^*\}$ and $V_{n^*} = S_{n^*}^N$ and there is an edge (u, v), $u \in U_{n^*}$, $v \in V_{n^*}$, if and only if $v \in \overline{S_u^N} \triangleq [[F]] \setminus S_u^N$.*

Note that the set V_{n^*} of the transition graph represents the tasks from the leaving machine n^* that need to be reallocated to other machines, while an edge (u, v) implies that the task $v \in V_{n^*}$ can be taken over by Machine u, i.e., this machine was not allocated this task before the transition. An example of such a graph is given below.

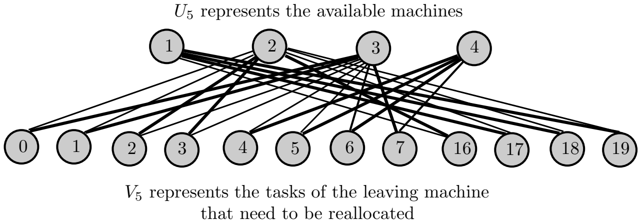

U_5 represents the available machines
1
2
3
4
0
1
2
3
4
5
6
7
16
17
18
19
V_5 represents the tasks of the leaving machine that need to be reallocated

Fig. 3: Illustration of the transition graph $\mathcal{G}_5$ in Example 3. An edge (u, v) means the task v from the leaving machine can be taken over by Machine u because Machine u was not allocated this task before the transition.

Example 3. When $N = 5$, $L = 3$, and $F = 20$, we consider the (N, L, F)-TAS given in Fig. 1 (a), $\mathcal{S}^5 = (S_1^5, \dots, S_5^5)$, where $S_1^5 = \{0, \dots, 11\}$, $S_2^5 = \{4, \dots, 15\}$, $S_3^5 = \{8, \dots, 19\}$, $S_4^5 = \{0, \dots, 3, 8, \dots, 19\}$, and $S_5^5 = \{0, \dots, 7, 16, \dots, 19\}$. Suppose that Machine 5 leaves, i.e., $n^* = 5$. Then the transition graph $\mathcal{G}_5$ is illustrated in Fig. 3.

A subset $\mathcal{M}$ of edges of a bipartite graph $\mathcal{G}$ with vertex set (U, V) is referred to as a *perfect Δ-matching* of $\mathcal{G}$ if each vertex in V is incident to precisely one edge in $\mathcal{M}$ while each vertex in U is incident to precisely Δ edges in $\mathcal{M}$.

Lemma 5. *There exists a zero-waste transition from an (N, L, F)-TAS $\mathcal{S}^N$ to an $(N-1, L, F)$-TAS $\mathcal{S}^{N-1}$ when Machine n^* leaves if and only if the transition graph $\mathcal{G}_{n^*}$ admits a perfect $\Delta_{N,N-1}$-matching.*

Proof. Recall that due to Lemma 2, the transition has a zero-waste if and only if $S_n^{N-1} \subset S_n^N$ for every $n \in [N] \setminus \{n^*\}$. This means that we need to reallocate tasks left over by Machine n^* to other $N-1$ machines by adding these new tasks to the existing task sets of these machines.

It is evident that a way to reallocate $\frac{LF}{N}$ tasks from Machine n^* to $N-1$ other machines corresponds precisely to a perfect $\Delta_{N,N-1}$-matching of the transition graph $\mathcal{G}_{n^*}$: each task, which corresponds to a vertex $v \in V_{n^*}$, is reallocated to exactly *one* machine, which corresponds to a vertex $u \in U_{n^*}$; moreover, each machine is allocated precisely $\Delta_{N,N-1} = \frac{LF}{N(N-1)}$ new tasks, which shows that each vertex u is incident to precisely $\Delta_{N,N-1}$ edges while each vertex v is incident to exactly one edge in the matching. ■

For instance, the zero-waste transition presented in Fig. 1 (a)(c) corresponds to the following perfect 3-matching of $\mathcal{G}_5$ (thicker edges in Fig. 3):

$$\begin{aligned}\mathcal{M} = \{&(1,17),(1,18),(1,19),(2,2),(2,3),\\ &(2,16),(3,0),(3,1),(3,7),(4,4),(4,5),(4,6)\}.\end{aligned}$$

Based on this matching, each machine 1, 2, 3, and 4 is allocated three new tasks from the leaving Machine 5. Moreover, every task from Machine 5, i.e., $\{0,1,2,3,4,5,6,7,16,17,18,19\}$, is reallocated to exactly one machine.

The following lemma is a straightforward corollary of Hall's marriage theorem.

Lemma 6. *A bipartite graph $\mathcal{G}$ with the vertex set $U \cup V$ has a perfect $\Delta_{N,N-1}$-matching if and only if the inequality*

$$|\cup_{n \in J} \Gamma_{\mathcal{G}}(n)| \geq |J| \Delta_{N,N-1}, \tag{3}$$

holds for every nonempty set $J \subseteq U$, where $\Gamma_{\mathcal{G}}(n)$ denotes the set of neighbors of n in $\mathcal{G}$.

Proof. The celebrated Hall's marriage theorem [28] states that a bipartite graph $\mathcal{G}$ with the vertex set (U, V) has a perfect matching (or, perfect 1-matching, in our notation), if and only if for every nonempty set $J \subseteq U$, it holds that $|\cup_{n \in J} \Gamma_{\mathcal{G}}(n)| \geq |J|$, where $\Gamma_{\mathcal{G}}(n)$ denotes the set of neighbors of n in $\mathcal{G}$. By duplicating each vertex of U and its incident edges $\Delta_{N,N-1}$ times and applying Hall's theorem to the resulting bipartite graph, we deduce that $\mathcal{G}$ has a perfect $\Delta_{N,N-1}$-matching if and only if (3) holds for every nonempty set $J \subseteq U$. ■

As a corollary of Lemma 5 and Lemma 6, we obtain a necessary and sufficient condition for the existence of a zero-waste transition when one particular machine leaves.

Corollary 2. *There exists a zero-waste transition from an (N, L, F)-TAS $\mathcal{S}^N$ to an $(N-1, L, F)$-TAS $\mathcal{S}^{N-1}$ when Machine n^* leaves if and only if the following inequality holds for every nonempty set $J \subseteq [N] \setminus \{n^*\}$.*

$$|(\cup_{n \in J} \overline{S_n^N}) \cap S_{n^*}^N| \geq |J| \Delta_{N,N-1}. \tag{4}$$

Proof. The conclusion is straightforward from Lemma 5 and Lemma 6 and the following observation: by the definition of the transition matrix $\mathcal{G}_{n^*}$, the set of neighbours of a vertex $n \in U_{n^*} = [N] \setminus \{n^*\}$ in $\mathcal{G}_{n^*}$ is $\Gamma_{\mathcal{G}_{n^*}}(n) = \overline{S_n^N} \cap S_{n^*}^N$. ■

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. Essentially, it states that as long as the sets of tasks of different machines do not overlap too much then there exists a zero-waste transition. Recall that $\Delta_{N,N-1} = \frac{LF}{N(N-1)}$.

Theorem 6. *There exists a zero-waste transition from an (N, L, F)-TAS $\mathcal{S}^N = (S_1^N, \dots, S_N^N)$ to an $(N-1, L, F)$-TAS when Machine n^* leaves for every $n^* \in [N]$ if and only if*

$$|\cap_{n \in I} S_n^N| \leq (N - |I|) \Delta_{N,N-1}, \tag{5}$$

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

Proof. Let $\mathcal{G}_{n^*}$ be the transition graph of an (N, L, F)-TAS $\mathcal{S}^N$ with the vertex set (U_{n^*}, V_{n^*}) (Definition 7). By Corollary 2, it suffices to show that the inequality (4) holds for every nonempty set $J \subseteq [N] \setminus \{n^*\}$ and for every $n^* \in [N]$ if and only if (5) holds for every nonempty set $I \subseteq [N]$.

Suppose that (4) holds as stated. Note that

$$\begin{aligned}|(\cup_{n \in J} \overline{S_n^N}) \cap S_{n^*}^N| &= |\overline{\cap_{n \in J} S_n^N} \cap S_{n^*}^N| = |S_{n^*}^N \setminus \cap_{n \in J} S_n^N| \\ &= |S_{n^*}^N| - |\cap_{n \in J \cup \{n^*\}} S_n^N|.\end{aligned}$$

Therefore, (4) is equivalent to

$$|\cap_{n \in J \cup \{n^*\}} S_n^N| \leq |S_{n^*}^N| - |J| \Delta_{N,N-1}.$$

Setting $I = J \cup \{n^*\}$, this is also equivalent to

$$\begin{aligned}|\cap_{n \in I} S_n^N| &\leq |S_{n^*}^N| - (|I| - 1)\Delta_{N,N-1} \\ &= (N-1)\Delta_{N,N-1} - (|I| - 1)\Delta_{N,N-1} \\ &= (N - |I|)\Delta_{N,N-1}.\end{aligned}$$

Note that as n^* varies over $[N]$ and J varies over all nonempty subsets of $[N] \setminus \{n^*\}$, $I = J \cup \{n^*\}$ varies over all subsets of $[N]$ of size at least two. Furthermore, (5) holds trivially (with equality) when $|I| = 1$. Therefore, (5) holds for all nonempty sets $I \subseteq [N]$. Hence, we settle the *only if* direction. As all steps are equivalent transformations, the *if* direction is also true. The complexity of finding a zero-waste transition comes from that of a network flow algorithm [29] employed to find a perfect matching for $\mathcal{G}_{n^*}$. This completes the proof. ■

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 in the next section.

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* $[N_{\mathsf{min}}, N_{\mathsf{max}}] \subset [L, F]$ where the system can start with any number N_0 of machines, $N_0 \in [N_{\mathsf{min}}, N_{\mathsf{max}}]$, 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 3. We first need a formal definition of a zero-waste range.

Definition 8 (Zero-waste range). *Given L and F, a range $[N_{\mathsf{min}}, N_{\mathsf{max}}]$, where $L \leq N_{\mathsf{min}} \leq N_{\mathsf{max}} \leq F$ is called an (L, F)-zero-waste range ((L, F)-ZWR) if for every $N_0 \in [N_{\mathsf{min}}, N_{\mathsf{max}}]$ there exists an (N_0, L, F)-ETAS $(\mathcal{S}^{N_0}, \mathcal{T})$ (see Definition 2) where the transition algorithm $\mathcal{T}$ incurs a zero waste whenever the transition is within the range $[N_{\mathsf{min}}, N_{\mathsf{max}}]$.*

Note that N_{min} and N_{max} are usually functions of L and F. Also, the transition algorithm $\mathcal{T}$ mentioned in Definitions 2 and 8 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. It turns out that if we can construct an (N_0, L, F)-ETAS $(\mathcal{S}^{N_0}, \mathcal{T})$ so that $\mathcal{T}$ incurs a zero transition waste within $[N_{\mathsf{min}}, N_{\mathsf{max}}]$ for *some* $N_0 \in [N_{\mathsf{min}}, N_{\mathsf{max}}]$ then we can also construct an (N_0', L, F)-ETAS satisfying the same property for *every* $N_0' \in [N_{\mathsf{min}}, N_{\mathsf{max}}]$, i.e., $[N_{\mathsf{min}}, N_{\mathsf{max}}]$ is an (L, F)-ZWR. In particular, we show that this claim is true when $N_0 = N_{\mathsf{max}}$.

Lemma 7. *If there exists an (N_{max}, L, F)-ETAS $(\mathcal{S}^{N_{\mathsf{max}}}, \mathcal{T})$ so that $\mathcal{T}$ always incurs a zero transition waste for every possible chain of $N_{\mathsf{max}} - N_{\mathsf{min}}$ transitions from N_{max} to N_{min} machines (machines leaving only) then $[N_{\mathsf{min}}, N_{\mathsf{max}}]$ is an (L, F)-ZWR.*

Before proving this lemma, we need the concept of a *transition tree*, which keeps track of all the possible *states* the system can be at and the transitions leading to them from the original state, where a state consists of the list of machines available and the corresponding TAS. The transition tree is, in fact, an explicit way to represent an ETAS.

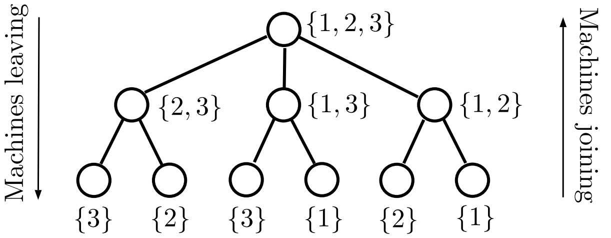


Fig. 4: Illustration of a transition tree when $N_{\mathsf{min}} = 1$ and $N_{\mathsf{max}} = 3$. The set of available machines is given at each node (we omit the TAS associated with each node).

Definition 9 (Transition tree). *Given an (N_{max}, L, F)-ETAS $(\mathcal{S}^{N_{\mathsf{max}}}, \mathcal{T})$ satisfying the assumption of Lemma 7, the corresponding transition tree $\mathfrak{T}$ is a rooted tree created as follows. The root node of the tree consists of the set $[N_{\mathsf{max}}]$ and the corresponding (N_{max}, L, F)-TAS. Other nodes can be created in a recursive manner. Suppose that a node u is already created that consists of a set of indices $\mathcal{I}$ and an $(|\mathcal{I}|, L, F)$-TAS. If $|\mathcal{I}| > N_{\mathsf{min}}$, the $|\mathcal{I}|$ child nodes of u can be created as follows. Each child node v corresponds to the removal of one machine indexed by $n^* \in \mathcal{I}$ and consists of the list $\mathcal{J} \triangleq \mathcal{I} \setminus \{n^*\}$ and a $(|\mathcal{J}|, L, F)$-TAS obtained by applying the transition algorithm $\mathcal{T}$ to the $(|\mathcal{I}|, L, F)$-TAS of u.*

For instance, when $N_{\mathsf{max}} = 3$, $N_{\mathsf{min}} = L = 1$, we have a transition tree illustrated in Fig. 4.

