

# Group Smart, Finish Fast: Minimizing Batch Makespan in Coded Distributed Computing

Yuxuan Lin<sup>1,2</sup>, Bin Tang<sup>1</sup>, and Rajkumar Buyya<sup>2</sup>

<sup>1</sup> Key Laboratory of Water Big Data Technology of Ministry of Water Resources,  
Hohai University, China  
School of Computer Science and Software Engineering, Hohai University, China  
`{linyx, cstb}@hhu.edu.cn`

<sup>2</sup> Quantum Cloud Computing and Distributed Systems (qCLOUDS) Lab, School of  
Computing and Information Systems, The University of Melbourne, Australia  
`rbuyya@unimelb.edu.au`

**Abstract.** We consider coded distributed computing for a batch of simultaneously arriving tasks, where tasks and workers are divided into corresponding groups and coding is applied within each group to mitigate stragglers. Optimizing the number of groups and the task and worker allocations is challenging because the grouping space grows combinatorially. We study this joint grouping problem under MDS and general coding. Under MDS coding, we prove that the single-group strategy is globally optimal for any distribution of worker computation times. Under general coding with linear recovery threshold, we derive an exact finite-sum expression for the expected batch makespan under shifted exponential worker computation times, enabling exact evaluation and optimization of feasible groupings. We develop a Grouping Branch and Bound (GBB) algorithm to obtain a globally optimal grouping and further propose a Slack Balanced Grouping Search (SBGS) algorithm to efficiently find groupings for larger instances within a prescribed computational budget. Experiments validate the MDS optimality result. Under the tested general coding settings, grouping substantially reduces the makespan, which is up to 66.1% higher under the single-group strategy. Moreover, SBGS attains the same optimal makespan as GBB in all tested instances and requires 95.84%-98.10% fewer expected makespan evaluations, indicating substantially lower computational cost. These results provide a theoretical basis and practical guidance for deciding when and how tasks and workers should be grouped under different coding schemes.

**Keywords:** Coded distributed computing · Batch processing · Makespan

## 1 Introduction

With the rapid growth of data-driven applications, distributed computing has become a fundamental infrastructure supporting a wide range of tasks. However, the overall performance of distributed systems is often limited by the straggler effect, in which some nodes randomly run slower than their normal pace due

to factors such as network fluctuations, resource contention, energy variability, or hardware and software failures [1]. Such stragglers are common and difficult to predict, and they can significantly delay task completion and degrade overall system performance, making their mitigation a key challenge in distributed computing systems.

Coded distributed computing has emerged as an effective approach to mitigating stragglers by encoding computations across workers and recovering a result from only a subset of their outputs [2]. It has been applied to matrix multiplication [3–5], gradient descent [6–8], multivariate polynomial evaluation [9–11], and deep neural network inference [12–14]. Existing studies have developed MDS, polynomial-based, and other code constructions to improve recovery thresholds, communication, numerical stability, or placement for a predetermined computation [2, 15, 16]. This substantial progress establishes coding as a powerful straggler-mitigation mechanism, but largely leaves the organization of a batch of independent tasks fixed before coding is applied.

Recent studies have extended coded distributed computing to batch processing, multiple task computation, and batch size optimization [17–19]. Task grouping has also been investigated to adjust computation granularity and exploit the work completed by stragglers [20]. Other studies allocate predetermined coded tasks among dedicated, shared, or heterogeneous workers [21, 22], while batch scheduling methods balance task loads and completion delays without modeling recovery thresholds induced by coding [23, 24]. Taken together, these studies optimize task organization or worker allocation separately rather than jointly partitioning a batch of tasks and a finite set of workers into corresponding coded groups. Such joint grouping is important because the task partition determines the recovery requirements, while the worker partition determines the recovery resources and completion times of the groups.

Motivated by this gap, we consider a batch of  $M$  simultaneously arriving tasks executed by  $N$  workers and jointly optimize the number of groups and the task and worker partitions under MDS and general coding to minimize the expected batch makespan. The problem is difficult because the number of feasible groupings grows combinatorially, while the makespan depends on the maximum of the group completion times. Moreover, increasing the number of groups can reduce the total recovery requirement under general coding but requires more groups to complete. Our analysis reveals how the recovery threshold governs this tradeoff and provides a theoretical and algorithmic basis for selecting and optimizing grouping strategies. The main contributions are summarized as follows.

- We develop a unified recovery-threshold-based framework for jointly grouping tasks and allocating workers to minimize the batch makespan in coded distributed computing.
- For MDS coding, we use a sample path comparison based on the global order of worker completion times to prove that the single-group strategy minimizes the batch makespan for every realization, and therefore for any computation time distribution.

- For general coding, we characterize how the number of groups determines the total recovery requirement and feasibility, and derive a finite sum expression for exactly evaluating the expected batch makespan under shifted exponential worker times. We then develop GBB, which combines rigorous lower bound pruning with canonical search to guarantee global optimality. For larger instances, we further propose SBGS, which refines multiple slack balanced initial groupings through task and worker moves to obtain high quality solutions with polynomial complexity.
- Experiments validate the MDS optimality result and demonstrate the benefit of grouping under general coding. Under the tested settings, the single-group strategy incurs up to 66.1% higher expected makespan than optimal grouping. SBGS matches the GBB optimum in all tested instances while reducing objective evaluations by 95.84%-98.10%.

The rest of the paper is organized as follows. Section 2 presents the system model and formulates the joint grouping problem. Section 3 establishes the optimal grouping under MDS coding. Section 4 derives the structural results for general coding and presents GBB and SBGS. Section 5 reports the experimental evaluation, and Section 6 concludes the paper along with future directions.

## 2 Problem Formulation

### 2.1 System Model

We consider a distributed computing system with  $N$  worker nodes. A batch of  $M$  homogeneous task requests arrives simultaneously, where  $N \geq M$ . To mitigate the impact of stragglers, the system employs coded computation. Following commonly adopted models in coded distributed computing [2, 25, 26], all tasks have the same normalized computational load, and encoding does not alter the load of each worker assignment.

For a group containing  $m$  original tasks, the recovery threshold is the minimum number of results returned by workers executing encoded tasks that are required to recover all  $m$  task outputs. We denote this threshold by  $d(m)$ . In this work, we consider two commonly used coding models.

