**Block-Diagonal Coding for Distributed Computing With Straggling Servers**

# Block-Diagonal Coding for Distributed Computing With Straggling Servers

Albin Severinson[†] [‡], Alexandre Graell i Amat[†], and Eirik Rosnes[‡]
[†]Department of Electrical Engineering, Chalmers University of Technology, Gothenburg, Sweden
[‡]Simula@UiB, Bergen, Norway

***Abstract*—We consider the distributed computing problem of multiplying a set of vectors with a matrix. For this scenario, Li *et al.* recently presented a unified coding framework and showed a fundamental tradeoff between computational delay and communication load. This coding framework is based on maximum distance separable (MDS) codes of code length proportional to the number of rows of the matrix, which can be very large. We propose a block-diagonal coding scheme consisting of partitioning the matrix into submatrices and encoding each submatrix using a shorter MDS code. We show that the assignment of coded matrix rows to servers to minimize the communication load can be formulated as an integer program with a nonlinear cost function, and propose an algorithm to solve it. We further prove that, up to a level of partitioning, the proposed scheme does not incur any loss in terms of computational delay (as defined by Li *et al.*) and communication load compared to the scheme by Li *et al.*. We also show numerically that, when the decoding time is also taken into account, the proposed scheme significantly lowers the overall computational delay with respect to the scheme by Li *et al.*. For heavy partitioning, this is achieved at the expense of a slight increase in communication load.**

## I. Introduction

Distributed computing has emerged as one of the most effective ways of tackling increasingly complex computation problems. One of the main application areas is large-scale machine learning and data analytics. Google routinely performs computations using several thousands of servers in their MapReduce clusters [1]. Distributed computing systems bring significant challenges. Among them, the problems of straggling servers and bandwidth scarcity have recently received significant attention. The straggler problem is a synchronization problem characterized by the fact that a distributed computing task must wait for the slowest server to complete its computation. On the other hand, distributed computing tasks typically require that data is moved between servers during the computation, the so-called *data shuffling*, which is a challenge in bandwidth-constrained networks.

Coding for distributed computing to reduce the computational delay and the communication load between servers has recently been considered in [2], [3]. In [2], a structure of repeated computation tasks across servers was proposed, enabling coded multicast opportunities that significantly reduce the required bandwidth to shuffle the results. In [3], the authors showed that maximum distance separable (MDS) codes can be applied to a linear computation task (e.g., multiplying a vector with a matrix) to alleviate the effects of straggling servers and reduce the computational delay. In [4], a unified coding framework was presented and a fundamental tradeoff between computational delay and communication load was identified. The ideas of [2], [3] can be seen as particular instances of the framework in [4], corresponding to the minimization of the communication load or the computational delay.

The distributed computing problem addressed in [4] is a matrix multiplication problem where a set of vectors $\boldsymbol{x}_1, \dots, \boldsymbol{x}_N$ are multiplied by a matrix $\boldsymbol{A}$. In particular, as in [3], the authors suggest the use of MDS codes, whose dimension is equal to the number of rows of $\boldsymbol{A}$, to generate some redundant computations. In practice, the size of $\boldsymbol{A}$ can be very large. For example, Google performs matrix-vector multiplications with matrices of dimension of the order of $10^{10} \times 10^{10}$ when ranking the importance of websites [5]. Since the decoding complexity of MDS codes on the packet erasure channel is quadratic (for Reed-Solomon (RS) codes) in the code length [6], for very large matrix sizes the decoding complexity may be prohibitively high.

In this paper, we introduce a block-diagonal encoding scheme for the distributed matrix multiplication problem. The proposed encoding is equivalent to partitioning the matrix and applying smaller MDS codes to each submatrix separately. The storage design for the proposed block-diagonal encoding can be cast as an integer optimization problem with a nonlinear objective function, whose computation scales exponentially with the problem size. We propose a heuristic solver for efficiently solving the optimization problem, and a branch-and-bound approach for improving on the resulting solution iteratively. We exploit a dynamic programming approach to speed up the branch-and-bound operations. We prove that up to a certain partitioning level, partitioning does not increase the computational delay (as defined in [4]) and the communication load with respect to the scheme in [4]. Interestingly, when the decoding time is taken into account, the proposed scheme achieves an overall computational delay significantly lower than the one of the scheme in [4]. This is due to the fact that the proposed scheme allows using significantly shorter MDS codes, hence reducing the decoding complexity and the decoding time. Also, numerical results show that a high level of partitioning can be applied at the expense of only a slight increase in the communication load.