Proof of Lemma 7. Based on the transition tree, it is easy to see that once the system can start from an (N_{max}, L, F)-TAS and transition with zero wastes down to an (N_{min}, L, F)-TAS in all possible ways then we can also start from any intermediate (N_0, L, F)-TAS, $N_0 \in [N_{\mathsf{min}}, N_{\mathsf{max}}]$, and transition with zero wastes within this range. Indeed, if one machine leaves and the system is currently at a state corresponding to a node in the tree, then it can transition to a child node depending on which node is leaving. Vice versa, if one machine joins, the system can transition to the state stored at the parent node. ■

Remark 3 (Overhead incurred by the transition tree)**.** As shown in the proof of Lemma 7, the transition tree is used to keep track of all zero-waste transitions possible within the range $[N_{\mathsf{min}}, N_{\mathsf{max}}]$. The entire tree can be created once by the master machine before the computation session starts or can be created on the fly. The tree has height $N_{\mathsf{max}} - N_{\mathsf{min}}$ and a total of $1 + \sum_{h=1}^{N_{\mathsf{max}}-N_{\mathsf{min}}} \prod_{i=0}^{h-1}(N - i)$ nodes, which is in the order of $N!$. To create a child node, a network flow algorithm is invoked to find the zero-waste transition (however, the computation required becomes lighter when it gets closer to the leaves). The creation and storage of the transition tree incurs significant storage and computation overheads at the master node, and therefore, using the tree is beneficial when we have relatively small N and F and intensive tasks so that having zero transition waste pays off. Maintaining a zero-waste ETAS with lower overheads remains an open question for future research.

Based on Lemma 7, we now describe our construction of (L, F)-ZWRs based on the so-called *symmetric configurations* from combinatorial designs.

Definition 10 (Configuration [30]). *A (v, b, k, r)-configuration is an incident structure of v points and b lines such that*

- *each line contains k points,*
- *each point lies on r lines, and*
- *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.

The famous Fano plane is a $(7, 3)$-configuration with seven points $\{1, 2, \ldots, 7\}$ and seven lines: $\{1, 2, 3\}$, $\{1, 4, 5\}$, $\{1, 6, 7\}$, $\{2, 4, 6\}$, $\{2, 5, 7\}$, $\{3, 5, 6\}$, and $\{3, 4, 7\}$ (Fig. 5).

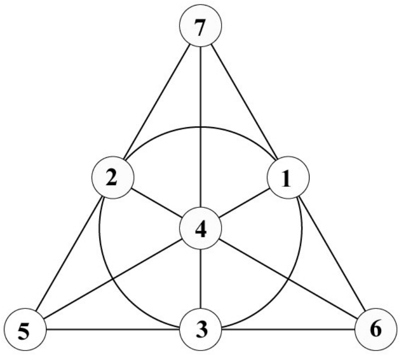


Fig. 5: A Fano plane with seven points and seven lines.

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

Construction 1. Suppose that N_{max} divides F and $\mathcal{B} = \{B_1, \ldots, B_{N_{\mathsf{max}}}\}$ is the set of N_{max} lines of an (N_{max}, L)-configuration. An (N_{max}, L, F)-TAS $\mathcal{S}^{N_{\mathsf{max}}}$ can be constructed as follows. First, partition $[[F]] = \{0, \ldots, F - 1\}$ into N_{max} equal sized parts $F_1, \ldots, F_{N_{\mathsf{max}}}$. Then for each $n \in [N_{\mathsf{max}}]$ we assign to Machine n the tasks indexed by the parts F_p's corresponding to all points p in the line B_n. In other words, we set $S_n^{N_{\mathsf{max}}} := \cup_{p \in B_n} F_p$, for every $n \in [N_{\mathsf{max}}]$.

For instance, when there are $N_{\mathsf{max}} = 7$ machines, $L = 3$, and $F = 14$ tasks, we first partition $[[F]]$ in to 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 $(7, 3)$-configuration (the Fano plane) in Construction 1, we obtain a $(7, 3, 14)$-TAS, represented by Fig. 6. For instance, Machine 1 is allocated the task set $S_1^7 = \{0, 1, \ldots, 5\} = F_1 \cup F_2 \cup F_3$, while Machine 2 has the task set $S_2^7 = \{0, 1, 6, 7, 8, 9\} = F_1 \cup F_4 \cup F_5$. Clearly, each task is performed by $L = 3$ machines and each machine performs $LF/N_{\mathsf{max}} = 6$ tasks.

S_1^7	S_2^7	S_3^7	S_4^7	S_5^7	S_6^7	S_7^7
$\{0,1\}$	$\{0,1\}$	$\{0,1\}$				
$\{2,3\}$			$\{2,3\}$	$\{2,3\}$		
$\{4,5\}$					$\{4,5\}$	$\{4,5\}$
	$\{6,7\}$		$\{6,7\}$			$\{6,7\}$
	$\{8,9\}$			$\{8,9\}$	$\{8,9\}$	
		$\{10,11\}$	$\{10,11\}$		$\{10,11\}$	
		$\{12,13\}$		$\{12,13\}$		$\{12,13\}$

Fig. 6: A $(7, 3, 14)$-TAS constructed from the Fano plane. The table rows/columns correspond to the plane points/lines.

Since every two lines in a configuration either don't intersect or intersect at only one point, the resulting TAS also has small pairwise intersections, which is crucial for our construction of a zero-waste range.

Lemma 8. *Construction 1 produces an (N_{max}, L, F)-TAS where every two task sets intersect at at most F/N_{max} tasks.*

Proof. According to Construction 1, each set of task has size

$$|S_n^{N_{\mathsf{max}}}| = |B_n| \frac{F}{N_{\mathsf{max}}} = \frac{LF}{N_{\mathsf{max}}}.$$

Moreover, as each point p in the configuration belongs to exactly L lines, each task also belongs to precisely L task sets. Hence, the resulting $\mathcal{S}^{N_{\mathsf{max}}}$ is indeed an (N_{max}, L, F)-TAS. Moreover, since every two lines in the configuration intersect at at most one point, every two task sets $S_n^{N_{\mathsf{max}}}$ and $\mathcal{S}_{n'}^{N}$, $n \neq n'$, intersect at at most F/N_{max} tasks as claimed. ∎

Note that the *expected* cardinality[1] of the intersection of two *random* subsets of cardinality LF/N of $[[F]] = \{0, 1, \ldots, F - 1\}$ is $\frac{L^2}{N}\frac{F}{N}$, which is approximately F/N for $L^2 \approx N$. Therefore, a random construction doesn't provide smaller (expected) pairwise intersections than Construction 1.

[1] Indeed, as each point in a set of size F belongs to a random subset of size $\frac{LF}{N}$ with probability $\frac{L}{N}$, the probability that a point belongs to two independent random subsets of size $\frac{LF}{N}$ is $\frac{L^2}{N^2}$. This implies that the expected size of the intersection of the two random subsets of that size is $\frac{L^2}{N^2}F = \frac{L^2}{N}\frac{F}{N}$.

By Lemma 8, Construction 1 produces an initial (N_{max}, 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, $|I| \le L$, we can guarantee that the intersections still satisfy the Hall-like condition in Theorem 6. As a consequence, zero-waste transitions will be possible within the range $[N_{\mathsf{max}} - R, N_{\mathsf{max}}]$.

Theorem 7. *If there exists an (N_{max}, L)-configuration then there exists an (N_{max}, L, F)-TAS $\mathcal{S}^{N_{\mathsf{max}}} = (S_1^{N_{\mathsf{max}}}, \dots, S_{N_{\mathsf{max}}}^{N_{\mathsf{max}}})$ where*

$$|S_n^{N_{\mathsf{max}}} \cap S_{n'}^{N_{\mathsf{max}}}| \le \frac{F}{N_{\mathsf{max}}},$$

for every $n, n' \in [N_{\mathsf{max}}]$, $n \ne n'$. This leads to the existence of an (L, F)-zero-waste range $[N_{\mathsf{max}} - R, N_{\mathsf{max}}]$ where

$$R \triangleq 1 + \left\lfloor \frac{(3LN_{\mathsf{max}} - 2N_{\mathsf{max}} - 2L + 1) - \sqrt{\Delta}}{4L - 2} \right\rfloor \tag{6}$$

and

$$\Delta = LN_{\mathsf{max}}(LN_{\mathsf{max}} + 8L^2 - 16L + 6) + (2L - 1)^2. \tag{7}$$

We assume here that N divides F for every $N \in [N_{\mathsf{max}} - R, N_{\mathsf{max}}]$, and $L \ge 1$ and $N_{\mathsf{max}} \ge 2$.

Proof. The first statement is due to Lemma 8. We now prove the second statement, assuming that there exists an (N_{max}, L, F)-TAS as specified. Thanks to Lemma 7, it suffices to show that for every $1 \le r < R$, after removing any r machines one after another, the resulting $(N_{\mathsf{max}} - r, L, F)$-TAS still admits a zero-waste transition when one more machine leaves. Equivalently, we aim to show that this TAS satisfies the Hall-like condition (5).

Suppose that $r < R$ machines have been removed with r zero-waste transitions and $\mathcal{S}^{N_{\mathsf{max}}-r} = (S_1^{N_{\mathsf{max}}-r}, \dots, S_{N_{\mathsf{max}}-r}^{N_{\mathsf{max}}-r})$ is the resulting $(N_{\mathsf{max}} - r, L, F)$-TAS. Let I be a nonempty subset of indices of $|I|$ machines among the remaining ones. Note that when $|I| = 1$ or $|I| > L$, the inequality (5) is trivially satisfied. Indeed, when $|I| = 1$, the equality is achieved. When $|I| > L$, as each task cannot belong to more than L task sets, the intersection of $|I|$ task sets is empty and hence, (5) holds trivially. We henceforth assume $2 \le |I| \le L$. Suppose $n, n' \in I$, $n \ne n'$. Note that whenever there is a zero-waste transition from an (N, L, F)-TAS to an $(N-1, L, F)$-TAS, each machine keeps its current task set and also takes $\Delta_{N,N-1}$ extra tasks. Hence, the intersection of a pair of task sets is increased by at most $2\Delta_{N,N-1}$ tasks. Therefore,

$$\begin{aligned}
&|\cap_{i\in I} S_i^{N_{\mathsf{max}}-r}| \le |S_n^{N_{\mathsf{max}}-r} \cap S_{n'}^{N_{\mathsf{max}}-r}| \\
&\le |S_n^{N_{\mathsf{max}}} \cap S_{n'}^{N_{\mathsf{max}}}| + \sum_{j=0}^{r-1} 2\Delta_{N_{\mathsf{max}}-j, N_{\mathsf{max}}-j-1} \\
&= |S_n^{N_{\mathsf{max}}} \cap S_{n'}^{N_{\mathsf{max}}}| + \sum_{j=0}^{r-1} \left(\frac{2LF}{N_{\mathsf{max}} - j - 1} - \frac{2LF}{N_{\mathsf{max}} - j} \right) \\
&\le \frac{F}{N_{\mathsf{max}}} + \left(\frac{2LF}{N_{\mathsf{max}} - r} - \frac{2LF}{N_{\mathsf{max}}} \right) \\
&= \frac{F}{N_{\mathsf{max}}} + \frac{2LFr}{(N_{\mathsf{max}} - r)N_{\mathsf{max}}}.
\end{aligned}$$

Therefore, in order to show that (5) holds for the $(N_{\mathsf{max}} - r, L, F)$-TAS $\mathcal{S}^{N_{\mathsf{max}}-r}$, that is,

$$|\cap_{i\in I} S_i^{N_{\mathsf{max}}-r}| \le (N_{\mathsf{max}} - r - |I|)\Delta_{N_{\mathsf{max}}-r, N_{\mathsf{max}}-r-1},$$

as we assume $|I| \le L$, it suffices to show that

$$\frac{F}{N_{\mathsf{max}}} + \frac{2LFr}{(N_{\mathsf{max}} - r)N_{\mathsf{max}}} \le (N_{\mathsf{max}} - r - L)\Delta_{N_{\mathsf{max}}-r, N_{\mathsf{max}}-r-1},$$

or equivalently,

$$\begin{aligned}
&\frac{F}{N_{\mathsf{max}}} + \frac{2LFr}{(N_{\mathsf{max}} - r)N_{\mathsf{max}}} \\
&\qquad \le (N_{\mathsf{max}} - r - L)\frac{LF}{(N_{\mathsf{max}} - r)(N_{\mathsf{max}} - r - 1)}.
\end{aligned} \tag{8}$$

Simplifying (8), we obtain

$$\begin{aligned}
&(2L-1)r^2 - (3LN_{\mathsf{max}} - 2N_{\mathsf{max}} - 2L + 1)r \\
&\qquad + N_{\mathsf{max}}(LN_{\mathsf{max}} - N_{\mathsf{max}} - L^2 + 1) \ge 0.
\end{aligned} \tag{9}$$

The left-hand side of (9) can be regarded as a quadratic polynomial in r, which has two positive roots

$$\frac{(3LN_{\mathsf{max}} - 2N_{\mathsf{max}} - 2L + 1) \pm \sqrt{\Delta}}{4L - 2},$$

where Δ is given as in (7). This is because when $L \ge 1$ and $N_{\mathsf{max}} \ge 2$, we have

$$\Delta = LN_{\mathsf{max}}\big((LN_{\mathsf{max}} - 2) + 8(L-1)^2\big) + (2L-1)^2 > 0,$$

and also, the coefficient of r^2 is $2L - 1 > 0$, the coefficient of r is negative, and the free coefficient is non-negative:

$$\begin{aligned}
&N_{\mathsf{max}}(LN_{\mathsf{max}} - N_{\mathsf{max}} - L^2 + 1) \\
&\qquad = N_{\mathsf{max}}(L-1)(N_{\mathsf{max}} - L - 1) \ge 0.
\end{aligned}$$

Therefore, $R = 1 + \lfloor r_1 \rfloor \ge 1$, where r_1 is the smaller (positive) root of (9). Moreover, when

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

the left-hand side of (9) is non-negative, which implies that this inequality holds. Therefore, we have shown that for every $r < R$ defined as in (6), the inequality (5) holds for the $(N_{\mathsf{max}} - r, L, F)$-TAS in consideration. Hence, there is a zero-waste transition from this TAS to an $(N_{\mathsf{max}} - r - 1, L, F)$-TAS. Thus, $[N_{\mathsf{max}} - R, N_{\mathsf{max}}]$ is an (L, F)-zero-waste range. ■

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.