- **MDS coding:** Maximum Distance Separable (MDS) codes are widely adopted in coded computation [2, 4, 6, 8, 12, 17–19, 21]. Under MDS coding, the recovery threshold is  $d(m) = m$ .
- **General coding:** More general coding schemes can exhibit a recovery threshold that grows linearly with the number of original tasks [9–11, 13, 15]. The corresponding recovery threshold is  $d(m) = cm - (c - 1)$ , where  $c \in \mathbb{Z}^+$ .

Let  $T_j$  be the completion time of worker  $j$ . Consistent with existing studies [6, 9, 19, 27], we assume that the worker completion times are independent and identically distributed. Following the commonly used shifted exponential model for distributed computing systems [2, 6, 16], we assume that  $T_j$  follows a shifted exponential distribution with a shift parameter  $\mu$  and a rate parameter  $\lambda$ , i.e.,  $T_j \sim \mu + \text{Exp}(\lambda)$ , where  $\mu \geq 0$  and  $\lambda > 0$ .

## 2.2 Joint Grouping and Batch Makespan

To further improve system performance, we jointly partition the tasks and workers into multiple groups. A grouping strategy is denoted by  $\mathcal{G} = (\ell, \mathbf{m}, \mathbf{n})$ , where  $\ell$  is the number of groups,  $\mathbf{m} = (m_1, \dots, m_\ell)$  is the task partition, and  $\mathbf{n} = (n_1, \dots, n_\ell)$  is the worker partition. A feasible strategy satisfies

$$m_i \geq 1, n_i \geq d(m_i), \quad \sum_{i=1}^{\ell} m_i = M, \quad \sum_{i=1}^{\ell} n_i = N, \quad m_i, n_i \in \mathbb{Z}^+, \forall i \in [\ell]. \quad (1)$$

Each worker is assigned to exactly one group. In group  $i$ , the  $m_i$  tasks are encoded across  $n_i$  workers, and the group completes upon receiving any  $d_i = d(m_i)$  worker results.

Let  $T_{d_i, n_i}$  denote the  $d_i$ -th smallest worker time in group  $i$ . Under the shifted exponential computation-time model, the completion time of group  $i$  has the following cumulative distribution function (CDF):

$$F_{d_i, n_i}(t) = \sum_{r=d_i}^{n_i} \binom{n_i}{r} \left(1 - e^{-\lambda(t-\mu)}\right)^r \left(e^{-\lambda(t-\mu)}\right)^{n_i-r}, \quad t \geq \mu. \quad (2)$$

The corresponding expected group completion time is

$$\mathbb{E}[T_{d_i, n_i}] = \mu + \frac{1}{\lambda}(H_{n_i} - H_{n_i-d_i}), \quad (3)$$

where  $H_k = \sum_{j=1}^k \frac{1}{j}$  is the  $k$ -th harmonic number.

Since all tasks arrive simultaneously, a batch is completed only when every group has finished. Therefore, the batch makespan under strategy  $\mathcal{G}$  is  $B(\mathcal{G}) = \max_{1 \leq i \leq \ell} T_{d_i, n_i}$ . Because different groups consist of non-overlapping sets of independent workers, their completion times are independent. Therefore, the CDF of the batch makespan can be written as

$$F_{B(\mathcal{G})}(t) = \prod_{i=1}^{\ell} F_{d_i, n_i}(t), \quad t \geq \mu. \quad (4)$$

Accordingly, the expected batch makespan is

$$\mathbb{E}[B(\mathcal{G})] = \mu + \int_{\mu}^{\infty} \left[1 - \prod_{i=1}^{\ell} F_{d_i, n_i}(t)\right] dt. \quad (5)$$

Our objective is to find a feasible grouping strategy that minimizes the expected batch makespan:

$$\min_{\ell, \mathbf{m}, \mathbf{n}} \mathbb{E}[B(\mathcal{G})], \quad \text{subject to} \quad (1). \quad (6)$$

The objective in (6) jointly determines the number of groups, the task partition, and the worker allocation. Two special cases are included in this formulation: the **Single-group Strategy**, given by  $\mathcal{G} = (1, (M), (N))$ , and **Replication Strategy**, given by  $\mathcal{G} = (M, (1, 1, \dots, 1), (n_1, n_2, \dots, n_M))$ .

### 3 Optimal Grouping for MDS Coding

In this section, we theoretically characterize the optimal grouping strategy for minimizing the expected batch makespan under MDS coding schemes.

**Theorem 1.** *Under MDS coding, for any given computation-time distribution, the single-group strategy achieves the minimum expected batch makespan.*

*Proof.* Denote the completion time of node  $i$  as  $v_i$ , and let the completion times of  $N$  nodes be ordered from slowest to fastest:  $v_N \geq v_{N-1} \geq \dots \geq v_1$ . If some nodes have the same completion time, their order can be determined arbitrarily but consistently.

Under the single-group MDS strategy with recovery threshold  $d(M) = M$ , the batch finishes when the  $M$ -th fastest node completes. We denote this completion time by  $A = v_M$ . For clarity, let  $S$  be the set of the  $M$  fastest nodes in the system, corresponding to the completion times  $v_M \geq v_{M-1} \geq \dots \geq v_1$ .

Now divide the  $N$  nodes into  $\ell$  groups. Group  $i$  contains  $n_i$  nodes and uses an  $(n_i, m_i)$  MDS code, so its finishing time is the completion time of the  $m_i$ -th fastest node in that group. Denote the ordered completion times within group  $i$  by  $v_{n_i}^{(i)} \geq v_{n_i-1}^{(i)} \geq \dots \geq v_1^{(i)}$ . So the group finishing time is  $v_{m_i}^{(i)}$ . The overall batch makespan is  $B = \max_{1 \leq i \leq \ell} v_{m_i}^{(i)}$ .

Let  $s_i$  denote the number of nodes in group  $i$  that belong to the globally fastest set  $S$ . Since  $\sum_{i=1}^{\ell} s_i = \sum_{i=1}^{\ell} m_i = M$ , there are two cases.