This work was funded by the Swedish Research Council under grant 2016-04253 and the Research Council of Norway under grant 240985/F20.

## II. System Model

We consider the problem of multiplying a set of vectors with a matrix. In particular, given a matrix $\boldsymbol{A} \in \mathbb{F}^{m \times n}$ and $N$ vectors $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N \in \mathbb{F}^n$, where $\mathbb{F}$ is some field, we want to compute the $N$ vectors $\boldsymbol{y}_1 = \boldsymbol{A}\boldsymbol{x}_1, \ldots, \boldsymbol{y}_N = \boldsymbol{A}\boldsymbol{x}_N$. The computation is performed in a distributed fashion using $K$ servers, $S_1, \ldots, S_K$, where each server stores $\mu m$ matrix rows, for some $\frac{1}{K} \leq \mu \leq 1$. Prior to distributing the rows among the servers, $\boldsymbol{A}$ is encoded by an $r \times m$ encoding matrix $\boldsymbol{\Psi}$, resulting in the coded matrix $\boldsymbol{C} = \boldsymbol{\Psi}\boldsymbol{A}$, of size $r \times n$, i.e., the rows of $\boldsymbol{A}$ are encoded using an $(r, m)$ linear code with $r \geq m$.

Let $q = K\frac{m}{r}$, where we assume that $r$ divides $Km$ and hence $q$ is an integer. The $r$ coded rows of $\boldsymbol{C}$, $\boldsymbol{c}_1, \ldots, \boldsymbol{c}_r$, are divided into $\binom{K}{\mu q}$ disjoint batches, each containing $r/\binom{K}{\mu q}$ coded rows. Each batch is assigned to $\mu q$ servers. Correspondingly, a batch $B$ is labeled by a unique set $\mathcal{T} \subset \{S_1, \ldots, S_K\}$, of size $|\mathcal{T}| = \mu q$, denoting the subset of servers that store that batch, and we write $B_{\mathcal{T}}$. Server $S_k$, $k = 1, \ldots, K$, stores the coded rows of $B_{\mathcal{T}}$ if and only if $S_k \in \mathcal{T}$.

### *A. Probabilistic Runtime Model*

We adopt the probabilistic model of the computation runtime of [3]. We assume that running a computation on a single server takes a random amount of time according to the shifted-exponential cumulative probability distribution

$$F(t) = \begin{cases} 1 - \mathrm{e}^{-\left(\frac{t}{\sigma} - 1\right)}, & \text{for } t \geq \sigma \\ 0, & \text{otherwise} \end{cases},$$

where $\sigma$ is the number of multiplications and divisions required to complete the computation. We do not take addition and subtraction into account as those operations are orders of magnitude faster [7].

When the computation is split into $K$ parallel subtasks running on separate servers, we assume that the runtimes of the subtasks are independent and identically distributed random variables with distribution $F(Kt)$ [3]. Furthermore, the runtime of the $g$-th, $g = 1, \ldots, K$, fastest server to complete its subtask is given by the $g$-th order statistic, $F_{(g)}$, with expectation [8]

$$f(\sigma, K, g) \triangleq \mathbb{E}\left(F_{(g)}\right) = \sigma\left(1 + \sum_{j=K-g+1}^{K} \frac{1}{j}\right).$$

### *B. Distributed Computing Model*

We consider the MapReduce framework described in [4], where we assume that the input vectors $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$ are known to all servers. The overall computation then proceeds in three phases: *map*, *shuffle*, and *reduce*.

*1) Map Phase:* In the map phase, we compute in a distributed fashion coded intermediate values, which will be later used to obtain vectors $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_N$. Server $S$ multiplies the input vectors $\boldsymbol{x}_j$, $j = 1, \ldots, N$, by all the coded rows of matrix $\boldsymbol{C}$ it stores, i.e., it computes

$$\mathcal{Z}_j^{(S)} \triangleq \{\boldsymbol{c}\boldsymbol{x}_j : \boldsymbol{c} \in \{B_{\mathcal{T}} : \; S \in \mathcal{T}\}\},\ j = 1, \ldots, N.$$

The map phase terminates when a set of servers $\mathcal{G} \subseteq \{S_1, \ldots, S_K\}$ that collectively store enough values to decode the output vectors have finished their map computations. We denote the cardinality of $\mathcal{G}$ by $g$. The $(r, m)$ linear code proposed in [4] is an MDS code for which $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_N$ can be obtained from any subset of $q$ servers, i.e., $g = q$.