Corollary 3. *The following zero-waste ranges exist for all relevant F, that is, F is divisible by $N(N-1)$ for every $N \in [N_{\mathsf{min}}+1, N_{\mathsf{max}}]$.*

1) *$L = 3$, $N_{\mathsf{max}} \geq 7$, $N_{\mathsf{min}} = N_{\mathsf{max}} - \left\lfloor \frac{7N_{\mathsf{max}}-5-\sqrt{\Delta}}{10} \right\rfloor - 1$, where $\Delta = 9N_{\mathsf{max}}^2 + 90N_{\mathsf{max}} + 25$.*
2) *$L = 4$, $N_{\mathsf{max}} \geq 13$, $N_{\mathsf{min}} = N_{\mathsf{max}} - \left\lfloor \frac{10N_{\mathsf{max}}-7-\sqrt{\Delta}}{14} \right\rfloor - 1$, where $\Delta = 16N_{\mathsf{max}}^2 + 280N_{\mathsf{max}} + 49$.*
3) *$L = q+1$, $N_{\mathsf{max}} = q^2+q+1$, $N_{\mathsf{min}} = N_{\mathsf{max}} - \left\lfloor \frac{3q^3+4q^2+2q-\sqrt{\Delta}}{4q+2} \right\rfloor - 1$, where $\Delta = q^6+12q^5+16q^4+4q^3-8q^2-12q-4$, for every prime power q.*
4) *$L = q$, $N_{\mathsf{max}} = q^2$, $N_{\mathsf{min}} = N_{\mathsf{max}} - \left\lfloor \frac{3q^3-2q^2-2q+1-\sqrt{\Delta}}{4q-2} \right\rfloor - 1$, where $\Delta = q^6+8q^5-24q^4+10q^3+4q^2-4q+1$, for every prime power q.*
5) *$L = q$, $N_{\mathsf{max}} = q^2-1$, $N_{\mathsf{min}} = N_{\mathsf{max}} - \left\lfloor \frac{3q^3-2q^2-5q+3-\sqrt{\Delta}}{4q-2} \right\rfloor - 1$, where $\Delta = q^6+8q^5-26q^4+2q^3+29q^2-14q+1$, for every prime power q.*

Proof. Note that (v,k)-configurations exist for the following v and k.

1) $k \in \{3,4\}$ and $v \geq k(k-1)+1$ (See [30]).
2) $k = q+1$ and $v = q^2+q+1$ for any prime power q. Such a $(q^2+q+1, q+1)$-configuration is also referred to as a finite projective plane. This gives us the Fano plane when $q = 2$. For this existence result and the following ones, see, e.g., [31, p. 2].
3) $k = q$ and $v = q^2$ for any prime power q. A (q^2, q)-configuration can be obtained from a $(q^2+q+1, q+1)$-configuration by removing a point P and all $q+1$ lines containing P *without* removing their points, and also removing one line *containing* P together with all of its points.
4) $k = q$ and $v = q^2-1$ for any prime power q. A (q^2-1, q)-configuration can be obtained from a $(q^2+q+1, q+1)$-configuration by removing a point P and all $q+1$ lines containing P *without* removing their points, and also removing one line *not* containing P together with all of its points.

Applying Theorem 7 to these configurations, setting $N_{\mathsf{max}} = v$ and $L = k$, we deduce the conclusions of the corollary. ■

Applying Corollary 3 to the case $L = 3$ and $N_{\mathsf{max}} = 7$, we obtain a $(3, F)$-ZWR $[N_{\mathsf{min}} = 5, N_{\mathsf{max}} = 7]$ 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 $N_{\mathsf{max}} = 13$, we obtain a $(4, F)$-ZWR $[N_{\mathsf{min}} = 9, N_{\mathsf{max}} = 13]$, which implies that zero-waste transitions are possible between *nine* and *thirteen* machines. When $L = q$ and $N_{\mathsf{max}} = q^2$, for instance, we obtain a (q, F)-ZWR $[N_{\mathsf{min}}, N_{\mathsf{max}}]$ where $N_{\mathsf{min}} = \Theta(N_{\mathsf{max}}/2)$. Ideally, we would like to expand these ranges to $[N_{\mathsf{min}} = L, N_{\mathsf{max}}]$ for every $N_{\mathsf{max}} > L$, which remains an open question.

V. Experiments and Evaluations

As discussed in Section II-D, the case of machines joining seem less practical due to the extra communication overhead associated with data downloading. Therefore, we focus on the case of (one) machine leaving. We first performed simulations of different task allocation schemes in Python to evaluate the impact of the transition wastes on the *CPU usage* and the *computation time* for different sets of parameters (Section V-B). We also implemented and ran these schemes on virtual machines for a specific set of parameters (corresponding to the Fano plane) to see the impact of the transition wastes on the actual *completion time* of different schemes (Section V-C). These experiments demonstrate reasonable reductions in the CPU usage, the computation time, as well as the completion time, when shifted cyclic TAS or zero-weight TAS are used compared to the original cyclic TAS [20].

A. Performance metrics

First, we note that from its definition (see Definition 5), the transition waste incurred at Machine n when Machine n^* leaves, i.e., $W_{n^*}(S_n^N \to S_n^{N-1}) \triangleq \left|S_n^N \Delta S_n^{N-1}\right| - \frac{LF}{N(N-1)}$, is equal to two times the number of tasks *abandoned* by Machine n, defined by $A_{n^*}(S_n^N \to S_n^{N-1}) \triangleq \left|S_n^N \setminus S_n^{N-1}\right|$. We henceforth use *abandoned* and *wasted* interchangeably. *Second*, in reality, the quantity $A_{n^*}(S_n^N \to S_n^{N-1})$ only serves as an upper bound on the actual number of tasks abandoned by Machine n; the reason is that only those tasks already *completed* by Machine n when Machine n^* left can be wasted. Tasks that were originally allocated to Machine n but hadn't been executed by the time Machine n^* left do not contribute to the (actual) transition waste[2]. We ignored the coding/decoding time as this is the same for all schemes.

As such, to reflect the system's performance more accurately, we use $C_{n^*}(S_n^N \to S_n^{N-1})$ to denote the set of *completed* tasks (indices) at Machine n when Machine n^* left and observe that $A_{n^*}^{\text{comp}}(S_n^N \to S_n^{N-1}) \triangleq \left|C_{n^*}(S_n^N \to S_n^{N-1}) \setminus S_n^{N-1}\right|$ is the number of tasks completed but abandoned (wasted) during the transition. We then use the following three different metrics to evaluate different task allocation schemes: the first metric represents the average waste in *CPU usage* while the second and the third represent the impact of transition waste on the actual *computation time* in slightly different ways. Regarding the CPU usage, as long as a task was executed and completed but not used, the CPU time spent on the task is consider wasted. The *completion time* is the time the system requires from the start of computation until the heaviest loaded machine (the bottleneck) finishes all tasks[3]. This is precisely the machine that wasted the largest number of completed tasks in the transition. That is why we need to examine the *maximum* number of wasted completed tasks (over all active machines), which directly translates into the

[2] Investigating the actual transition waste given the list of completed tasks at all machines is a more general problem and left for future research.

[3] To avoid overcomplicating the discussion, we do not consider in our evaluation the straggler-tolerance capability of the underlying coded computing schemes. The analysis can be readily extended to that context by considering, for example, the second or higher-order maximum instead of the maximum.

extra amount of time required for the system to complete the computation compared to the case when none of the completed tasks are wasted (as in a zero-waste TAS). In the following metrics, `avg` is the abbreviation of "average".

- `avg_avg_wasted` $\triangleq$ $\frac{1}{N}\sum_{n^*\in[N]}\left(\frac{1}{N-1}\sum_{n\in[N]\setminus\{n^*\}} A^{\text{comp}}_{n^*}\big(S^N_n \to S^{N-1}_n\big)\right)$, which is the average over all possible indices $n^* \in [N]$ of the average numbers of abandoned completed tasks $A^{\text{comp}}_{n^*}(S^N_n \to S^{N-1}_n)$ over all active machines $n \in [N]$, $n \neq n^*$. The higher `avg_avg_wasted`, the higher waste in CPU usage *on average*.
- `avg_max_wasted` $\triangleq$ $\frac{1}{N}\sum_{n^*\in[N]}\left(\max_{n\in[N]\setminus\{n^*\}} A^{\text{comp}}_{n^*}\big(S^N_n \to S^{N-1}_n\big)\right)$, which is the average over all possible indices $n^* \in [N]$ of the maximum numbers of wasted completed tasks $A^{\text{comp}}_{n^*}(S^N_n \to S^{N-1}_n)$ among all active machines $n \in [N]$, $n \neq n^*$. The higher `avg_max_wasted`, the longer the *averaged* computation time over all n^*.
- `max_max_wasted` $\triangleq$ $\max_{n^*\in[N]}\left(\max_{n\in[N]\setminus\{n^*\}} A^{\text{comp}}_{n^*}\big(S^N_n \to S^{N-1}_n\big)\right)$, which is the maximum among all possible indices $n^* \in [N]$ of the maximum numbers of wasted completed tasks $A^{\text{comp}}_{n^*}(S^N_n \to S^{N-1}_n)$ among all active machines $n \in [N]$, $n \neq n^*$. The higher `max_max_wasted`, the longer the *maximum* computation time among all n^*.

Note that for a zero-waste TAS, $C_{n^*}(S^N_n \to S^{N-1}_n) \subseteq S^N_n \subseteq S^{N-1}_n$, which implies that $A^{\text{comp}}_{n^*}(S^N_n \to S^{N-1}_n) = 0$. Alternatively, this can be deduced from the fact that the number of abandoned *completed* tasks is not greater than the number of abandoned tasks, or half of the transition waste, which is zero in this case. Therefore, all the three metrics defined above are zero for a zero-waste TAS, which is the best possible. It remains to evaluate the performance of the cyclic and shifted cyclic schemes (against the zero-waste schemes).

Here, we examine the transitions when a `fraction` of the original tasks assigned to each machine have been completed, for `fraction` $\in \{0.1, 0.5, 0.9\}$. Note that $\big|C_{n^*}(S^N_n \to S^{N-1}_n)\big|$, i.e., the number of *completed* tasks at Machine n is around `fraction` $\times \frac{LF}{N}$. As both cyclic and shifted cyclic TAS allocate a (cyclically) contiguous chunk of task indices to each machine, naturally, we assume that (cyclically) consecutive tasks are executed starting from the starting point of that set. We measure the percentage of the completed tasks that have been wasted and the percentage of the maximum number of the wasted completed tasks among all machines over the total number of tasks allocated to one machine when each machine has performed a fraction of 1/10, 1/2, and 9/10 originally allocated tasks. Note that for each machine, the amount of *extra* tasks it has to do compared to the case of zero waste ($LF/(N-1)$ tasks) is precisely the number of wasted completed tasks. Instead of using the three aforementioned metrics `avg_avg_wasted`, `avg_max_wasted`, and `max_max_wasted` directly, we transform them into percentages as follows.