*Case 1:*  $s_{i^*} < m_{i^*}$  for some group  $i^*$ . This group contains fewer than  $m_{i^*}$  nodes from  $S$ , so its  $m_{i^*}$ -th fastest completion time is no smaller than  $A$ . Thus,

$$B = \max_{1 \leq i \leq \ell} v_{m_i}^{(i)} \geq v_{m_{i^*}}^{(i^*)} \geq A. \quad (7)$$

*Case 2:*  $s_i \geq m_i$  for every group  $i$ . The equality of the two sums implies  $s_i = m_i$  for all  $i$ . Let  $i^*$  be the group containing the node with completion time  $v_M = A$ . Since this group contains exactly  $m_{i^*}$  nodes from  $S$ , its  $m_{i^*}$ -th fastest completion time is  $A$ . Therefore,

$$B \geq v_{m_{i^*}}^{(i^*)} = A. \quad (8)$$

In both cases,  $B \geq A$  for every feasible grouping. Taking expectations gives  $\mathbb{E}[B] \geq \mathbb{E}[A]$ , proving the optimality of the single-group strategy.

### 4 Grouping Design for General Coding

Unlike MDS coding, general coding introduces a tradeoff in grouping. Dividing the tasks into more groups reduces the total recovery requirement, but also splits the available workers among more independent groups, making the batch more susceptible to a slow group. Consequently, obtaining a closed-form optimal grouping is difficult. In this section, we first derive structural properties for evaluating feasible groupings, then develop the Grouping Branch and Bound (GBB) algorithm that finds the globally optimal solution. To improve scalability for larger instances, we further propose the Slack Balanced Grouping Search (SBGS) algorithm.

#### 4.1 Structural Properties of Grouping

We consider the general recovery threshold  $d(m) = cm - (c - 1)$  with  $c \geq 2$ . For a partition with  $\ell$  task groups, the total recovery threshold is

$$D_\ell = \sum_{i=1}^{\ell} d(m_i) = d(M) - (c - 1)(\ell - 1). \quad (9)$$

Hence, each additional group reduces the total recovery threshold by  $c - 1$ . For a fixed  $\ell$ ,  $D_\ell$  depends only on the number of groups, while the task partition determines how the recovery requirements are distributed among them.

An  $\ell$ -group strategy is feasible only if the available workers can satisfy the total recovery requirement, i.e.,  $D_\ell \leq N$ , with  $\ell \leq M$ . This yields the minimum feasible number of groups

$$\ell_{\min} = \max \left\{ 1, \left\lceil \frac{cM - N}{c - 1} \right\rceil \right\}, \quad \ell_{\min} \leq \ell \leq M. \quad (10)$$

Since  $N \geq M$ , we have  $\ell_{\min} \leq M$ . If  $N \geq d(M)$ , then  $\ell_{\min} = 1$  and the single-group strategy is feasible. Otherwise, the tasks must be divided into at least  $\ell_{\min}$  groups to obtain a feasible strategy. Thus, under limited worker resources, grouping may be required to satisfy the recovery requirements.

Although increasing the number of groups reduces the total number of required results, it does not necessarily reduce the batch makespan. Each group must independently receive at least  $d_i$  worker results, so results from completed groups cannot help the other unfinished groups. Therefore, the grouping problem should balance the reduction in the total number of required results against the additional waiting caused by completing multiple groups separately.

To compare different feasible groupings, the expected batch makespan must be evaluated repeatedly. Since the integral form in (5) is inconvenient for evaluation, we derive an equivalent finite sum. From (2), each  $F_{d_i, n_i}(t)$  can be expressed as a polynomial in  $e^{-\lambda(t-\mu)}$  with degree  $n_i$ . Since  $\sum_{i=1}^{\ell} n_i = N$ , the batch completion distribution in (4) can be written as

$$\prod_{i=1}^{\ell} F_{d_i, n_i}(t) = 1 + \sum_{k=1}^N \beta_k e^{-k\lambda(t-\mu)}, \quad (11)$$

where the coefficients  $\beta_k$  are obtained by multiplying the group polynomials. Substituting (5) into (11) gives

$$\mathbb{E}[B(\mathcal{G})] = \mu - \sum_{k=1}^N \beta_k \int_{\mu}^{\infty} e^{-k\lambda(t-\mu)} dt = \mu - \frac{1}{\lambda} \sum_{k=1}^N \frac{\beta_k}{k}. \quad (12)$$

Thus, the expected batch makespan of any feasible grouping can be evaluated exactly through a finite sum without numerical integration.

## 4.2 Grouping Branch and Bound

The grouping problem in (6) involves a rapidly growing number of feasible strategies, making exhaustive search impractical. To solve this problem, we propose the Grouping Branch and Bound (GBB) algorithm, which systematically explores the grouping space and prunes noncompetitive branches using a lower bound, thereby reducing the search space while guaranteeing global optimality.

For a grouping strategy  $\mathcal{G}$  with  $\ell$  groups, let  $\mathcal{P} = \{(m_1, n_1), \dots, (m_p, n_p)\}$  be a partial grouping containing  $p$  fixed groups, where  $p \leq \ell$ . The numbers of tasks, workers, and groups that remain to be assigned are

$$M_R = M - \sum_{i=1}^p m_i, \quad N_R = N - \sum_{i=1}^p n_i, \quad \ell_R = \ell - p. \quad (13)$$

Using the relationship in (9), the total recovery requirement of the remaining groups is determined by  $M_R$  and  $\ell_R$ , and is given by  $D_R = cM_R - (c-1)\ell_R$ . Thus, a partial grouping is infeasible if  $M_R < \ell_R$  or  $N_R < D_R$ .

To construct a lower bound, we relax the completion requirement for the remaining groups. Instead of requiring each of the  $\ell_R$  groups to reach its own recovery threshold, we consider the remaining part complete once any  $D_R$  results are returned by the  $N_R$  remaining workers, regardless of which groups those workers would belong to.