We define the computational delay of the map phase as its average runtime per source row and vector $\boldsymbol{y}$, i.e., $D_{\mathsf{map}} = \frac{1}{mN} f(\frac{\sigma_{\mathsf{map}}}{K}, K, g)$. $D_{\mathsf{map}}$ is referred to simply as the computational delay in [4]. As all $K$ servers compute $\mu m$ inner products, each requiring $n$ multiplications for each of the $N$ input vectors, we have $\sigma_{\mathsf{map}} = K\mu m n N$.

After the map phase, the computation of $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_N$ proceeds using only the servers in $\mathcal{G}$. We denote by $\mathcal{Q} \subseteq \mathcal{G}$ the set of the first $q$ servers to complete the map phase. Each of the $q$ servers in $\mathcal{Q}$ is responsible to compute $N/q$ of the vectors $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_N$. Let $\mathcal{W}_S$ be the set containing the indices of the vectors $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_N$ server $S \in \mathcal{Q}$ is responsible for. The remaining servers in $\mathcal{G}$ assist the servers in $\mathcal{Q}$ in the shuffle phase.

*2) Shuffle Phase:* In the shuffle phase, intermediate values calculated in the map phase are exchanged between servers in $\mathcal{G}$ until all servers in $\mathcal{Q}$ hold enough values to compute the vectors they are responsible for. As in [4], we allow creating and multicasting coded messages that are simultaneously useful for multiple servers. For a subset of servers $\mathcal{S} \subset \mathcal{Q}$ and $S \in \mathcal{Q} \backslash \mathcal{S}$, we denote the set of intermediate values needed by server $S$ and known *exclusively* by the servers in $\mathcal{S}$ by $\mathcal{V}_{\mathcal{S}}^{(S)}$. More formally,

$$\mathcal{V}_{\mathcal{S}}^{(S)} \triangleq \{\boldsymbol{c}\boldsymbol{x}_j : \; j \in \mathcal{W}_S \text{ and } \boldsymbol{c} \in \{B_{\mathcal{T}} : \; \mathcal{T} \cap \mathcal{Q} = \mathcal{S}\}\}.$$

Let $\alpha_j \triangleq \frac{\binom{q-1}{j}\binom{K-q}{\mu q - j}}{\frac{q}{K}\binom{K}{\mu q}}$ and $s_q \triangleq \inf\left(s : \sum_{l=s}^{\mu q} \alpha_l \leq 1 - \mu\right)$. For each $j \in \{\mu q, \mu q - 1, \ldots, s_q\}$, and every subset $\mathcal{S} \subseteq \mathcal{Q}$ of size $j + 1$, the shuffle phase proceeds as follows.

1) For each $S \in \mathcal{S}$, we evenly and arbitrarily split $\mathcal{V}_{\mathcal{S} \backslash S}^{(S)}$ into $j$ disjoint segments $\mathcal{V}_{\mathcal{S} \backslash S}^{(S)} = \{\mathcal{V}_{\mathcal{S} \backslash S, \tilde{S}}^{(S)} : \tilde{S} \in \mathcal{S} \setminus S\}$, and associate the segment $\mathcal{V}_{\mathcal{S} \backslash S, \tilde{S}}^{(S)}$ with server $\tilde{S} \in \mathcal{S} \backslash S$.
2) Server $\tilde{S} \in \mathcal{S}$ multicasts the bit-wise XOR of all the segments associated with it in $\mathcal{S}$. More precisely, it multicasts $\oplus_{S \in \mathcal{S} \backslash \tilde{S}} \mathcal{V}_{\mathcal{S} \backslash S, \tilde{S}}^{(S)}$ to the other servers in $\mathcal{S} \setminus \tilde{S}$.