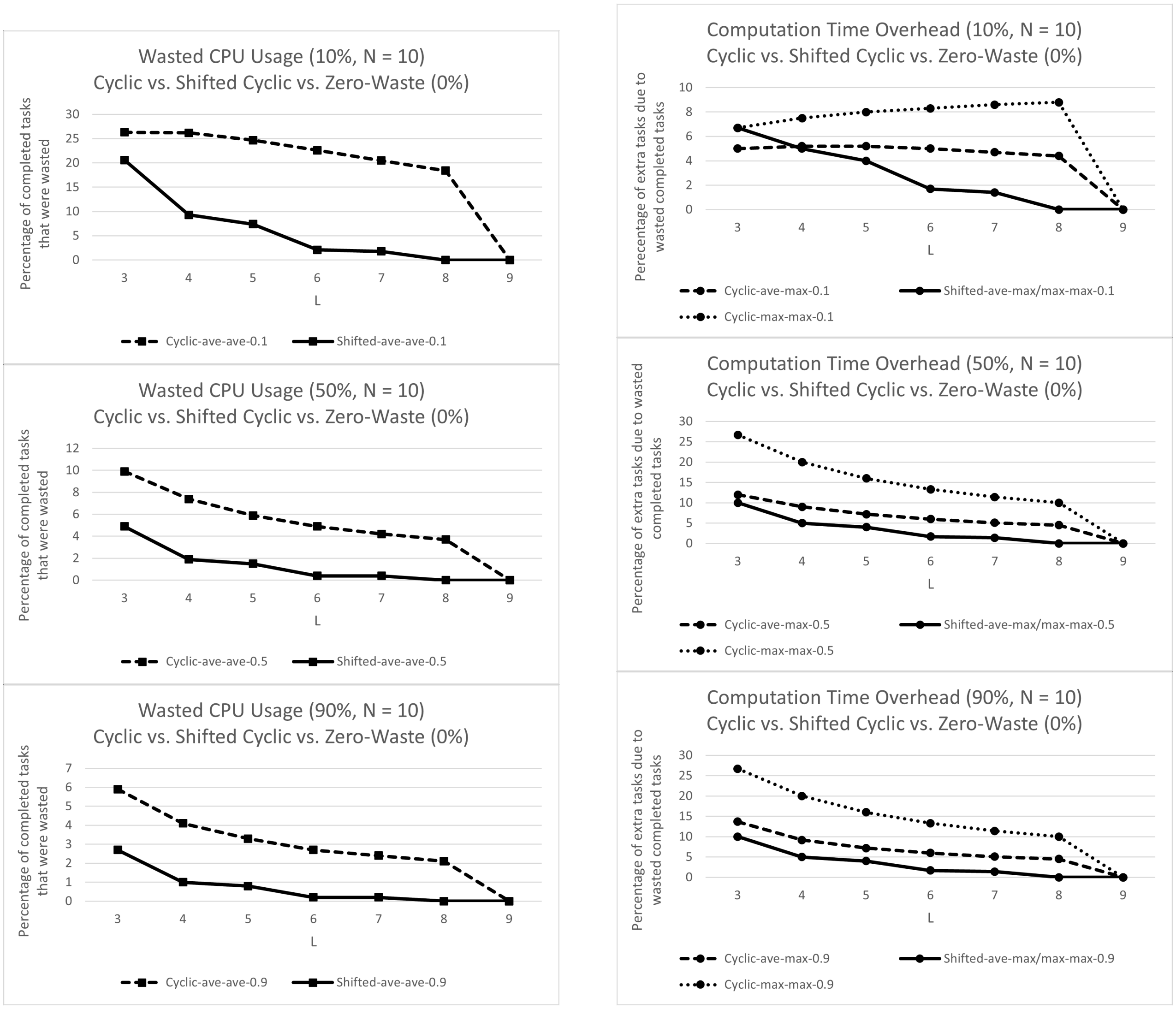
- `aaw_percentage` $\triangleq$ $100\times$ `avg_avg_wasted` $/\big($`fraction` $\times (LF/N)\big)$: the percentage of the completed tasks that have been wasted (averaged over all active machines $n \neq n^*$ and then averaged over all $n^* \in [N]$).
- `amw_percentage` $\triangleq$ $100 \times$ `avg_max_wasted` $/\big(LF/(N-1)\big)$: the percentage of the wasted completed tasks over the total number of allocated task per machine (maximized over active machines $n \neq n^*$ and then averaged over $n^* \in [N]$).
- `mmw_percentage` $\triangleq$ $100 \times$ `max_max_wasted` $/\big(LF/(N-1)\big)$: the percentage of the wasted completed tasks over the total number of allocated task per machine (maximized over all active machines $n \neq n^*$ and over $n^* \in [N]$).

We assume that each task takes the same amount of time to carry out (which makes sense because tasks correspond to computations over data of the same dimensions) and that machines have homogeneous computational capacities/loads (we focus on the performance evaluation of tasks allocation schemes and separate it from the underlying coded computing schemes, which consider stragglers). Then, the CPU usage and computation time of each TAS, as discussed earlier, can be captured accurately by the metrics defined in this section.

B. Simulation

Our simulation results are summarized in Fig. 7, where we fix $N = 10$ and let $L \in [3, 4, \dots, 9]$, and in Fig. 8, in which we fix $L = 5$ and let $N \in [6, 7, \dots, 15]$. We let the transition (one machine leaving) happen when 10%, 50%, or 90% of the originally allocated tasks to each machine had been completed. These are the points where there are some differences in the number of wasted completed tasks, which represent better the differences in the performance metrics among different schemes (e.g., compared to 25%, 50%, and 75%). Note that all metrics used in Section V-A depend on the number of completed tasks that are wasted/abandoned. Our codes are available online at [32].

In summary, the shifted cyclic TAS almost always incurs less waste in CPU and smaller overhead in computation time compared to the cyclic TAS. We also observe that the gap between the performance of the cyclic/shifted cyclic TAS and the zero-waste TAS (no waste in CPU usage or computation time) grows *gradually* when N increases but shrinks more *sharply* when L increases. This is consistent with the derived formulas of the transition waste (in Theorem 2 and Theorem 4), which serves as the upper bound on twice of the number of actual wasted completed tasks. Intuitively, this could also be explained by the fact that the total number of tasks allocated to each machine (the denominator of the metrics), that is $\frac{LF}{N}$, grows linearly with L, while the number of wasted completed tasks (the numerator of the metrics) seems to stay mostly independent of L. Thus, another quick take-away from the simulation is that the zero-waste TAS offers the largest gain over the cyclic TAS for small L (minimum number of machines required by the system) and large N (the number of available machines), and that L plays a more significant role than N in their performance.



(a) Percentage of completed tasks that were wasted due to the transition (averaging over all active machines n and over all machine n^* that left). This represents the waste in CPU usage.

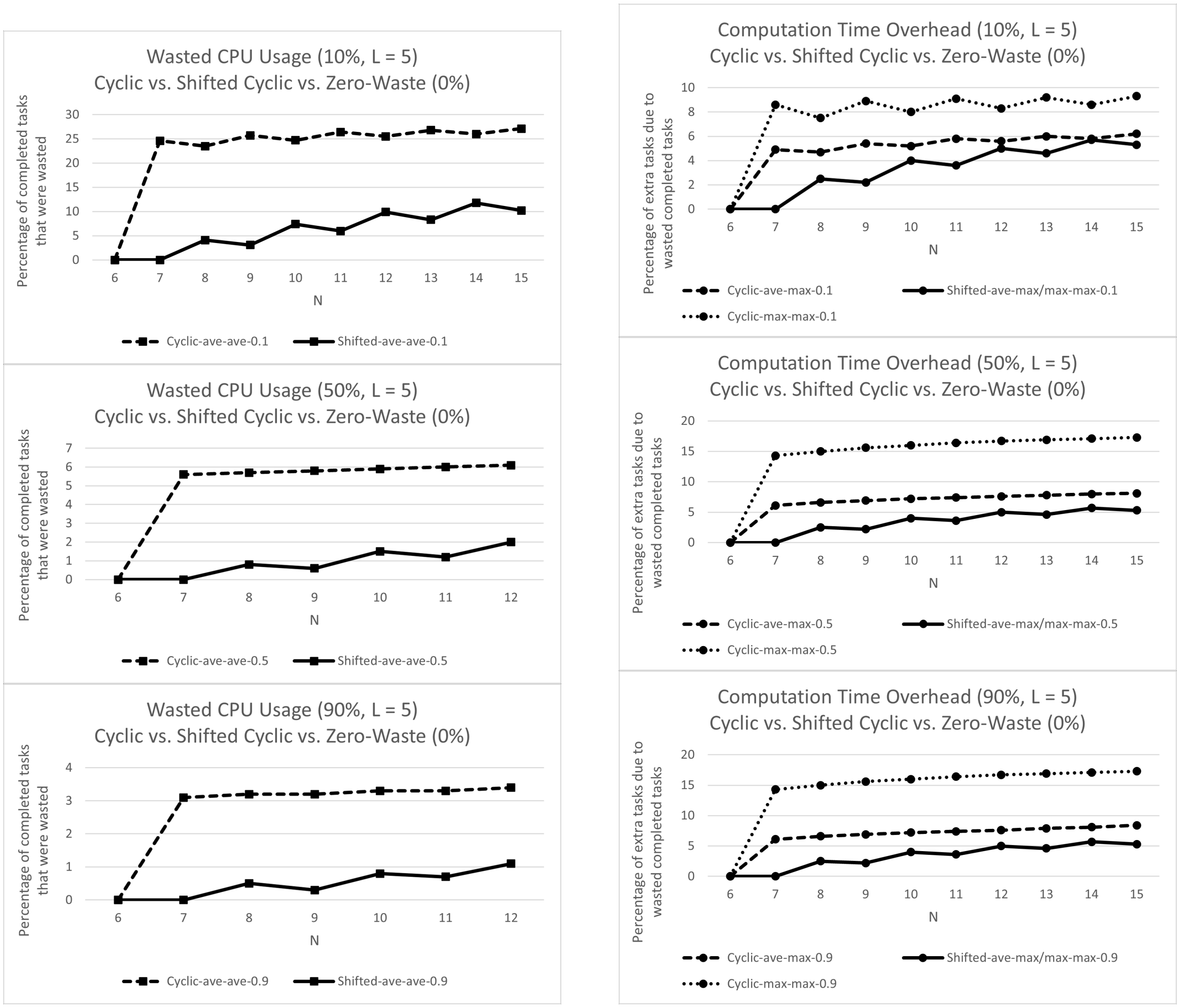
(b) The percentage of extra tasks (compared to a zero-waste scheme) due to wasted completed tasks in the transition. This corresponds to the overhead in computation time.

Fig. 7: The evaluation of the waste in CPU usage (measured by `aaw_percentage`) and the overhead in computation time (measured by `amw_percentage` and `mmw_percentage`) due to completed tasks being wasted during a transition when one machine leaves. We set up the transitions when 10%, 50%, and 90% of the tasks originally allocated to one machine (LF/N tasks) had been completed. We set $N = 10$ and $L \in \{3, 4, \ldots, 9\}$. We observe that the shifted cyclic TAS almost always performs better than the cyclic counterpart (except for a peculiar case when $L = 3$ and $N = 10$ at 10%) and moreover, the gap between the performance of the cyclic and the zero-waste TAS decreases as L grows.

C. Implementation

We implemented the three task allocation schemes (cyclic, shifted cyclic, and zero-waste) on virtual machines and evaluated their performance (completion time) when $N = 7$, $L = 3$, and $F = 210$. The goal is to see if the completion times of different TAS are consistent with our simulation results. The selection of N and L is due to the parameter of the Fano plane (seven lines with three points per line). In theory, we only need $F = 42$ to ensure that F is divisible by $N = 7$ and $N - 1 = 6$, considering one machine leaving in our experiment. However, we set $F = 210$, which is a medium number of tasks to make sure that the computation time is not too short to be ignored and at the same time, to avoid large overhead for the zero-waste scheme. Each task was carried out by multiplying a 2000×5000 matrix and a vector of length 5000 with integer entries randomly generated between -100 and 100. It took approximately 0.8 second to run each task at a worker. Each machine was initially allocated $LF/N = 90$ tasks when there are seven machines, and later with $LF/(N-1) = 105$ tasks when one machine leaves.

We used one virtual machine (the master) to run a bash script that coordinates the experiment on seven other virtual machines (the workers), all of which are Oracle cloud's virtual



(a) Percentage of completed tasks that were wasted due to the transition (averaging over all active machines n and over all machine n^* that left). This represents the waste in CPU usage.

(b) The percentage of extra tasks (compared to a zero-waste scheme) due to wasted completed tasks in the transition. This corresponds to the overhead in computation time.

Fig. 8: The evaluation of the waste in CPU usage and the overhead in computation time due to wasted completed tasks in the cyclic and shifted cyclic schemes. Same as Fig. 7 but we set $L = 5$ and $N \in \{6, 7, \ldots, 15\}$ instead. We observe that the shifted cyclic TAS almost always performs better than the cyclic counterpart and moreover, the gap between the performance of the cyclic and the zero-waste TAS tends to increase as N grows.

machines VM.Standard.E2.1 with one OCPU, 0.7Gbs network bandwidth and 8GB of memory. The Python modules that performed the tasks were loaded into the workers. The master used `parallel-ssh` to send/retrieve data and commands to/from the workers. First, the master set `time_start` to be the start time and issue a command to run the Python modules on all seven workers. To simulate the transition when one machine leaves at different times, we let the main Python module in each worker stop itself once it had completed 10% (9 tasks), 50% (45 tasks), and 90% (81 tasks) of its originally allocated tasks (90 tasks), respectively. Each machine wrote into its log the list of tasks that had been completed. As all workers have the same configurations, they finished almost at the same time.

Once the master gathered that all workers had stopped, it removed one worker (Machine n^*) and issued another command to run the main Python modules on the six remaining ones (Machines $n = 1, 2, \ldots, 7$ with $n \neq n^*$). At each remaining worker, the main Python module allocated a new set of tasks to the machine, depending on its index n and the index of the machine that left n^*, and also on the particular task allocation scheme selected for that experiment (cyclic, shifted cyclic, or zero-waste). The list of tasks completed before the transition was read from its log and ignored because there is no need to run them the second time. Only tasks that hadn't been completed before were run. The master then waited for all six workers to complete their allocated tasks and set `time_end` to be the ending time. The *completion* *time* of the system was set to be completion_time = time_end − time_start. This effectively recorded the *maximum* running time among all remaining machines, which was then averaged out over all $n^* = 1, 2, \ldots, 7$.

Note that the completion time includes the computation time and others such as I/O and communication overhead. Compared to the simulated computation times (Fig. 9(a)), the gaps in the completion times of these three schemes are smaller (Fig. 9(b)). This was partly due to the impact of the communication overhead caused by parallel-ssh and of the I/O time (reading the large matrix into the memory). In total, the overhead, apart from computing the tasks, was approximately 20 seconds. Better management of the communication and I/O may increase the impact of the computation time on the overall completion time.