**Lemma 1.** *Define the lower bound  $B_{\text{lb}}(\mathcal{P}) = \max \{T_{d_1, n_1}, \dots, T_{d_p, n_p}, T_{D_R, N_R}\}$ . Any feasible complete grouping  $\mathcal{G}$  extending  $\mathcal{P}$  satisfies  $B_{\text{lb}}(\mathcal{P}) \leq B(\mathcal{G})$ . Consequently,*

$$\mathbb{E}[B_{\text{lb}}(\mathcal{P})] \leq \mathbb{E}[B(\mathcal{G})]. \quad (14)$$

*Proof.* Let  $B_R(\mathcal{G}) = \max_{p+1 \leq i \leq \ell} T_{d_i, n_i}$  denote the completion time of the remaining groups. By time  $B_R(\mathcal{G})$ , each remaining group  $i$  has received at least  $d_i$  worker results. Since the worker groups are disjoint, at least  $\sum_{i=p+1}^{\ell} d_i = D_R$  results have been returned by the  $N_R$  remaining workers. Therefore,  $T_{D_R, N_R} \leq B_R(\mathcal{G})$ . It follows that

$$\begin{aligned} B_{\text{lb}}(\mathcal{P}) &= \max \{T_{d_1, n_1}, \dots, T_{d_p, n_p}, T_{D_R, N_R}\} \\ &\leq \max \{T_{d_1, n_1}, \dots, T_{d_p, n_p}, B_R(\mathcal{G})\} = B(\mathcal{G}). \end{aligned} \quad (15)$$

Taking expectations yields (14). This completes the proof.

The expectation  $\mathbb{E}[B_{\text{lb}}(\mathcal{P})]$  is evaluated by applying the finite expression in (12) to the  $p+1$  pairs  $(d_1, n_1), \dots, (d_p, n_p), (D_R, N_R)$ . By Lemma 1, if this value is no smaller than the expected makespan of the current best complete grouping, no feasible extension of  $\mathcal{P}$  can improve the current solution, and the corresponding branch can be pruned.

At each branching step, a pair  $(m, n)$  is selected for the next group and appended to  $\mathcal{P}$ . When  $\ell_R > 1$ , the selected pair should satisfy

$$1 \leq m \leq M_R - (\ell_R - 1), n \geq d(m), N_R - n \geq c(M_R - m) - (c-1)(\ell_R - 1). \quad (16)$$

Together, these constraints assign at least one task and enough workers for recovery to the new group, while leaving at least one task for each remaining group and sufficient workers to meet their total recovery requirement. When  $\ell_R = 1$ , all remaining tasks and workers are assigned to the final group, resulting in the pair  $(M_R, N_R)$ .

Different orders of the same group pairs represent the same grouping strategy. To avoid exploring these duplicate orders, we sort the pairs first by the number of tasks and then by the number of workers when their task numbers are equal. Specifically, when  $\mathcal{P}$  is nonempty, a new pair  $(m, n)$  can be appended if  $m > m_p$ , or if  $m = m_p$  and  $n \geq n_p$ . This ordering gives each feasible grouping a single sequence, so each grouping is explored exactly once.

GBB starts with a feasible complete grouping to establish an initial upper bound and enable pruning. We use uniform replication with  $\ell = M$  and  $m_i = 1$ , and distribute the workers as evenly as possible among the groups. This grouping is always feasible because  $d(1) = 1$  and  $N \geq M$ . The GBB algorithm is summarized in Algorithm 1.

**Theorem 2.** *The grouping strategy returned by Algorithm 1 is globally optimal.*

*Proof.* The outer loop considers all feasible group counts specified by (10). For each group count, the branching rules generate all feasible task and worker allocations. Every feasible grouping is arranged according to the lexicographic rule and thus appears once in the search.

A state violating (16) has no feasible extension. For any other pruned state  $\mathcal{P}$ , Lemma 1 ensures that every feasible extension  $\mathcal{G}$  satisfies  $\mathbb{E}[B(\mathcal{G})] \geq \mathbb{E}[B_{\text{lb}}(\mathcal{P})] = L_{\text{lb}} \geq L^*$ . Hence, pruning cannot discard a grouping that improves the incumbent.

The search space is finite, and every complete grouping retained by the search is evaluated using (12). Upon termination, the returned grouping  $\mathcal{G}^*$  has the minimum expected batch makespan among all feasible groupings.

For a fixed  $\ell$ , there are  $\binom{M-1}{\ell-1}$  ordered positive task partitions. By the expression for  $D_\ell$  in Section 4.1, the number of workers remaining after every group receives its required workers is  $N - D_\ell$ . Distributing these workers among the  $\ell$  groups gives  $\binom{N-D_\ell+\ell-1}{\ell-1}$  ordered worker allocations. Thus, before canonical ordering and pruning, the number of complete candidates is at most  $\binom{M-1}{\ell-1} \binom{N-D_\ell+\ell-1}{\ell-1}$ . Each expected-makespan evaluation requires  $\mathcal{O}(N^2)$  arithmetic operations to multiply the group polynomials. Since a root-to-leaf path contains at most  $M$  states, a conservative worst-case complexity is

$$\mathcal{O} \left( MN^2 \sum_{\ell=\ell_{\min}}^M \binom{M-1}{\ell-1} \binom{N-D_\ell+\ell-1}{\ell-1} \right). \quad (17)$$

The worst-case complexity is exponential in  $M$ , but the lower bound and canonical ordering can substantially reduce the number of states evaluated in practice.