For every pair of servers $S, \tilde{S} \in \mathcal{S}$, since server $S$ has computed locally the segments $\mathcal{V}_{\mathcal{S} \backslash S', \tilde{S}}^{(S')}$ for all $S' \in \mathcal{S} \backslash \{\tilde{S}, S\}$, it can cancel them from the message sent by server $\tilde{S}$, and recover the intended segment. We finish the shuffle phase by either unicasting any remaining needed values until all servers in $\mathcal{Q}$ hold enough intermediate values to decode successfully, or by repeating the above two steps for $j = s_q - 1$, selecting the strategy achieving the lower communication load.

**Definition 1.** *The communication load, denoted by L, is the number of messages per source row and vector $\boldsymbol{y}$ exchanged during the shuffle phase, i.e., the total number of messages sent during the shuffle phase divided by $mN$.*

Depending on the strategy, the communication load after completing the multicast phase is $\sum_{j=s_q}^{\mu q} \frac{\alpha_j}{j}$ or $\sum_{j=s_q-1}^{\mu q} \frac{\alpha_j}{j}$ [4]. For the scheme in [4], the total communication load is

$$L_{\mathsf{MDS}} = \min\left( \sum_{j=s_q}^{\mu q} \frac{\alpha_j}{j} + 1 - \mu - \sum_{j=s_q}^{\mu q} \alpha_j, \sum_{j=s_q-1}^{\mu q} \frac{\alpha_j}{j} \right). \tag{1}$$

As in [4], we consider the cost of a multicast message to be equal to that of a unicast message. In real systems, however, it may vary depending on the network architecture.

*3) Reduce Phase:* Finally, in the reduce phase, the vectors $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_N$ are computed. More specifically, server $S \in \mathcal{Q}$ uses the locally computed sets $\mathcal{Z}_1^{(S)}, \ldots, \mathcal{Z}_N^{(S)}$ and the received messages to compute the vectors in $\{\boldsymbol{y}_j : j \in \mathcal{W}_S\}$. The computational delay of the reduce phase is its average runtime per source row and output vector $\boldsymbol{y}$, i.e., $D_{\mathsf{reduce}} = \frac{1}{mN} f(\frac{\sigma_{\mathsf{reduce}}}{q}, q, q)$, where $\sigma_{\mathsf{reduce}}$ is given in Section IV-B.

**Definition 2.** *The overall computational delay, $D$, is the sum of the map and reduce phase delays, i.e., $D = D_{\mathsf{map}} + D_{\mathsf{reduce}}$.*

## III. Block-Diagonal Coding

We introduce a block-diagonal encoding matrix of the form

$$\boldsymbol{\Psi} = \begin{bmatrix} \boldsymbol{\psi}_1 & & \\ & \ddots & \\ & & \boldsymbol{\psi}_T \end{bmatrix},$$

where $\boldsymbol{\psi}_1, \ldots, \boldsymbol{\psi}_T$ are $\frac{r}{T} \times \frac{m}{T}$ encoding matrices of an $(\frac{r}{T}, \frac{m}{T})$ MDS code, for some integer $T$ that divides $m$ and $r$. Note that the encoding given by $\boldsymbol{\Psi}$ amounts to partitioning the rows of $\boldsymbol{A}$ into $T$ disjoint submatrices $\boldsymbol{A}_1, \ldots, \boldsymbol{A}_T$ and encoding each submatrix separately. We refer to an encoding $\boldsymbol{\Psi}$ with $T$ disjoint submatrices as a $T$-partitioned scheme, and to the submatrix of $\boldsymbol{C} = \boldsymbol{\Psi}\boldsymbol{A}$ corresponding to $\boldsymbol{\psi}_i$ as the $i$-th partition. We remark that all submatrices can be encoded using the same encoding matrix, i.e., $\boldsymbol{\psi}_i = \boldsymbol{\psi}$, $i = 1, \ldots, T$, reducing the storage requirements, and encoding/decoding can be performed in parallel if many servers are available. We further remark that the case $\boldsymbol{\Psi} = \boldsymbol{\psi}$ (i.e., the number of partitions is $T = 1$) corresponds to the scheme in [4], which we will sometimes refer to as the *unpartitioned* scheme.