VI. Conclusions

Building up on the work of Yang *et al.* [20] on coded elastic computing, we first propose a complete *separation* between the elastic task allocation scheme and the coded computing scheme. As a result, we have the freedom to design *efficient* elastic task allocation schemes as a combinatorial object *independent* of the underlying coded computing schemes. Moreover, our result can be applied to almost every coded computing scheme developed in the literature. We illustrate the application of our result in matrix-vector and matrix-matrix multiplication, linear regression, and multivariate polynomial evaluation. The proposed separation *simplifies* the coupling significantly compared to the original approach in [20].

Our main contributions in this work include the introduction of a new performance criterion for elastic task allocation schemes called the *transition waste* and constructions of different schemes that achieve *optimal* transition wastes. This quantity measures the number of tasks that available machines must abandon or take anew when one machine leaves or joins in the middle of the computation of a large scaled job. Smaller transition wastes reduce the waste of computing resources and speed up the job completion time.

Our work and a few others [20]–[24] address the need to bridge the gap between the common setup of most coded computing schemes in the literature, where the number of available machines remain fixed, and an emerging trend in the cloud computing industry where the number of available machines can vary, due to the fact that low-priority virtual machines are often offered at much cheaper prices but can be taken back under a short notice (e.g. Amazon EC2 Spot and Microsoft Azure Batch).

We can imagine one application of the coded elastic computing scheme as follows. We purchase a number of EC2 on-demand instances at a higher price while also get a few Spot instances at a much cheaper cost to run our computation. During the computation cycle, the low-priority Spot instances may leave, reducing the number of available machines. Our system can still handle this if we employ a coded elastic computing scheme in which the number of on-demand instances is greater than or equal to the minimum number of available machines required by the scheme. Thus, instead of maintaining all the costly on-demand instances from the beginning to the end, this approach allows us to take advantage of low-cost Spot instances available to us while keeping the computation run smoothly even when machines leave. An interesting related approach from Amazon in 2018 was implemented in a new feature called Amazon EC2 Fleet [33], which allows users to specify the target capacity and the preferred EC2 instances while automatically performs mix-and-match to meet customers specifications at a lowest price.

Acknowledgement

This work was supported by the Australian Research Council via the Discovery Project under Grant DP200100731. The implementation was done on Oracle Cloud virtual machines sponsored by the Oracle for Research. The work of Yu-Chih Huang was supported by the Ministry of Science and Technology (MOST), Taiwan, under Grant MOST 111-2221-E-A49-069-MY3.

We thank Yaoqing Yang for helpful discussions and Pham Ngoc Duy and Khang Vo for their assistance in the implementation. We also thank the anonymous referees for many constructive comments, which helped to greatly improve the paper.

References

[1] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, “Spark: Cluster computing with working sets,” in *Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing*, 2010.

[2] Apache Hadoop. http://hadoop.apache.org.

[3] G. Ananthanarayanan, A. Ghodsi, S. Shenker, and I. Stoica, “Effective straggler mitigation: Attack of the clones,” in *Proceedings of the 10th USENIX Conference on Networked Systems Design and Implementation*, 2013, pp. 185–198.

[4] F. Dean and L. A. Barroso, “The tail at scale,” *Communications of the ACM*, vol. 56, no. 2, pp. 74–80, 2013.

[5] N. J. Yadwadkar, B. Hariharan, J. E. Gonzalez, and R. Katz, “Multi-task learning for straggler avoiding predictive job scheduling,” *Journal of Machine Learning Research*, vol. 17, no. 1, pp. 3692–3728, 2016.

[6] K. Lee, M. Lam, R. Pedarsani, D. Papailiopoulos, and K. Ramchandran, “Speeding up distributed machine learning using codes,” *IEEE Transactions on Information Theory*, vol. 64, no. 3, pp. 1514–1529, 2018.

[7] S. Li, M. A. Maddah-Ali, and A. S. Avestimehr, “Coded MapReduce,” in *Proceedings of the 53rd Annual Allerton Conference on Communication, Control, and Computing*, 2015, pp. 964–971.

[8] R. Tandon, Q. Lei, A. G. Dimakis, and N. Karampatziakis, “Gradient coding: Avoiding stragglers in distributed learning,” in *Proceedings of the 34th International Conference on Machine Learning*, 2017, pp. 3368–3376.

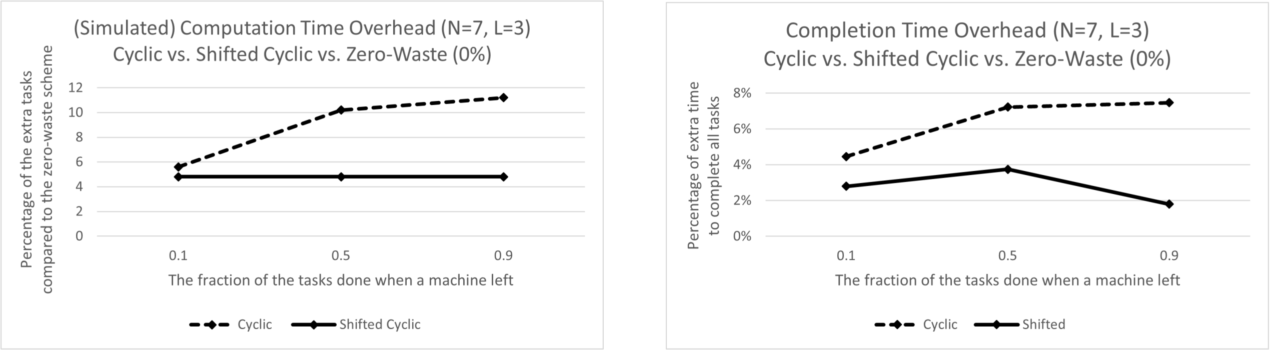
[9] K. H. Huang and J. A. Abraham, “Algorithm-based fault tolerance for matrix operations,” *IEEE Transactions on Computers*, vol. C-33, no. 6, pp. 518–528, 1984.

[10] S. Dutta, V. Cadambe, and P. Grover, “Short-dot: Computing large linear transforms distributedly using coded short dot products,” in *Advances in Neural Information Processing Systems*, 2016, pp. 2100–2108.

[11] Q. Yu, M. A. Maddah-Ali, and S. Avestimehr, “Polynomial codes: an optimal design for high-dimensional coded matrix multiplication,” in *Advances in Neural Information Processing Systems*, 2017, pp. 4403–4413.

[12] C. Karakus, Y. Sun, S. Diggavi, and W. Yin, “Straggler mitigation in distributed optimization through data encoding,” in *Advances in Neural Information Processing Systems 30*, 2017, pp. 5434–5442.

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



(a) (Simulated) Computation overheads of the cyclic and shifted cyclic schemes compared to the zero-waste scheme (set at 0%).

(b) Completion time overheads of the cyclic and shifted cyclic schemes compared to the zero-waste scheme (set at 0%).

Fig. 9: The predicted computation time and the actual completion time overheads of the cyclic and shifted cyclic schemes versus the zero-waste scheme when $N = 7$, $L = 3$, and $F = 210$, running on Oracle's virtual machines.

[14] A. Mallick, M. Chaudhari, U. Sheth, G. Palanikumar, and G. Joshi, "Rateless codes for near-perfect load balancing in distributed matrix-vector multiplication," *Communications of the ACM*, vol. 65, no. 5, pp. 111–118, 2022.

[15] Q. Yu, S. Li, N. Raviv, S. M. M. Kalan, M. Soltanolkotabi, and S. Avestimehr, "Lagrange coded computing: Optimal design for resiliency, security, and privacy," in *Proceedings of Machine Learning Research*, vol. 89, 2019, pp. 1215–1225.

[16] J. Kosaian, 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

[17] ——, "Learning-based coded computation," *IEEE Journal on Selected Areas in Information Theory*, vol. 1, no. 1, pp. 227–236, 2020.

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

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

[20] Y. Yang, P. Grover, and S. Kar, "Coded elastic computing," in *IEEE International Symposium on Information Theory*, 2019, pp. 2654–2658.

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

[22] ——, "Coded elastic computing on machines with heterogeneous storage and computation speed," *IEEE Transactions on Communications*, vol. 69, no. 5, pp. 2894–2908, 2021.

[23] N. Woolsey, J. Kliewer, R.-R. Chen, and M. Ji, "A practical algorithm design and evaluation for heterogeneous elastic computing with stragglers," in *IEEE Global Communications Conference*, 2021, pp. 1–6.

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

[25] R. R. Muntz and J. C. S. Lui, "Performance analysis of disk arrays under failure," in *Proceedings of the 16th International Conference on Very Large Data Bases*, ser. VLDB '90, 1990, pp. 162–173.

[26] M. Holland and A. G. Gibson, "Parity declustering for continuous operation in redundant disk arrays," in *Proceedings of the Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS V, 1992, pp. 23–35.

[27] S. H. Dau, Y. Jia, C. Jin, W. Xi, and K. S. Chan, "Parity declustering for fault-tolerant storage systems via t-designs," in *2014 IEEE International Conference on Big Data*, 2014, pp. 7–14.

[28] P. Hall, "On representatives of subsets," *Journal of the London Mathematical Society*, vol. s1-10, no. 1, pp. 26–30, 1935.

[29] R. K. Ahuja, R. L. Magnanti, and J. B. Orlin, *Network Flows*. Englewood Cliffs, NJ: Prentice-Hall, 1993.

[30] C. J. Colbourn and J. H. Dinitz, *Handbook of Combinatorial Designs, Second Edition (Discrete Mathematics and Its Applications)*. CRC Press, 2006.

[31] M. Funk, D. Labbate, and V. Napolitano, "Tactical (de-)compositions of symmetric configurations," *Discrete Mathematics*, vol. 309, no. 4, pp. 741–747, 2009.

[32] Python codes for task allocations in coded elastic computing. [Online]. Available: https://github.com/dausonhoang/coded_elastic_computing

[33] Introducing Amazon EC2 Fleet. https://aws.amazon.com/about-aws/whats-new/2018/04/introducing-amazon-ec2-fleet/.

[34] M. Fahim, H. Jeong, F. Haddadpour, S. Dutta, V. Cadambe, and P. Grover, "On the optimal recovery threshold of coded matrix multiplication," in *Proceedings of the 55th Annual Allerton Conference on Communication, Control, and Computing*, 2017, pp. 1264–1270.

VII. Appendix

A. Coupling an Elastic Task Allocation Scheme and a Coded Computing Scheme

Matrix-Matrix Multiplication. The goal is to compute the product $\boldsymbol{AB}$, where $\boldsymbol{A}$ and $\boldsymbol{B}$ are matrices of matching dimensions, in the presence of E stragglers $(0 \leq E < L)$ and with a varied number of available machines N $(L \leq N \leq N_{\mathsf{max}})$.

We partition $\boldsymbol{A}$ and $\boldsymbol{B}$ column-wise and row-wise, respectively, into F equal-sized sub-matrices (padding with zeros if necessary) as follows,

$$\boldsymbol{A} = \begin{bmatrix} \boldsymbol{A}_0, & \boldsymbol{A}_1, & \ldots & \boldsymbol{A}_{F-1} \end{bmatrix}, \quad \boldsymbol{B} = \begin{bmatrix} \boldsymbol{B}_0 \\ \boldsymbol{B}_1 \\ \vdots \\ \boldsymbol{B}_{F-1} \end{bmatrix}.$$

The pair $(\boldsymbol{A}_f, \boldsymbol{B}_f)$, $f \in [[F]]$, forms the fth sub-instance and the computation of $\boldsymbol{A}_f\boldsymbol{B}_f$ is referred to as Task f. As $\boldsymbol{AB} = \sum_{f=0}^{F-1} \boldsymbol{A}_f\boldsymbol{B}_f$, the completion of all F tasks gives us the product $\boldsymbol{AB}$. For each Task f, an existing CCS for matrix-matrix multiplication can be applied (e.g., MatDot [34]).

Linear Regression. Given a data matrix $\boldsymbol{X}$ and a vector $\boldsymbol{y}$, we aim to find a weight vector $\boldsymbol{w}$ that minimizes the loss function $\|\boldsymbol{Xw} - y\|^2$. Using gradient descent, in each iteration, we update the weight using the gradient of the loss function, which requires the computation of $\boldsymbol{X}^{\mathrm{T}}(\boldsymbol{Xw}^{(t)} - \boldsymbol{y})$.

The algorithm in [20] first computes $\boldsymbol{Xw}^{(t)}$ via coded elastic computing, computes $\boldsymbol{z}^{(t)} = \boldsymbol{Xw}^{(t)} - \boldsymbol{y}$ at the master node, and adaptively encodes $\boldsymbol{z}^{(t)}$ according to the knowledge of machines that are active. Hence, it is not suitable for the scenario where machines join or leave in the middle of each iteration. Our approach presented below simplifies the approach in [20] and also overcomes its drawback.

Note that both $\boldsymbol{X}$ and $\boldsymbol{y}$ are fixed while $\boldsymbol{w}^{(t)}$ varies from one iteration to the next. Therefore, the matrix-matrix product $\boldsymbol{A} = \boldsymbol{X}^T\boldsymbol{X}$ and the matrix-vector product $\boldsymbol{X}^T\boldsymbol{y}$ can be computed once in advance with amortized cost using an ETAS as described earlier. The only job left is to repeatedly compute $\boldsymbol{A}\boldsymbol{w}^{(t)}$, $t = 0, 1, \dots$ Again, we use an ETAS to perform this matrix-vector multiplication. Despite of its conceptual simplicity, this procedure not only allows machines join or leave in the middle of each iteration but also saves communication bandwidth as at each iteration, we only send $\boldsymbol{w}^{(t)}$ to machines rather than both $\boldsymbol{w}^{(t)}$ and a coded version of $\boldsymbol{z}^{(t)}$.