**Algorithm 1:** Grouping Branch and Bound (GBB)

---

**Input:**  $M$ ,  $N$ , and  $c$   
**Output:** A globally optimal grouping  $\mathcal{G}^*$  and its expected makespan  $L^*$   
**Initialization:** Construct  $\mathcal{G}^*$  with  $\ell = M$ ,  $m_i = 1$ , and an even worker allocation,  $L^* \leftarrow \mathbb{E}[B(\mathcal{G}^*)]$  using (12),  
 $\ell_{\min} \leftarrow \max\{1, \lceil (cM - N)/(c - 1) \rceil\}$

```

1 for  $\ell \leftarrow \ell_{\min}$  to  $M$  do
2   Initialize  $\mathcal{Q}$  with the root state  $(\emptyset, M, N, \ell)$ 
3   while  $\mathcal{Q}$  is not empty do
4     Pop  $(\mathcal{P}, M_R, N_R, \ell_R)$  from  $\mathcal{Q}$ 
5      $D_R \leftarrow cM_R - (c - 1)\ell_R$ 
6     if  $M_R < \ell_R$  or  $N_R < D_R$  then
7       continue
8     end
9     if  $\ell_R = 1$  then
10      if  $(M_R, N_R)$  satisfies the lexicographic rule then
11        Append  $(M_R, N_R)$  to  $\mathcal{P}$  to obtain  $\mathcal{G}$ 
12         $L \leftarrow \mathbb{E}[B(\mathcal{G})]$  using (12)
13        if  $L < L^*$  then
14           $(\mathcal{G}^*, L^*) \leftarrow (\mathcal{G}, L)$ 
15      else
16         $L_{\text{lb}} \leftarrow \mathbb{E}[B_{\text{lb}}(\mathcal{P})]$  using (12)
17        if  $L_{\text{lb}} < L^*$  then
18          foreach  $(m, n)$  satisfying (16) and the lexicographic rule do
19            Append  $(m, n)$  to  $\mathcal{P}$  to obtain  $\mathcal{P}'$ 
20            Push  $(\mathcal{P}', M_R - m, N_R - n, \ell_R - 1)$  onto  $\mathcal{Q}$ 
21          end
22        end
23      end
24    end
25  end
26 return  $(\mathcal{G}^*, L^*)$ 

```

---

**4.3 Slack Balanced Grouping Search**

GBB returns a globally optimal solution, but its exponential worst-case complexity limits its scalability. To efficiently handle larger instances, we further propose the Slack Balanced Grouping Search (SBGS) algorithm. Since different initial groupings may lead to different local optima, for each feasible group number, SBGS starts from  $R$  initial groupings and performs at most  $I_{\max}$  best-improvement iterations from each start. The algorithm stops earlier if no feasible task or worker move strictly decreases the expected batch makespan. The best grouping obtained from all starts is returned.

Consider a fixed task partition with  $\ell$  groups. Let  $d_i = d(m_i)$  and  $D_\ell = \sum_{i=1}^{\ell} d_i$ . When group  $i$  becomes recoverable,  $d_i$  workers have completed their assignments, while the remaining  $n_i - d_i$  workers need not finish. We define

$s_i = n_i - d_i$  as the *worker slack* of group  $i$ . Since  $\sum_{i=1}^{\ell} n_i = N$ , the total worker slack is  $\sum_{i=1}^{\ell} s_i = N - D_{\ell}$ . The following proposition gives the allocation that maximizes the minimum worker slack among all groups.

**Proposition 1.** *For a fixed feasible task partition with  $\ell$  groups, we have*

$$\max_{\substack{\sum_{i=1}^{\ell} n_i = N \\ n_i \geq d_i}} \min_{1 \leq i \leq \ell} (n_i - d_i) = \left\lfloor \frac{N - D_{\ell}}{\ell} \right\rfloor. \quad (18)$$

*The maximum is attained by an allocation satisfying*

$$|(n_i - d_i) - (n_j - d_j)| \leq 1, \quad \forall i, j \in [\ell]. \quad (19)$$

*Proof.* After assigning  $d_i$  workers to every group,  $N - D_{\ell}$  workers remain. Hence, the minimum slack cannot exceed the average slack and is at most  $\lfloor (N - D_{\ell})/\ell \rfloor$ . Otherwise, every group would have slack at least  $\lfloor (N - D_{\ell})/\ell \rfloor + 1$ , making the total slack greater than  $N - D_{\ell}$ .

To attain this upper bound, we assign  $d_i + \lfloor (N - D_{\ell})/\ell \rfloor$  workers to each group  $i$  and distribute the remaining workers among distinct groups. Since fewer than  $\ell$  workers remain, assigning them to distinct groups ensures that each group receives at most one additional worker. The resulting slack values differ by at most one and have a minimum of  $\lfloor (N - D_{\ell})/\ell \rfloor$ . Therefore, this allocation satisfies (19) and attains the maximum in (18).

This result provides a feasible starting allocation that avoids assigning substantially less worker slack to one group than to the others. For every feasible group count  $\ell$ , SBGS uses  $R \geq 2$  initial task partitions. The first is the balanced partition, in which the task counts differ by at most one. The second is the maximally uneven partition  $(M - \ell + 1, 1, \dots, 1)$ . The remaining  $R - 2$  starts, if any, are generated randomly. For each start, we select  $\ell - 1$  distinct cut positions uniformly at random from the  $M - 1$  gaps between consecutive tasks. The cut positions define an ordered positive partition of  $M$  into  $\ell$  groups and yield a uniform distribution over all such partitions. For each task partition, SBGS computes  $d_i = d(m_i)$ , first assigns  $d_i$  workers to group  $i$ , and then distributes the remaining  $N - D_{\ell}$  workers according to (19). If  $N - D_{\ell}$  is not divisible by  $\ell$ , the workers left are assigned to different groups across the initializations.

Starting from each slack-balanced initialization, SBGS explores both task and worker allocations through two move types.

- A *worker move* transfers one worker from group  $i$  to group  $j$  while keeping the task partition unchanged. The move is feasible if  $n_i > d_i$ .
- A *task move* transfers one task from group  $i$  to group  $j$ , where  $m_i \geq 2$ . Under  $d(m) = cm - (c - 1)$ , the recovery requirements of the source and destination groups decrease and increase by  $c$ , respectively. Transferring  $c$  workers with the task preserves the worker slack of both groups, specifically,  $(m_i, n_i) \leftarrow (m_i - 1, n_i - c)$ ,  $(m_j, n_j) \leftarrow (m_j + 1, n_j + c)$ . Therefore, the task move also preserves feasibility.