### A. Assignment of Coded Rows to Batches

For a block-diagonal encoding matrix $\boldsymbol{\Psi}$, we denote by $\boldsymbol{c}_i^{(t)}$, $t = 1, \ldots, T$ and $i = 1, \ldots, r/T$, the $i$-th coded row of $\boldsymbol{C}$ within partition $t$. For example, $\boldsymbol{c}_1^{(2)}$ denotes the first coded row of the second partition. As described in Section II, the coded rows are divided into $\binom{K}{\mu q}$ disjoint batches. To formally describe the assignment of coded rows to batches we use a $\binom{K}{\mu q} \times T$ integer matrix $\boldsymbol{P} = [p_{i,j}]$, where $p_{i,j}$ describes the number of rows from partition $j$ that are stored in batch $i$. Note that, due to the MDS property, any set of $m/T$ rows of a partition is sufficient to decode the partition. Thus, without loss of generality, we consider a *sequential* assignment of rows of a partition into batches. For example, for the assignment $\boldsymbol{P}$ in Example 1 (see (2)), rows $\boldsymbol{c}_1^{(1)}$ and $\boldsymbol{c}_2^{(1)}$ are assigned to batch 1, $\boldsymbol{c}_3^{(1)}$ and $\boldsymbol{c}_4^{(1)}$ are assigned to batch 2, and so on. The rows of $\boldsymbol{P}$ are labeled by the subset of servers the corresponding batch is stored at, and the columns are labeled by its partition index. We refer to the pair $(\boldsymbol{\Psi}, \boldsymbol{P})$ as the *storage design*. The assignment matrix $\boldsymbol{P}$ must satisfy the following conditions.

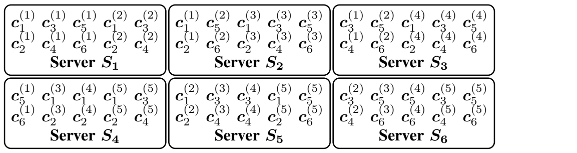
$\boldsymbol{c}_1^{(1)}$ $\boldsymbol{c}_3^{(1)}$ $\boldsymbol{c}_5^{(1)}$ $\boldsymbol{c}_1^{(2)}$ $\boldsymbol{c}_3^{(2)}$
$\boldsymbol{c}_2^{(1)}$ $\boldsymbol{c}_4^{(1)}$ $\boldsymbol{c}_6^{(1)}$ $\boldsymbol{c}_2^{(2)}$ $\boldsymbol{c}_4^{(2)}$
Server $S_1$
$\boldsymbol{c}_1^{(1)}$ $\boldsymbol{c}_5^{(2)}$ $\boldsymbol{c}_1^{(3)}$ $\boldsymbol{c}_3^{(3)}$ $\boldsymbol{c}_5^{(3)}$
$\boldsymbol{c}_2^{(1)}$ $\boldsymbol{c}_6^{(2)}$ $\boldsymbol{c}_2^{(3)}$ $\boldsymbol{c}_4^{(3)}$ $\boldsymbol{c}_6^{(3)}$
Server $S_2$
$\boldsymbol{c}_3^{(1)}$ $\boldsymbol{c}_5^{(2)}$ $\boldsymbol{c}_1^{(4)}$ $\boldsymbol{c}_3^{(4)}$ $\boldsymbol{c}_5^{(4)}$
$\boldsymbol{c}_4^{(1)}$ $\boldsymbol{c}_6^{(2)}$ $\boldsymbol{c}_2^{(4)}$ $\boldsymbol{c}_4^{(4)}$ $\boldsymbol{c}_6^{(4)}$
Server $S_3$
$\boldsymbol{c}_5^{(1)}$ $\boldsymbol{c}_1^{(3)}$ $\boldsymbol{c}_1^{(4)}$ $\boldsymbol{c}_1^{(5)}$ $\boldsymbol{c}_3^{(5)}$
$\boldsymbol{c}_6^{(1)}$ $\boldsymbol{c}_2^{(3)}$ $\boldsymbol{c}_2^{(4)}$ $\boldsymbol{c}_2^{(5)}$ $\boldsymbol{c}_4^{(5)}$
Server $S_4$
$\boldsymbol{c}_1^{(2)}$ $\boldsymbol{c}_3^{(3)}$ $\boldsymbol{c}_3^{(4)}$ $\boldsymbol{c}_1^{(5)}$ $\boldsymbol{c}_5^{(5)}$
$\boldsymbol{c}_2^{(2)}$ $\boldsymbol{c}_4^{(3)}$ $\boldsymbol{c}_4^{(4)}$ $\boldsymbol{c}_2^{(5)}$ $\boldsymbol{c}_6^{(5)}$
Server $S_5$
$\boldsymbol{c}_3^{(2)}$ $\boldsymbol{c}_5^{(3)}$ $\boldsymbol{c}_5^{(4)}$ $\boldsymbol{c}_3^{(5)}$ $\boldsymbol{c}_5^{(5)}$
$\boldsymbol{c}_4^{(2)}$ $\boldsymbol{c}_6^{(3)}$ $\boldsymbol{c}_6^{(4)}$ $\boldsymbol{c}_4^{(5)}$ $\boldsymbol{c}_6^{(5)}$
Server $S_6$