Multivariate polynomial evaluation. We aim to compute $g(\boldsymbol{X}_1), \dots, g(\boldsymbol{X}_K)$, where g is a multivariate polynomial and $\boldsymbol{X}_k$ is a large matrix or vector ($k \in [K]$), in a way that tolerates E stragglers and allows the number of available machines vary between L and N_{max}.

Suppose that K is divisible by F (padding if necessary). We partition the set of evaluation points into F equal parts

$$\mathcal{P}_f = \left\{\boldsymbol{X}_{fK/F+1}, \dots, \boldsymbol{X}_{fK/F+K/F}\right\}, \quad f \in [[F]].$$

Task f refers to the computations of $g(\boldsymbol{X}_p)$, $p \in \mathcal{P}_f$. Clearly, the completion of all F tasks gives us $g(\boldsymbol{X}_1), \dots, g(\boldsymbol{X}_K)$ as desired. Yu *et al.* [15] propose a CCS called the Lagrange coded computing to perform distributed polynomial evaluation that tolerates stragglers. We can apply this CCS to each task using N_{max} machines and recovery threshold $L - E$.

B. Proof of Lemma 4

Proof of Lemma 4. As $n > n^*$, we have

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

We now apply Lemma 1 to the sets

$$S = S_n^{N-1} = [a, b] \pmod F, \quad T = S_n^N = [c, d] \pmod F.$$

The common assumptions of Lemma 1 are verified as follows. We have

$$0 \le a = (n-2)\frac{F}{N-1} < c = (n-1)\frac{F}{N} < F,$$

$$0 < |S| = \frac{LF}{N-1} < F, \quad 0 < |T| = \frac{LF}{N} < F.$$

Case 1. When $n^* \ge N - L$ or $n^* < N - L$ but $n > N - L$, we aim to show $W_{n^*}(S_n^N \to S_n^{N-1}) = 0$ by proving that $S_n^N \subset S_n^{N-1}$ (Lemma 2). Note that in this case, we always have $n \ge N - L + 1$. Therefore,

$$\frac{LF}{N(N-1)} \ge \frac{(N-n+1)F}{N(N-1)}, \tag{10}$$

which is equivalent to $|S_n^{N-1}| - |S_n^N| \ge (c - a)$, or $|S| \le (c - a) + T$. By Lemma 1 (b), we conclude that $T = S_n^N \subset S = S_n^{N-1}$, as desired. Hence the transition waste incurred at Machine n is zero.

Case 2. Suppose that $n^* < n \le N - L$. The inequality (10) is reversed, which gives us $|S| < (c-a) + |T|$. We now verify that other conditions of Lemma 1 (a) are also satisfied. First, it is clear that

$$c - a = \frac{(N-n+1)F}{N(N-1)} < \frac{LF}{N-1} = |S|.$$

Moreover, as $N > L + 1$ (our assumption),

$$(c-a) + |T| = \frac{(N-n+1)F}{N(N-1)} + \frac{LF}{N} < F.$$

Therefore, by Lemma 1 (a), we obtain

$$\begin{aligned} |S_n^N \Delta S_n^{N-1}| &= 2(c-a) + (|S_n^N| - (|S_n^{N-1}|) \\ &= \frac{2(N-n+1)F}{N(N-1)} - \frac{LF}{N(N-1)}. \end{aligned}$$

Noting that $\Delta_{N,N-1} = \frac{LF}{N(N-1)}$, we obtain

$$\begin{aligned} W_{n^*}(S_n^N \to S_n^{N-1}) &= |S_n^N \Delta S_n^{N-1}| - \Delta_{N,N-1} \\ &= \frac{2(N-L-n+1)F}{N(N-1)}. \end{aligned}$$

This completes the proof. ■

C. Proof of Theorem 3

Proof of Theorem 3. Without loss of generality, we can always assume that $\delta' = 0$ and $\delta = \lfloor \frac{N+L-1}{2} \rfloor \frac{F}{N(N+1)}$. We provide a proof when $N + L$ is odd, i.e., $\delta = \frac{(N+L-1)F}{2N(N+1)}$ noting that we assume $N(N+1)$ divides F (padding with dummy tasks if necessary). A proof for the case when $N + L$ is even can be done similarly.

With $\delta' = 0$ and $\delta = \frac{(N+L-1)F}{2N(N+1)}$, we have

$$\mathcal{S}^N_{\delta'\text{-}\mathsf{cyc}} = (S_1^N, \dots, S_N^N), \quad \mathcal{S}^{N+1}_{\delta\text{-}\mathsf{cyc}} = (S_1^{N+1}, \dots, S_{N+1}^{N+1}),$$

where for $n \in [N]$,

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

$$\begin{aligned} S_n^{N+1} = \Bigg[&(n-1)\frac{F}{N+1} + \frac{(N+L-1)F}{2N(N+1)}, \\ &(n-1)\frac{F}{N+1} + \frac{LF}{N+1} - 1 + \frac{(N+L-1)F}{2N(N+1)}\Bigg] \quad (\mathrm{mod}\ F). \end{aligned}$$

To compute the transition waste $W(S_n^N \to S_n^{N+1})$ incurred at Machine $n \in [N]$, we consider the following three cases.

Case 1. $1 \le n < \frac{N-L+1}{2}$. It can be easily verified that all conditions of Lemma 1 (a) are satisfied for $S \triangleq S_n^N = [a, b]$ $(\mathrm{mod}\ F)$ and $T \triangleq S_n^{N+1} = [c, d]$ $(\mathrm{mod}\ F)$. Therefore,

$$\begin{aligned} W(S_n^N \to S_n^{N+1}) &= 2(c-a) - 2\Delta_{N,N+1} \\ &= \frac{(N+L+1-2n)F}{N(N+1)} - \frac{2LF}{N(N+1)} \\ &= \frac{(N-L+1-2n)F}{N(N+1)}. \end{aligned}$$

Case 2. $\frac{N-L+1}{2} \le n < \frac{N+L+1}{2}$. We can verify that all conditions of Lemma 1 (b) are satisfied for $S \triangleq S_n^N = [a, b]$ $(\mathrm{mod}\ F)$ and $T \triangleq S_n^{N+1} = [c, d]$ $(\mathrm{mod}\ F)$. Hence, $T \subset S$ and $W(S_n^N \to S_n^{N+1}) = 0$.

Case 3. $\frac{N+L+1}{2} \le n \le N$. We can verify that all conditions of Lemma 1 (a) are satisfied for $S \triangleq S_n^{N+1} = [a, b]$ $(\mathrm{mod}\ F)$ and $T \triangleq S_n^N = [c, d]$ $(\mathrm{mod}\ F)$. Therefore,

$$\begin{aligned} W(S_n^N \to S_n^{N+1}) &= 2(c-a) + \Delta_{N,N+1} - \Delta_{N,N+1} \\ &= \frac{(2n - (N+L+1))F}{N(N+1)}. \end{aligned}$$

Thus, the waste when transitioning from $\mathcal{S}^N_{\delta\text{-cyc}}$ to $\mathcal{S}^{N+1}_{\delta'\text{-cyc}}$ is

$$
\begin{aligned}
W(\mathcal{S}^N_{\delta\text{-cyc}} \to \mathcal{S}^{N+1}_{\delta'\text{-cyc}}) &= \frac{F}{N(N+1)}\Bigg(\sum_{n=1}^{\frac{N-L-1}{2}} (N-L+1-2n) \\
&\quad + \sum_{n=\frac{N-L+1}{2}}^{\frac{N+L-1}{2}} 0 + \sum_{n=\frac{N+L+1}{2}}^{N} (2n-(N+L+1))\Bigg) \\
&= \frac{(N-L-1)(N-L+1)F}{2N(N+1)}.
\end{aligned}
$$

This completes the proof. ■

D. Proof of Theorem 5

Note that we only need to prove Theorem 5 for the case when Machine $N+1$ joins. The following lemma holds for all $\delta \in [[F]]$.

Lemma 9. *The transition waste when transitioning from a cyclic (N, L, F)-TAS $\mathcal{S}^N_{\text{cyc}}$ to a δ-shifted cyclic $(N+1, L, F)$-TAS $\mathcal{S}^{N+1}_{\delta\text{-cyc}}$ is*

$$W(\mathcal{S}^N_{\text{cyc}} \to \mathcal{S}^{N+1}_{\delta\text{-cyc}}) = \texttt{Sum1} + \texttt{Sum2} + \texttt{Sum3},$$

where these three sums are given as follows. Setting $d = \frac{F}{N(N+1)} \in \mathbb{Z}$, the first sum is

$$\texttt{Sum1} = \sum_{n\in[N]:(n-1)d>\delta} 2((n-1)d-\delta).$$

When $L < \lceil \frac{N+1}{2} \rceil$, the second and third sums are

$$
\begin{aligned}
\texttt{Sum2} &= \sum_{n\in[N]:(n-1+L)d\leq\delta<(n-1+L+LN)d} 2\big(\delta-(n-1+L)d\big), \\
\texttt{Sum3} &= \sum_{n\in[N]:(n-1+L+LN)d\leq\delta\leq F+(n-1)d-LNd} 2LNd \\
&\quad + \sum_{n\in[N]:F+(n-1)d-LNd<\delta} 2\big(F+(n-1)d-\delta\big).
\end{aligned}
$$

When $L \geq \lceil \frac{N+1}{2} \rceil$, the second and third sums are

$$
\begin{aligned}
\texttt{Sum2} &= \sum_{n\in[N]:(n-1+L)d\leq\delta\leq F+(n-1)d-LNd} 2\big(\delta-(n-1+L)d\big) \\
&\quad + \sum_{F+(n-1)d-LNd<\delta<(n-1+L+LN)d} 2(N-L)F/N, \\
\texttt{Sum3} &= \sum_{n\in[N]:(n-1+L+LN)d\leq\delta} 2\big(F+(n-1)d-\delta\big).
\end{aligned}
$$

Proof. These sums are obtained by considering all possible cases of the intersection between S_n^N and S_n^{N+1} taking into account the fact that we have shifted S_n^{N+1} cyclicly by δ positions compared to the ordinary cyclic TAS.

Let $\mathcal{S}^N_{\text{cyc}} = (\mathcal{S}^N_1, \ldots, \mathcal{S}^N_N)$ and $\mathcal{S}^{N+1}_{\delta\text{-cyc}} = (\mathcal{S}^{N+1}_1, \ldots, \mathcal{S}^{N+1}_{N+1})$. For $n \in [N]$,

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

$$
\begin{aligned}
S_n^{N+1} = \Bigg[&(n-1)\frac{F}{N+1} + \delta, \\
&(n-1)\frac{F}{N+1} + \frac{LF}{N+1} - 1 + \delta\Bigg] \pmod F.
\end{aligned}
$$

To compute the transition waste $W(S_n^N \to S_n^{N+1})$ incurred at Machine $n \in [N]$, we consider the following three cases depending on the relative position of the endpoints of S_n^N and S_n^{N+1} on the circle of integers mod F.

Case 1. $\delta < \frac{(n-1)F}{N(N+1)} = (n-1)d$. The left endpoint of S_n^{N+1} lies between 0 and the left endpoint of S_n^N (see Fig. 10). Applying Lemma 1 (a) to $S = S_n^{N+1}$ and $T = S_n^N$, we have

$$W(S_n^N \to S_n^{N+1}) = 2((n-1)d - \delta).$$

Case 1 gives rise to `Sum1`.

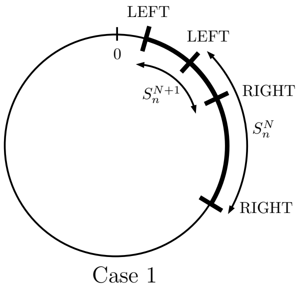
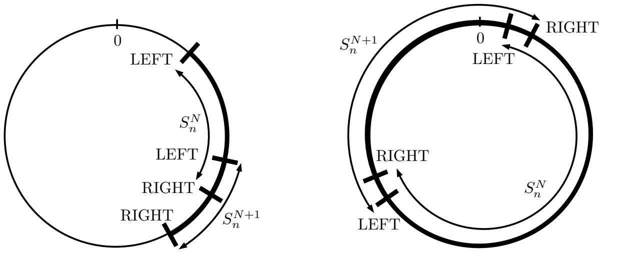


Fig. 10: Illustration of Case 1.



(a) Continguous Intersection (b) Non-contiguous intersection

Fig. 11: Illustration of scenarios in Sub-case 2.2.

Case 2. $(n-1)d \leq \delta < (n-1+L+LN)d$. The left endpoint of S_n^{N+1} lies between the two endpoints of S_n^N (inclusive). We further divide Case 2 into two sub-cases.

Sub-case 2.1. $(n-1)d \leq \delta < (n-1+L)d$. Since $S_n^{N+1} \subset S_n^N$, by Lemma 2, the transition waste is zero and we can ignore this sub-case.