**Algorithm 2:** Slack Balanced Grouping Search (SBGS)

---

**Input:**  $M, N, c$ , the number of starts  $R \geq 2$ , and the maximum number of iterations  $I_{\max} \geq 1$   
**Output:** A grouping  $\widehat{\mathcal{G}}$  and its expected batch makespan  $\widehat{L}$   
**Initialization:**  $\widehat{L} \leftarrow \infty, \ell_{\min} \leftarrow \max\{1, \lceil (cM - N)/(c - 1) \rceil\}$

```

1 for  $\ell \leftarrow \ell_{\min}$  to  $M$  do
2   Generate the balanced and maximally uneven task partitions
3   Generate  $R - 2$  random positive integer partitions
4   foreach initial task partition  $\mathbf{m}$  do
5     Compute  $d_i = d(m_i)$  for every group
6     Allocate the workers according to (19)
7      $\mathcal{G} \leftarrow (\ell, \mathbf{m}, \mathbf{n})$ 
8      $L \leftarrow \mathbb{E}[B(\mathcal{G})]$  using (12)
9      $I \leftarrow 0$ ,  $\text{improved} \leftarrow \text{true}$ 
10    while  $\text{improved}$  and  $I < I_{\max}$  do
11       $I \leftarrow I + 1$ ,  $\text{improved} \leftarrow \text{false}$ 
12       $(\mathcal{G}', L') \leftarrow (\mathcal{G}, L)$ 
13      foreach feasible worker move or task move producing  $\mathcal{H}$  do
14         $U \leftarrow \mathbb{E}[B(\mathcal{H})]$ 
15        if  $U < L'$  then
16           $(\mathcal{G}', L') \leftarrow (\mathcal{H}, U)$ 
17        end
18        if  $L' < L$  then
19           $(\mathcal{G}, L) \leftarrow (\mathcal{G}', L')$ 
20           $\text{improved} \leftarrow \text{true}$ 
21      end
22      if  $L < \widehat{L}$  then
23         $(\widehat{\mathcal{G}}, \widehat{L}) \leftarrow (\mathcal{G}, L)$ 
24      end
25    end
26 end
27 return  $(\widehat{\mathcal{G}}, \widehat{L})$ 

```

---

Both moves preserve the total numbers of tasks and workers and maintain feasibility. At each iteration, SBGS evaluates all feasible neighbors using (12) and moves to the one with the smallest expected batch makespan if it strictly improves the current grouping. The search terminates when no improving neighbor exists or when  $I_{\max}$  iterations have been performed. In the former case, the result is locally optimal with respect to the worker and task moves. In the latter case, the algorithm returns the best grouping found within the prescribed iteration budget. The SBGS algorithm is summarized in Algorithm 2.

For a grouping with  $\ell$  groups, a worker move selects an ordered source-destination pair, so there are at most  $\ell(\ell - 1)$  feasible worker-move candidates. The same upper bound applies to task moves. Therefore, one search iteration evaluates at most  $2\ell(\ell - 1)$  neighboring groupings. Since each exact expected

makespan evaluation requires  $\mathcal{O}(N^2)$  arithmetic operations, the total complexity of SBGS is

$$\mathcal{O}\left(RI_{\max}N^2 \sum_{\ell=\ell_{\min}}^M \ell(\ell-1)\right) = \mathcal{O}(RI_{\max}M^3N^2). \quad (20)$$

For fixed  $R$  and  $I_{\max}$ , SBGS has polynomial complexity  $\mathcal{O}(M^3N^2)$ . The solution obtained by SBGS can be used as the initial incumbent for GBB, providing a tighter upper bound and enabling more effective pruning without affecting the global optimality guarantee of GBB.

## 5 Performance Evaluation

We validate the optimality of single-group strategy under MDS coding and then evaluate the solution quality and computational efficiency of GBB and SBGS under general coding by comparing them with representative grouping strategies.

### 5.1 Experimental Set-up

Following the shifted exponential setting in [28], we set the shift and rate parameters to  $\mu = 1$  and  $\lambda = 0.5$ , respectively. We consider MDS coding with  $d(m) = m$  and general coding with  $d(m) = 2m - 1$ . For both coding schemes, the number of workers is fixed at  $N = 50$ , while the number of tasks is varied starting from  $M = 5$ . We consider the following benchmark grouping strategies.

- **Single-group** All  $M$  tasks and  $N$  workers form one group, with  $\ell = 1$ ,  $\mathbf{m} = (M)$ , and  $\mathbf{n} = (N)$ .
- **Uniform** For a fixed  $2 \leq \ell < M$ , this method divides the tasks and the nodes as evenly as possible, so that  $|m_i - m_j| \leq 1$  and  $|n_i - n_j| \leq 1$  for  $\forall i, j \in \{1, \dots, \ell\}$ .
- **Replication** Each task forms a separate group, with  $\ell = M$  and  $m_i = 1$  for all  $i \in [M]$ . The workers are divided as evenly as possible among the  $M$  groups.

Under MDS coding, we compare the benchmark strategies above. Under general coding, we additionally evaluate GBB and SBGS. GBB obtains the globally optimal grouping using Algorithm 1, while SBGS returns the best grouping found within its prescribed search budget using Algorithm 2. For SBGS, we set the number of starts to  $R = 5$  and the maximum number of best-improvement iterations per start to  $I_{\max} = 20$ .

### 5.2 Evaluation Results

We evaluate different grouping strategies using both simulations and numerical analysis. In the simulations, 2000 batches of size  $M$  are simulated to estimate

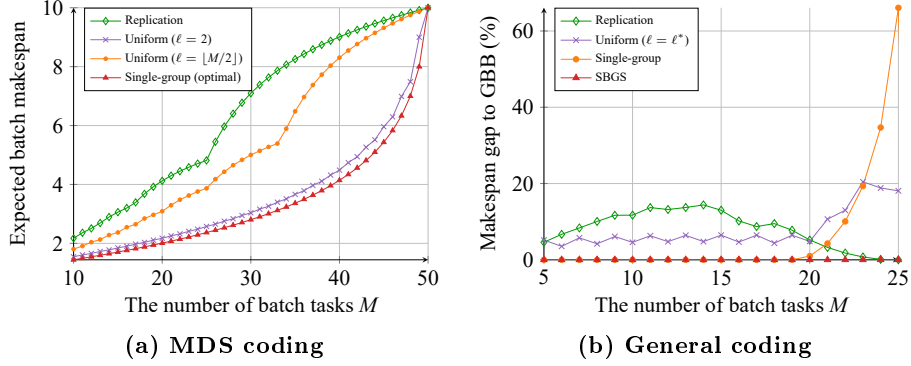


**Fig. 1.** Expected batch makespan under MDS and general coding with  $N = 50$ ,  $\mu = 1$ , and  $\lambda = 0.5$ . In (b),  $\ell^*$  denotes the best feasible group count under the uniform strategy.

the expected performance. The numerical analysis computes the expected batch makespan directly for each grouping. Since the simulation and numerical results differ only at the  $10^{-3}$  level, we present only the numerical results as representative in the following.