Fig. 1. Storage design for $m = 20$, $N = 4$, $K = 6$, $q = 4$, $\mu = 1/2$, and $T = 5$.

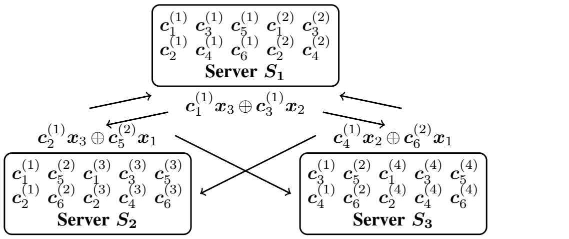
$\boldsymbol{c}_1^{(1)}$ $\boldsymbol{c}_3^{(1)}$ $\boldsymbol{c}_5^{(1)}$ $\boldsymbol{c}_1^{(2)}$ $\boldsymbol{c}_3^{(2)}$
$\boldsymbol{c}_2^{(1)}$ $\boldsymbol{c}_4^{(1)}$ $\boldsymbol{c}_6^{(1)}$ $\boldsymbol{c}_2^{(2)}$ $\boldsymbol{c}_4^{(2)}$
Server $S_1$
$\boldsymbol{c}_1^{(1)}\boldsymbol{x}_3 \oplus \boldsymbol{c}_3^{(1)}\boldsymbol{x}_2$
$\boldsymbol{c}_2^{(1)}\boldsymbol{x}_3 \oplus \boldsymbol{c}_5^{(2)}\boldsymbol{x}_1$
$\boldsymbol{c}_4^{(1)}\boldsymbol{x}_2 \oplus \boldsymbol{c}_6^{(2)}\boldsymbol{x}_1$
$\boldsymbol{c}_1^{(1)}$ $\boldsymbol{c}_5^{(2)}$ $\boldsymbol{c}_1^{(3)}$ $\boldsymbol{c}_3^{(3)}$ $\boldsymbol{c}_5^{(3)}$
$\boldsymbol{c}_2^{(1)}$ $\boldsymbol{c}_6^{(2)}$ $\boldsymbol{c}_2^{(3)}$ $\boldsymbol{c}_4^{(3)}$ $\boldsymbol{c}_6^{(3)}$
Server $S_2$
$\boldsymbol{c}_3^{(1)}$ $\boldsymbol{c}_5^{(2)}$ $\boldsymbol{c}_1^{(4)}$ $\boldsymbol{c}_3^{(4)}$ $\boldsymbol{c}_5^{(4)}$
$\boldsymbol{c}_4^{(1)}$ $\boldsymbol{c}_6^{(2)}$ $\boldsymbol{c}_2^{(4)}$ $\boldsymbol{c}_4^{(4)}$ $\boldsymbol{c}_6^{(4)}$
Server $S_3$

Fig. 2. Multicasting coded values between servers $S_1$, $S_2$, and $S_3$.

1) The entries of each row of $\boldsymbol{P}$ must sum to the batch size, i.e., $\sum_{j=1}^{T} p_{i,j} = r/\binom{K}{\mu q}$, $1 \leq i \leq \binom{K}{\mu q}$.
2) The entries of each column of $\boldsymbol{P}$ must sum to the number of rows per partition, i.e., $\sum_{i=1}^{\binom{K}{\mu q}} p_{i,j} = \frac{r}{T}$, $1 \leq j \leq T$.