Sub-case 2.2. $(n-1+L)d \leq \delta < (n-1+L+LN)d$. When $L < \lceil \frac{N+1}{2} \rceil$, the intersection of S_n^N and S_n^{N+1} is contiguous (see Fig. 11 (a) and we can use similar argument as in Lemma 1 (a) to deduce that

$$W(S_n^N \to S_n^{N+1}) = 2(\delta - (n-1+L)d).$$

When $L \geq \lceil \frac{N+1}{2} \rceil$, we have

$$F + (n-1)d - LNd < (n-1+L+LN)d.$$

This inequality is important because for $(n-1+L)d \leq \delta \leq F + (n-1)d - LNd$, the intersection of S_n^N and S_n^{N+1} is contiguous and the transition waste is

$$W(S_n^N \to S_n^{N+1}) = 2(\delta - (n-1+L)d),$$

while for $F + (n-1)d - LNd < \delta < (n-1+L+LN)d$, the intersection between the two sets is non-contiguous (see Fig. 11 (b)) and the transition waste is

$$W(S_n^N \to S_n^{N+1}) = 2(N-L)F/N.$$

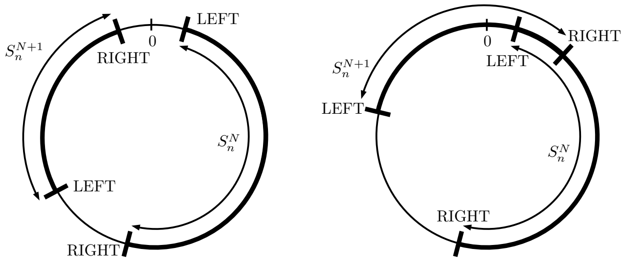
Indeed, as the right endpoint of S_n^{N+1} is $(n-1)\frac{F}{N+1} + \frac{LF}{N+1} - 2 + \delta - F$ in this case, the intersection of the two sets has size

$$\left(\left(\frac{(n-1)F}{N} + \frac{LF}{N} - 1\right) - \left(\frac{(n-1)F}{N+1} + \delta\right) + 1\right)$$
$$+ \left(\left(\frac{(n-1)F}{N+1} + \frac{LF}{N+1} - 1 + \delta - F\right) - \frac{(n-1)F}{N} + 1\right)$$
$$= \frac{LF}{N} + \frac{LF}{N+1} - F.$$

Therefore, the transition waste is

$$W(S_n^N \to S_n^{N+1}) = (|S_n^N| + |S_n^{N+1}|) - 2|S_n^N \cap S_n^{N+1}| - \Delta_{N,N+1}$$
$$= \left(\frac{LF}{N} + \frac{LF}{N+1}\right) - 2\left(\frac{LF}{N} + \frac{LF}{N+1} - F\right) - \frac{LF}{N(N+1)}$$
$$= 2(N-L)F/N.$$

These explain the formula of Sum 2.



(a) Non-intersecting case (b) Intersecting case

Fig. 12: Illustration of scenarios in Case 3.

Case 3. $(n-1+L+LN)d \leq \delta < F$. The right endpoint of S_n^N lies between its left endpoint and the left endpoint of S_n^{N+1}. We divide this case further into two sub-cases, depending on whether the two sets intersect or not (see Fig. 12). Note that when the two sets do intersect, the right endpoint of S_n^{N+1} is $(n-1)\frac{F}{N+1} + \frac{LF}{N+1} - 2 + \delta - F$ (i.e., having $-F$).

When $L < \lceil\frac{N+1}{2}\rceil$, for $(n-1+L+LN)d \leq \delta \leq F + (n-1)d - LNd$, the two sets do not intersect (see Fig. 12 (a)), and so, the transition waste is

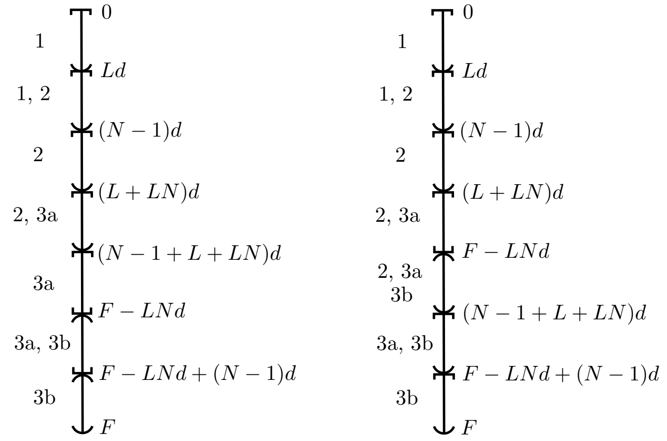
$$W(S_n^N \to S_n^{N+1}) = \frac{LF}{N} + \frac{LF}{N+1} - \frac{LF}{N(N+1)} = 2LNd,$$

while for $F + (n-1)d - LNd < \delta < F$, the two sets intersect (see Fig. 12 (a)) and the transition waste is

$$W(S_n^N \to S_n^{N+1}) = (|S_n^N| + |S_n^{N+1}|) - 2|S_n^N \cap S_n^{N+1}| - \Delta_{N,N+1}$$
$$= \left(\frac{LF}{N} + \frac{LF}{N+1}\right) - 2\left(\left(\frac{(n-1)F}{N+1} + \frac{LF}{N+1} - 1 + \delta - F\right)\right.$$
$$\left. - \frac{(n-1)F}{N} + 1\right) - \frac{LF}{N(N+1)} = 2(F + (n-1)d - \delta).$$

When $L \geq \lceil\frac{N+1}{2}\rceil$, the two sets S_n^N and S_n^{N+1} always intersect and the transition waste is $2(F + (n-1)d - \delta)$. These explain the formula of Sum3. ■

Proof of Theorem 5. Lemma 9 establishes an *implicit* formula for the transition waste when transitioning from a cyclic (N, L, F)-TAS $\mathcal{S}_{\textsf{cyc}}^N$ to a δ-shifted cyclic $(N+1, L, F)$-TAS $\mathcal{S}_{\delta\textsf{-cyc}}^{N+1}$. It remains to determine an *explicit* form of the transition waste and show that it is minimized at $\delta_{\textsf{opt}} = \left\lfloor\frac{N+L-1}{2}\right\rfloor d$. To simplify the computation, we assume that δ is divisible by $d \triangleq \frac{F}{N(N+1)}$. Even with this simplification, the computation is still very tedious with many cases depending on the relation between N and L and the exact interval δ lies in (four cases, each has seven intervals to consider - Figs. 13, 14).



Case 1. $L < \frac{N}{2}$ **Case 2.** N even and $L = \frac{N}{2}$

Fig. 13: Illustration of the intervals for δ and the non-empty sums contributing to the transition waste when $L < \lceil\frac{N+1}{2}\rceil$. The labels 3a/3b refer to the two component sums of `Sum3` (see Lemma 9). The appearance of the labels 1, 2, 3a, 3b in each interval indicate that these sums are non-empty in that interval of δ.

Note that while the transition waste can be written as the sum of four component sums, depending on the interval that δ belongs to, only a few sums are *non-empty* (the lower limit doesn't exceed the upper limit). We must know which sums are non-empty in which intervals of δ to obtain a precise formula for the transition waste. We provide below the explicit formulas of the transition wastes in all four cases and seven intervals, resulting in 28 sub-cases in total. For each sub-case, given the expression of the transition waste, we identify the δ^* (divisible by d) in that interval that minimizes the transition waste and show that this minimum transition waste is greater than or equal to the transition waste provided in Theorem 3.

Case 1: $L < \frac{N}{2}$ (see Fig. 13).

- **Case 1-a:** $0 \leq \delta < Ld$. By Lemma 9, and noting that only `Sum1` is non-empty, we have

$$W(\mathcal{S}_{\textsf{cyc}}^N \to \mathcal{S}_{\delta\textsf{-cyc}}^{N+1}) = \texttt{Sum1} = \sum_{n=\frac{\delta}{d}+2}^{N} 2((n-1)d - \delta)$$
$$= \frac{\delta^2}{d} - (2N-1)\delta + N(N-1)d,$$

  which achieves its minimum value $(N-L+1)(N-L)d$ (among all δ divisible by d) at $\delta^* = (L-1)d$. This value is larger than the transition waste obtained in Theorem 3, noting that $d = F/(N(N+1))$.

- **Case 1-b:** $Ld \leq \delta < (N-1)d$. By Lemma 9,

$$W(\mathcal{S}^{N}_{\mathsf{cyc}} \rightarrow \mathcal{S}^{N+1}_{\delta\text{-}\mathsf{cyc}}) = \mathtt{Sum1} + \mathtt{Sum2},$$

where

$$\mathtt{Sum1} = \sum_{n=\delta/d+2}^{N} 2((n-1)d - \delta)$$
$$= \frac{\delta^2}{d} - (2N-1)\delta + N(N-1)d,$$
$$\mathtt{Sum2} = \sum_{n=1}^{\delta/d-L+1} 2(\delta - (n-1+L)d)$$
$$= \frac{\delta^2}{d} - (2L-1)\delta + L(L-1)d.$$

Note that it is important to determine the precise lower and upper limits for each sum. Hence,

$$W(S^N_n \rightarrow S^{N+1}_n)$$
$$= \frac{2\delta^2}{d} - 2(N+L-1)\delta + (N(N-1) + L(L-1))d.$$

This is a quadratic function of δ, which achieves the minimum at $\delta_{\mathsf{opt}} = \left\lfloor \frac{N+L-1}{2} \right\rfloor d$. This is indeed the shift recommended in Theorem 3.

- **Case 1-c:** $(N-1)d \leq \delta < (L+LN)d$. We have

$$W(\mathcal{S}^{N}_{\mathsf{cyc}} \rightarrow \mathcal{S}^{N+1}_{\delta\text{-}\mathsf{cyc}}) = \mathtt{Sum2}$$
$$= \begin{cases} \sum_{n=1}^{\frac{\delta}{d-L+1}} 2(\delta - (n-1+L)d), & \text{if } \delta \leq (N+L-1)d, \\ \sum_{n=1}^{N} 2(\delta - (n-1+L)d), & \text{if } \delta \geq (N+L)d, \end{cases}$$
$$= \begin{cases} \frac{\delta^2}{d} - (2L-1)\delta + L(L-1)d, & \text{if } \delta \leq (N+L-1)d, \\ \underset{\delta=(N+L)d}{\geq} N(N+1)d = F, & \text{if } \delta \geq (N+L)d, \end{cases}$$

which achieves its minimum value $\frac{(N-L)(N-L-1)F}{N(N+1)}$ (among all δ divisible by d) at $\delta^* = (N-1)d$. This value is larger than the transition waste obtained in Theorem 3.

- **Case 1-d:** $(L+LN)d \leq \delta < (N-1+L+LN)d$. We have

$$W(\mathcal{S}^{N}_{\mathsf{cyc}} \rightarrow \mathcal{S}^{N+1}_{\delta\text{-}\mathsf{cyc}}) = \mathtt{Sum2} + \mathtt{Sum3a}$$
$$= \sum_{n=\delta/d-LN-L+2}^{N} 2(\delta - (n-1+L)d) + \sum_{n=1}^{\delta/d-LN-L+1} 2LNd$$
$$= -\frac{\delta^2}{d} + (N-L-1+2LN)\delta$$
$$- (N^2L - N^2 - NL - 3N - 2L + 2)Ld,$$

which is minimized at either $\delta^*_1 = (LN+L)d$ or $\delta^*_2 = (N-2+L+LN)d$, i.e., $\min\{Ld(2N^2+3N-3), d(2N^2L+N+3L-2)\}$, which is greater than $2N^2Ld$, which in turn is larger than the transition waste in Theorem 3.

- **Case 1-e:** $(N-1+L+LN)d \leq \delta \leq F - LNd$. We have

$$W(S^N_n \rightarrow S^{N+1}_n) = \mathtt{Sum3a} = \sum_{n=1}^{N} 2LNd = 2LN^2d,$$

which is larger than the transition waste in Theorem 3.

- **Case 1-f:** $F-LNd < \delta \leq F-LNd+(N-1)d$. We have

$$W(\mathcal{S}^{N}_{\mathsf{cyc}} \rightarrow \mathcal{S}^{N+1}_{\delta\text{-}\mathsf{cyc}}) = \mathtt{Sum3a} + \mathtt{Sum3b}$$
$$= \sum_{n=\delta/d-N^2-N+LN+1}^{N} 2LNd + \sum_{n=1}^{\delta/d+LN-N^2-N} 2(F+(n-1)d-\delta)$$
$$= -\frac{\delta^2}{d} + (2N^2 - 2LN + 2N - 1)\delta$$
$$- Nd(N^3 - 2N^2L + 2N^2 + NL^2 - 4NL + L - 1),$$

which is minimized at either $\delta^*_1 = F - LNd + d = (N^2 + N - LN + 1)d$ or $\delta^*_2 = F - LNd + (N-1)d = (N^2+2N-LN-1)d$. Therefore, the minimum transition waste in this range of δ (assuming δ is divisible by d) is

$$\min\{2N^2Ld - 2d, 2N^2Ld - N^2d + Nd\}$$
$$= 2N^2Ld - N^2d + Nd,$$

which is greater than the transition waste in Theorem 3.

- **Case 1-g:** $F - LNd + (N-1)d < \delta < F$. We have

$$W(\mathcal{S}^{N}_{\mathsf{cyc}} \rightarrow \mathcal{S}^{N+1}_{\delta\text{-}\mathsf{cyc}}) = \mathtt{Sum3b} = \sum_{n=1}^{N} 2(F+(n-1)d-\delta)$$
$$= -2N\delta + (2N^3d + 3N^2d - Nd)$$
$$\geq -2N(F-d) + (2N^3d + 3N^2d - Nd) = N(N+1)d,$$

which is greater than the transition waste in Theorem 3.

Case 2: $L = \frac{N}{2}$ and N is even (see Fig. 13).