Figure 1(a) compares the four grouping strategies under MDS coding and shows that the single-group strategy achieves the smallest expected batch makespan for all tested values of  $M$ , validating the theoretical result proved in Section 3. Replication generally performs worst because it creates the largest number of groups, increasing the likelihood that the batch is delayed by a slow group.

Figure 1(b) reports the relative makespan gap, defined as  $(L - L_{\text{GBB}})/L_{\text{GBB}} \times 100\%$ , where  $L_{\text{GBB}}$  is the globally optimal value returned by GBB. Under general coding, the total recovery requirement decreases with the number of groups. The Single-group strategy has zero gap for  $5 \leq M \leq 19$ , confirming its optimality when sufficient worker redundancy is available. Its gap becomes positive at  $M = 20$  and increases rapidly thereafter, reaching 66.1% at  $M = 25$ . This shows that grouping becomes increasingly beneficial as the single group recovery requirement approaches the total number of workers. Uniform grouping generally performs better than Replication for smaller  $M$ , but its gap increases beyond 10% for  $M \geq 21$  and reaches 18.1% at  $M = 25$ . In contrast, the gap of Replication decreases from 5.2% at  $M = 20$  to 0% at  $M = 25$ , because using more groups reduces the total recovery requirement and becomes more advantageous as worker redundancy diminishes. The SBGS matches GBB throughout the tested range and consequently maintains a zero gap, demonstrating its effectiveness in finding high-quality groupings within the prescribed search budget.

We next compare the computational efficiency of GBB and SBGS. Let  $E_{\text{lb}}$  and  $E_{\text{leaf}}$  denote the numbers of lower-bound evaluations and complete grouping evaluations in GBB, respectively. Including the evaluation of its initial incumbent, the total count is  $E_{\text{GBB}} = E_{\text{lb}} + E_{\text{leaf}} + 1$ . For SBGS, let  $E_{\text{init}}$ ,  $E_{\text{worker}}$ , and  $E_{\text{task}}$  denote the numbers of previously unevaluated initial groupings, worker-

**Table 1.** Empirical search effort of GBB and SBGS.

| $M$ | $E_{\text{GBB}}$ | $E_{\text{SBGS}}$ | Work ratio | Reduction (%) |
|-----|------------------|-------------------|------------|---------------|
| 5   | 1,948            | 81                | 24.05      | 95.84         |
| 10  | 22,858           | 434               | 52.67      | 98.10         |
| 15  | 94,797           | 3,010             | 31.49      | 96.82         |
| 20  | 264,115          | 6,291             | 41.98      | 97.62         |
| 25  | 116,637          | 3,742             | 31.17      | 96.79         |
| 30  | 143,147          | 5,915             | 24.20      | 95.87         |

move candidates, and task-move candidates, respectively. Candidates retrieved from the cache are not counted again. Thus,  $E_{\text{SBGS}} = E_{\text{init}} + E_{\text{worker}} + E_{\text{task}}$ . The two comparison metrics in Table 1 are calculated as

$$\text{Workratio} = \frac{E_{\text{GBB}}}{E_{\text{SBGS}}}, \quad \text{Reduction} = 100 \left( 1 - \frac{E_{\text{SBGS}}}{E_{\text{GBB}}} \right) \%. \quad (21)$$

They measure the observed search effort rather than the asymptotic worst-case complexity of either algorithm.

Table 1 compares the computational costs of GBB and SBGS in terms of objective evaluations. Across all reported problem sizes, SBGS reduces the evaluation count by more than one order of magnitude. Specifically, GBB performs 24.05 to 52.67 times as many evaluations as SBGS, corresponding to a reduction of 95.84%-98.10%. The counts vary nonmonotonically with  $M$  because the feasible grouping space changes with the task number. For  $c = 2$  and  $N = 50$ ,  $\ell_{\min} = \max\{1, 2M - 50\}$ . Once  $M > 25$ ,  $\ell_{\min}$  increases with  $M$ , and the feasible range of group counts gradually narrows. Consequently, GBB explores fewer branches after feasibility checks and lower-bound pruning, while SBGS examines fewer feasible moves and can terminate sooner. Together with Fig. 1(b), these results show that SBGS obtains high-quality groupings with substantially fewer objective evaluations, demonstrating its efficiency.

## 6 Conclusions and Future Work

This paper formulated the joint task-grouping and worker-allocation problem for minimizing the expected makespan of coded batch computation. We first proved that the single-group strategy is optimal under MDS coding for arbitrary worker-time distributions. For the general coding scheme with threshold  $d(m) = cm - (c - 1)$ , we characterized how the number of groups determines the total recovery requirement and feasibility and derived an exact finite-sum expression for the expected batch makespan under shifted exponential worker times. Building on these results, we first developed GBB, which returns a globally optimal grouping through lower-bound pruning. To extend the framework to larger instances, we further developed SBGS, a polynomial-time local search that improves several slack-balanced initial groupings by reallocating tasks and

workers, obtaining high-quality solutions with far fewer objective evaluations than GBB. Experiments show that grouping becomes beneficial as worker redundancy decreases; moreover, SBGS matches the GBB optimum in all tested instances while reducing objective evaluations by 95.84%–98.10%. These results clarify when grouping can improve coded batch completion, offer guidance on how tasks and workers should be grouped, and provide both exact and scalable solution approaches. Future work will extend the proposed framework to broader distributed computing settings and practical system constraints.