**Example 1** ($m = 20$, $N = 4$, $K = 6$, $q = 4$, $\mu = 1/2$, $T = 5$)**.** *For these parameters, there are $r/T = 6$ coded rows per partition, of which $m/T = 4$ are sufficient for decoding, and $\binom{K}{\mu q} = 15$ batches, each containing $r/\binom{K}{\mu q} = 2$ coded rows. We construct the storage design shown in Fig. 1 with assignment matrix*

$$\boldsymbol{P} = \begin{array}{c} \\ (S_1, S_2) \\ (S_1, S_3) \\ (S_1, S_4) \\ (S_1, S_5) \\ \vdots \\ (S_4, S_6) \\ (S_5, S_6) \end{array} \begin{array}{c} \begin{matrix} 1 & 2 & 3 & 4 & 5 \end{matrix} \\ \begin{pmatrix} 2 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 \\ & & \vdots & & \\ 0 & 0 & 0 & 0 & 2 \\ 0 & 0 & 0 & 0 & 2 \end{pmatrix} \end{array}, \tag{2}$$

*where rows are labeled by the subset of servers the batch is stored at, and columns are labeled by the partition index. For this storage design, any $g = 4$ servers collectively store at least 4 coded rows from each partition. However, some servers store more rows than needed to decode some partitions, suggesting that this storage design is suboptimal.*

*Assume $\mathcal{G} = \{S_1, S_2, S_3, S_4\}$ is the set of $g = 4$ servers that finish their map computations first. Also, assign vector $\boldsymbol{y}_i$ to server $S_i$, $i = 1, 2, 3, 4$. We illustrate the coded shuffling scheme for $\mathcal{S} = \{S_1, S_2, S_3\}$ in Fig. 2. $S_1$ multicasts $\boldsymbol{c}_1^{(1)}\boldsymbol{x}_3 \oplus \boldsymbol{c}_3^{(1)}\boldsymbol{x}_2$ to $S_2$ and $S_3$. Since $S_2$ and $S_3$ can cancel $\boldsymbol{c}_1^{(1)}\boldsymbol{x}_3$ and $\boldsymbol{c}_3^{(1)}\boldsymbol{x}_2$, respectively, both servers receive one needed intermediate value. Similarly, $S_2$ multicasts $\boldsymbol{c}_2^{(1)}\boldsymbol{x}_3 \oplus \boldsymbol{c}_5^{(2)}\boldsymbol{x}_1$, while $S_3$ multicasts $\boldsymbol{c}_4^{(1)}\boldsymbol{x}_2 \oplus \boldsymbol{c}_6^{(2)}\boldsymbol{x}_1$. This process is repeated for $\mathcal{S} = \{S_2, S_3, S_4\}$, $\mathcal{S} = \{S_1, S_3, S_4\}$,* *and $\mathcal{S} = \{S_1, S_2, S_4\}$. After the shuffle phase, we have sent $12$ multicast messages and $30$ unicast messages, resulting in a communication load of $(12+30)/20/4 = 0.525$, a $50\%$ increase from the load of the unpartitioned scheme ($0.35$, given by (1)). In this case, $S_1$ received additional intermediate values from partition $2$, despite already storing enough, further indicating that the assignment in (2) is suboptimal.*

## IV. Performance of the Block-Diagonal Coding

In this section, we analyze the performance impact of partitioning. We have the following theorem.

**Theorem 1.** *For $T \leq r/\binom{K}{\mu q}$ there exists an assignment matrix $\boldsymbol{P}$ such that the communication load and computational delay of the map phase are equal to those of the unpartitioned scheme.*

### A. Communication Load

For the unpartitioned scheme of [4], $\mathcal{G} = \mathcal{Q}$, and the number of remaining values that need to be unicasted after the multicast phase is constant, regardless which subset $\mathcal{Q}$ of servers finish first their map computations. However, for the block-diagonal (partitioned) coding scheme, both $g$ and the number of remaining unicasts may vary.