- **Case 2-a:** $0 \leq \delta < Ld$. The formula of the transition waste is the same as Case 1-a.
- **Case 2-b:** $Ld \leq \delta < (N-1)d$. The formula of the transition waste is the same as Case 1-b.
- **Case 2-c:** $(N-1)d \leq \delta < (L+LN)d$. The formula of the transition waste is the same as Case 1-c.
- **Case 2-d:** $(L+LN)d \leq \delta \leq F - LNd$. The formula of the transition waste turns out to be the same as Case 1-d.
- **Case 2-e:** $F-LNd < \delta < (N-1+L+LN)d$. We have

$$W(S^N_n \rightarrow S^{N+1}_n) = \mathtt{Sum2} + \mathtt{Sum3a} + \mathtt{Sum3b}$$
$$\geq \mathtt{Sum3a} = \sum_{n=\delta/d-N^2-N+LN+1}^{\delta/d-LN-L+1} 2LNd$$
$$= (N^2 + N - 2LN - L)2LNd \underset{L=N/2}{=} LN^2d,$$

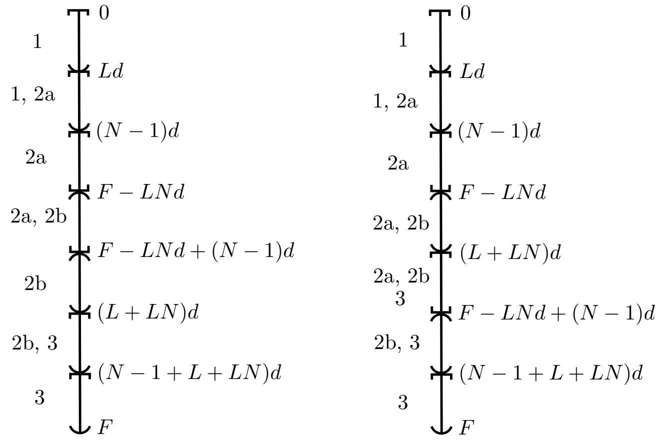
which is larger than the transition waste in Theorem 3.

- **Case 2-f:** $(N-1+L+LN)d \leq \delta < F-LNd+(N-1)d$. We have

$$W(\mathcal{S}^{N}_{\mathsf{cyc}} \rightarrow \mathcal{S}^{N+1}_{\delta\text{-}\mathsf{cyc}}) = \mathtt{Sum3a} + \mathtt{Sum3b} \geq \mathtt{Sum3a}$$
$$= \sum_{n=\delta/d-N^2-N+LN+1}^{N} 2LNd = 2(N^2+2N-LN-1-\delta/d)LNd$$
$$\underset{\delta=F-LNd+(N-2)d}{\geq} 2\big(N^2+2N-LN-1-(N^2+N-LN+N-2)\big)LNd$$
$$= 2LNd \underset{L=N/2}{=} N^2d,$$

which is greater than the transition waste in Theorem 3.

- **Case 2-g:** $F - LNd + (N-1)d < \delta < F$. The formula of the transition waste is the same as Case 1-g.



Case 3. $L > \frac{N+1}{2}$ **Case 4.** N odd and $L = \frac{N+1}{2}$

Fig. 14: Illustration of the intervals for δ and the non-empty sums contributing to the transition waste when $L \geq \lceil\frac{N+1}{2}\rceil$. The labels 2a/2b refer to the component sums of `Sum2` (see Lemma 9). The appearance of the labels 1, 2a, 2b, 3 in each interval indicate that these sums are non-empty in that interval.

Case 3: $L > \frac{N+1}{2}$ (see Fig. 14).

- **Case 3-a:** $0 \leq \delta < Ld$. The same as Case 1-a.
- **Case 3-b:** $Ld \leq \delta < (N-1)d$. The same as Case 1-b. The minimum transition waste is achieved at $\delta^* = \lfloor\frac{N+L-1}{2}\rfloor d$, which is indeed the shift provided in Theorem 3.
- **Case 3-c:** $(N-1)d \leq \delta \leq F-LNd$. The same as Case 1-c.
- **Case 3-d:** $F-LNd < \delta \leq F-LNd+(N-1)d$. We have
$$\begin{aligned} W(\mathcal{S}^N_{\mathsf{cyc}} \to \mathcal{S}^{N+1}_{\delta\text{-}\mathsf{cyc}}) &= \mathtt{Sum2a} + \mathtt{Sum2b} \geq \mathtt{Sum2a} \\ &= \sum_{n=1}^{N} 2(\delta - (n-1+L)d) \\ &\underset{\delta=F-LNd+d}{\geq} 2N^2 d(N-L) + (N^2 d + Nd) \\ &\geq N(N+1)d = F, \end{aligned}$$
which is greater than the transition waste in Theorem 3.
- **Case 3-e:** $F-LNd+(N-1)d < \delta < (L+LN)d$. From Lemma 9, we deduce that
$$W(S^N_n \to S^{N+1}_n) = \mathtt{Sum2b} = \sum_{n=\delta/d-LN-L+2}^{\delta/d+LN-N^2-N} 2\frac{(N-L)F}{N},$$
which is larger than the transition waste in Theorem 3 as long as there is at least one term in the sum. This can be easily shown by verifying that the upper limit is strictly larger than the lower limit of the sum.
- **Case 3-f:** $(L+LN)d \leq \delta < (L+LN+N-1)d$. We have
$$W(\mathcal{S}^N_{\mathsf{cyc}} \to \mathcal{S}^{N+1}_{\delta\text{-}\mathsf{cyc}}) = \mathtt{Sum2b} + \mathtt{Sum3} \geq \mathtt{Sum2b},$$
which is greater than the transition waste in Theorem 3, using the same argument as Case 3-e.
- **Case 3-g:** $(L+LN+(N-1))d \leq \delta < F$. We have
$$\begin{aligned} W(\mathcal{S}^N_{\mathsf{cyc}} \to \mathcal{S}^{N+1}_{\delta\text{-}\mathsf{cyc}}) &= \mathtt{Sum3} = \sum_{n=1}^{N} 2(F + (n-1)d - \delta) \\ &\underset{\delta=F-d}{\geq} N(N+1)d = F, \end{aligned}$$
which is greater than the transition waste in Theorem 3.

Case 4: $L = \frac{N+1}{2}$ and N is odd (see Fig. 14).

- **Case 4-a:** $0 \leq \delta < Ld$. The formula of the transition waste is the same as Case 3-a.
- **Case 4-b:** $Ld \leq \delta < (N-1)d$. The formula of the transition waste is the same as Case 3-b.
- **Case 4-c:** $(N-1)d \leq \delta \leq F-LNd$. The formula of the transition waste is the same as Case 3-c.
- **Case 4-d:** $F-LNd < \delta < (L+LN)d$. This case can be settled using exactly the same argument as in Case 3-d.
- **Case 4-e:** $(L+LN)d \leq \delta \leq F-LNd+(N-1)d$. This case can be settled using exactly the same argument as in Case 3-e.
- **Case 4-f:** $F-LNd+(N-1)d < \delta < (N-1+L+LN)d$. This case can be settled using exactly the same argument as in Case 3-f.
- **Case 4-g:** The formula of the transition waste is the same as Case 3-g. ■

E. Frequently Used Notations

Notation	Meaning
N	The total number of machines in the system.
n	The label of an individual machine. We have $n \in [N] \triangleq \{1,2,\ldots,N\}$.
L	The number of machines required by the underlying coded computing scheme. In the task allocation scheme, each task (index) is allocated to exactly L different machines.
F	The total number of tasks.
f	The label of an individual task. We have $f \in [[F]] \triangleq \{0,1,\ldots,F-1\}$.
S^N_n	A subset of $[[F]]$ representing the set of task indices allocated to Machine n when the system has N machines.
(N,L,F)-TAS	An ordered list of N sets $\mathcal{S}^N = (S^N_1,\ldots,S^N_N)$ satisfying the L-Redundancy and the Load Balancing properties (see Definition 1).
$\mathcal{S}^N_{\mathsf{cyc}} = (S^N_1,\ldots,S^N_N)$	The order list of sets of task indices allocated to Machine $n \in [N]$ by a cyclic TAS (see (2)).
$\mathcal{S}^N_{\delta\text{-}\mathsf{cyc}} = (S^N_1,\ldots,S^N_N)$	The order list of sets of task indices allocated to Machine $n \in [N]$ by a δ-shifted cyclic task allocation scheme (see Definition 6).
$\Delta_{N,N'}$	$\Delta_{N,N'} \triangleq \lvert LF/N - LF/N'\rvert$: the necessary load change when the system transitions from N machines to $N'=N$±1 ones. This is called *necessary* as when a machine leaves/joins, even without any transition waste, the remaining ones must take more/less tasks to maintain the L-redundancy: every task must be covered by L machines.
$W(S^N_n \to \mathcal{S}^{N'}_n)$	$W(S^N_n \to \mathcal{S}^{N'}_n) \triangleq \lvert S^N_n \Delta \mathcal{S}^{N'}_n\rvert - \Delta_{N,N'}$: the transition waste incurred at Machine n when transitioning from a set of tasks S^N_n to another set of tasks $\mathcal{S}^{N'}_n$ (see Definition 4).
$W_{n^*}(S^N_n \to S^{N-1}_n)$	Same as $W(S^N_n \to \mathcal{S}^{N'}_n)$ but more specific to the case $N' = N-1$ and Machine n^* leaves.
$W(\mathcal{S}^N \to \mathcal{S}^{N+1})$	$W(\mathcal{S}^N \to \mathcal{S}^{N+1}) \triangleq \sum_{n\in[N]} W(S^N_n \to S^{N+1}_n)$ is the total transition waste at all machines when Machine $N+1$ joins.
$W_{n^*}(\mathcal{S}^N \to \mathcal{S}^{N-1})$	$W_{n^*}(\mathcal{S}^N \to \mathcal{S}^{N-1}) \triangleq \sum_{n\in[N]\setminus\{n^*\}} W_{n^*}(S^N_n \to S^{N-1}_n)$: the total transition waste at all machines when Machine n^* leaves.
$W_{\mathsf{avg}}(\mathcal{S}^N \to \mathcal{S}^{N-1})$	The average of $W_{n^*}(\mathcal{S}^N \to \mathcal{S}^{N-1})$ over $n^* \in [N]$.

TABLE I: Frequently Used Notations.

Hoang Dau (Member, IEEE) received the B.S. degree in applied mathematics and informatics from Vietnam National University, Hanoi, Vietnam, in 2006, and the M.S. and Ph.D. degrees in mathematical sciences from Nanyang Technological University, Singapore, in 2009 and 2012, respectively. He is currently a senior lecturer in Computer Science at School of Computing Technologies, STEM College, RMIT University. His research interests include coding theory, discrete mathematics, and blockchain.

Ryan Gabrys (Member, IEEE) received the B.S. degree in mathematics and computer science from the University of Illinois at Urbana-Champaing in 2005, and the Ph.D. degree in electrical engineering from the University of California, Los Angeles in 2014. He is currently a Scientist jointly affiliated with the Naval Information Warfare Center and the California Institute for Telecommunications and Information Technology (Calit2) at the University of California, San Diego. His research interests broadly lie in the areas of theoretical computer science and electrical engineering, including coding theory, combinatorics, and communication theory.

Chen Feng (Member, IEEE) received the B.Eng. degree from the Department of Electronic and Communications Engineering, Shanghai Jiao Tong University, China, in 2006, and the M.A.Sc. and Ph.D. degrees from the Department of Electrical and Computer Engineering, University of Toronto, Canada, in 2009 and 2014, respectively. From 2014 to 2015, he was a Post-Doctoral Fellow with Boston University, USA, and École Polytechnique Fédérale de Lausanne (EPFL), Switzerland. He joined the School of Engineering, The University of British Columbia (UBC), Kelowna, Canada, in July 2015, where he is currently the Tier-2 Principal's Research Chair in Blockchain and the Co-Cluster Lead of Blockchain at UBC. His research interests are in coding theory and its applications in various fields, including quantum communications and blockchain technology.

Yu-Chih Huang (Member, IEEE) received the Ph.D. degree in electrical and computer engineering from Texas A&M University (TAMU) in 2013. From 2013 to 2015, he was a Postdoctoral Research Associate with TAMU. In 2015, he joined the Department of Communication Engineering, National Taipei University, Taiwan, as an Assistant Professor and was promoted to an Associate Professor in 2018. In 2020, he joined the Institute of Communications Engineering, National Chiao Tung University (NCTU), Taiwan. He is currently an Associate Professor at National Yang Ming Chiao Tung University (the merger of National Yang Ming University and NCTU in 2021). His research interests are in information theory, coding theory, wireless communications, and statistical signal processing. He received the 2018 IEEE Information Theory Society Taipei Chapter and IEEE Communications Society Taipei/Tainan Chapter's Best Paper Award for Young Scholars and was a recipient of the MOST Young Scholar Fellowship 2020. He is currently serving as an Associate Editor for IEEE Communications Letters.

Quang-Hung Luu (Member, IEEE) received the B.Sc. degree in applied mathematics and mechanics from Vietnam National University, Hanoi, Vietnam, in 2004, and the Ph.D. degrees in earth and planetary sciences, and computer science and software engineering from Kyoto University and Swinburne University of Technology in 2012 and 2021, respectively. He is currently a research fellow sharing the time between Monash University and Swinburne University of Technology. His research interests include software testing, ocean modelling, connected and autonomous vehicles and data analysis.

Eidah J. Alzahrani is an assistant professor at Albaha University (Saudi Arabia). He obtained a bachelor's degree from Albaha University in 2007, a Master of Information Technology from La Trobe University (Australia) in 2010, and a PhD from RMIT University (Australia) in 2020, with a PhD thesis titled ”Proactive auto-scaling techniques for containerised applications”. His current research is on resource management for cloud computing data center, as well as Internet of Things (IoT) solutions in manufacturing.

Zahir Tari is a full professor at RMIT University (Australia) and the Research Director of the RMIT Cyber Security Research and Innovation (CCSRI). His expertise is in the areas of system performance (e.g. P2P, Cloud, Edge/IoT) and security (e.g. SCADA, SmartGrid, Cloud, Edge/IoT).