## References

1. Mehmet Fatih Aktaş and Emina Soljanin. Straggler mitigation at scale. *IEEE/ACM Transactions on Networking*, 27(6):2266–2279, 2019.
2. Kangwook Lee, Maximilian Lam, Ramtin Pedarsani, Dimitris Papailiopoulos, and Kannan Ramchandran. Speeding up distributed machine learning using codes. *IEEE Transactions on Information Theory*, 64(3):1514–1529, 2017.
3. Marvin Xhemrishi, Rawad Bitar, and Antonia Wachter-Zeh. Distributed matrix-vector multiplication with sparsity and privacy guarantees. In *2022 IEEE International Symposium on Information Theory (ISIT)*, pages 1028–1033, 2022.
4. Weiqi Li, Zhen Chen, Zhiying Wang, Syed A Jafar, and Hamid Jafarkhani. Flexible distributed matrix multiplication. *IEEE Transactions on Information Theory*, 68(11):7500–7514, 2022.
5. Mingjun Dai, Zelong Zhang, Ziyang Zheng, Zhonghao Zhang, Xiaohui Lin, and Hui Wang. 2d-sazd: A novel 2d coded distributed computing framework for matrix-matrix multiplication. *IEEE Transactions on Services Computing*, 17(3):705–717, 2024.
6. Amirhossein Reisizadeh, Saurav Prakash, Ramtin Pedarsani, and Amir Salman Avestimehr. Codedreduce: A fast and robust framework for gradient aggregation in distributed learning. *IEEE/ACM Transactions on Networking*, 30(1):148–161, 2021.
7. Weiheng Tang, Jingyi Li, Lin Chen, and Xu Chen. Design and optimization of hierarchical gradient coding for distributed learning at edge devices. *IEEE Transactions on Communications*, 72(12):7727–7741, 2024.
8. Chengxi Li and Mikael Skoglund. Distributed learning based on 1-bit gradient coding in the presence of stragglers. *IEEE Transactions on Communications*, 72(8):4903–4916, 2024.
9. Qian Yu, Songze Li, Netanel Raviv, Seyed Mohammadreza Mousavi Kalan, Mahdi Soltanolkotabi, and Salman A Avestimehr. Lagrange coded computing: Optimal design for resiliency, security, and privacy. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pages 1215–1225, 2019.
10. Tayyeb Jahani-Nezhad and Mohammad Ali Maddah-Ali. Berrut approximated coded computing: Straggler resistance beyond polynomial computing. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(1):111–122, 2022.
11. Wilton Kim, Stanislav Kruglik, and Han Mao Kiah. Verifiable coded computation of multiple functions. *IEEE Transactions on Information Forensics and Security*, 19:8009–8022, 2024.
12. Jack Kosaian, KV Rashmi, and Shivaram Venkataraman. Parity models: erasure-coded resilience for prediction serving systems. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles (SOSP)*, pages 30–46, 2019.

13. Jack Kosaian, KV Rashmi, and Shivaram Venkataraman. Learning-based coded computation. *IEEE Journal on Selected Areas in Information Theory*, 1(1):227–236, 2020.
14. Jingzhe Zhang, Xiaofan He, and Huaiyu Dai. Speeding up distributed learning via sparse and flexible coded computing. *IEEE Transactions on Information Theory*, 71(4):3167–3180, 2025.
15. Mohammad Fahim and Viveck R Cadambe. Numerically stable polynomially coded computing. *IEEE Transactions on Information Theory*, 67(5):2758–2785, 2021.
16. Yuhan Wang and Youlong Wu. Coded distributed computing with pre-set data placement and output functions assignment. *IEEE Transactions on Information Theory*, 71(3):2195–2217, 2025.
17. Baoqian Wang, Junfei Xie, Kejie Lu, Yan Wan, and Shengli Fu. On batch-processing based coded computing for heterogeneous distributed computing systems. *IEEE Transactions on Network Science and Engineering*, 8(3):2438–2454, 2021.
18. Zhongming Ji, Li Chen, Hongguang Fu, Xinghua Zhao, Jun Xu, Lingkun Meng, and Junwei Liu. Multiple-task coded computing for distributed computation framework: Modeling and delay analysis. *IEEE Transactions on Communications*, 72(8):5032–5046, 2024.
19. Swapnil Saha, Emina Soljanin, and Philip Whiting. On optimal batch size in coded computing. In *2025 IEEE International Symposium on Information Theory (ISIT)*, pages 1–6. IEEE, 2025.
20. Tharindu Adikari, Haider Al-Lawati, Jason Lam, Zhenhua Hu, and Stark Draper. Straggler exploitation in distributed computing systems with task grouping. In *2023 59th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pages 1–7, 2023.
21. Daejin Kim, Hyegyeong Park, Dusit Niyato, and Junkyun Choi. Worker assignment for multiple masters to speed up coded distributed computing in heterogeneous clusters. *IEEE Trans. Serv. Comput.*, 16(3):2283–2298, 2023.
22. Yuxuan Sun, Fan Zhang, Junlin Zhao, Sheng Zhou, Zhisheng Niu, and Deniz Gündüz. Coded computation across shared heterogeneous workers with communication delay. *IEEE Transactions on Signal Processing*, 70:3371–3385, 2022.
23. Sophiya Shiekh, Mohammad Shahid, Manas Sambare, Raza Abbas Haidri, and Dileep Kumar Yadav. A load-balanced hybrid heuristic for allocation of batch of tasks in cloud computing environment. *International Journal of Pervasive Computing and Communications*, 19(5):756–781, 2023.
24. Tiangang Li, Shi Ying, Yishi Zhao, and Jianga Shang. Batch jobs load balancing scheduling in cloud computing using distributional reinforcement learning. *IEEE Transactions on Parallel and Distributed Systems*, 35(1):169–185, 2023.
25. Rashish Tandon, Qi Lei, Alexandros G. Dimakis, and Nikos Karampatziakis. Gradient coding: Avoiding stragglers in distributed learning. In *Proceedings of the 34th International Conference on Machine Learning*, pages 3368–3376, 2017.
26. Qian Yu, Mohammad Ali Maddah-Ali, and A. Salman Avestimehr. Polynomial codes: An optimal design for high-dimensional coded matrix multiplication. In *Advances in Neural Information Processing Systems*, 2017.
27. Anindya Bijoy Das, Aditya Ramamoorthy, David J Love, and Christopher G Brinton. Sparsity-preserving encodings for straggler-optimal distributed matrix computations at the edge. *IEEE Internet Things J.*, 11(21):34455–34470, 2024.
28. Ajay Badita, Parimal Parag, and Vaneet Aggarwal. Single-forking of coded subtasks for straggler mitigation. *IEEE/ACM Transactions on Networking*, 29(6):2413–2424, December 2021.