For a given assignment $\boldsymbol{P}$ and a specific $\mathcal{Q}$, we denote by $U_{\mathcal{Q}}^{(S)}(\boldsymbol{P})$ the number of remaining values needed after the multicast phase by server $S \in \mathcal{Q}$, and by $U_{\mathcal{Q}}(\boldsymbol{P}) \triangleq \sum_{S\in\mathcal{Q}} U_{\mathcal{Q}}^{(S)}(\boldsymbol{P})$ the total number of remaining values needed by the servers in $\mathcal{Q}$. Let $\mathbb{Q}^q$ denote the superset of all sets $\mathcal{Q}$ and define $L_{\mathbb{Q}} \triangleq \frac{1}{mN}\frac{1}{|\mathbb{Q}^q|}\sum_{\mathcal{Q}\in\mathbb{Q}^q} U_{\mathcal{Q}}(\boldsymbol{P})$. Then, for a given storage design $(\boldsymbol{\Psi}, \boldsymbol{P})$, the communication load of the block-diagonal coding scheme is given by

$$L_{\text{BDC}}(\boldsymbol{\Psi},\boldsymbol{P}) = \min\left(\sum_{j=s_q}^{\mu q}\frac{\alpha_j}{j} + L_{\mathbb{Q}}, \sum_{j=s_q-1}^{\mu q}\frac{\alpha_j}{j} + L_{\mathbb{Q}}\right), \quad (3)$$

where $L_{\mathbb{Q}}$ depends on the shuffling scheme (see Section II-B2) and is different in the first and second term of the minimization in (3). To evaluate $U_{\mathcal{Q}}^{(S)}$, we count the total number of intermediate values that need to be unicasted to server $S$ until it holds $m/T$ intermediate values from each partition.

For a given $\boldsymbol{\Psi}$, the assignment of rows into batches can be formulated as an optimization problem, where one would like to minimize $L_{\text{BDC}}$ over all assignments $\boldsymbol{P}$. More precisely, the optimization problem is $\min_{\boldsymbol{P}\in\mathbb{P}} L_{\text{BDC}}(\boldsymbol{\Psi},\boldsymbol{P})$, where $\mathbb{P}$ is the set of all assignments $\boldsymbol{P}$, and where the dependence of $L_{\text{BDC}}$ on $\boldsymbol{P}$ is nonlinear. This is a computationally complex problem since both the complexity of evaluating the performance of a given assignment and the number of assignments scale exponentially in the problem size.

### B. Computational Delay

We consider the delay incurred by both the map and reduce phases (see Definition 2). Note that in [4] only $D_{\mathsf{map}}$ is considered, i.e., $D = D_{\mathsf{map}}$. However, one should not neglect the computational delay incurred by the reduce phase. Thus, one should consider $D = D_{\mathsf{map}} + D_{\mathsf{reduce}}$. The reduce phase consists of decoding the $N$ output vectors and hence the delay it incurs depends on the code and decoding algorithm. We assume that each partition is encoded using an RS code and is decoded using the Berlekamp-Massey algorithm. We measure the decoding complexity by its associated shifted-exponential parameter $\sigma$ (see Section II-A).

The number of field multiplications required to decode an $(r/T, m/T)$ RS code is $(r/T)^2\xi$ [7], where $\xi$ is the fraction of erased symbols. With $\xi$ upperbounded by $1-\frac{q}{K}$ (the map phase terminates when a fraction of at least $\frac{q}{K}$ symbols from each partition is available) the complexity of decoding the $T$ partitions for all $N$ output vectors is at most

$$\sigma_{\mathsf{reduce}} = \frac{r^2(1-\frac{q}{K})N}{T}. \quad (4)$$

The decoding complexity of the scheme in [4] is given by evaluating (4) for $T = 1$. By choosing $T$ close to $r$, we can thus significantly reduce the delay of the reduce phase.

## V. Assignment Solvers

We propose two solvers for the problem of assigning rows into batches: a heuristic solver that is fast even for large problem instances, and a hybrid solver combining the heuristic solver with a branch-and-bound solver. The branch-and-bound solver produces an optimal assignment but is significantly slower, hence it can be used as stand-alone only for small problem instances. We use a dynamic programming approach to speed up the branch-and-bound solver by caching $U_{\mathcal{Q}}^{(S)}$ for all $\mathcal{Q} \in \mathbb{Q}^q$, indexed by the batches each $U_{\mathcal{Q}}^{(S)}$ is computed from. This way we only need to update the affected $U_{\mathcal{Q}}^{(S)}$ when assigning a row to a batch. For all solvers, we first label the batches lexiographically and then optimize $L_{\text{BDC}}$ in (3). The solvers are available under the Apache 2.0 license [9].

### A. Heuristic Solver

The heuristic solver creates an assignment matrix $\boldsymbol{P}$ in two steps. We first set each entry of $\boldsymbol{P}$ to $\gamma \triangleq \left\lfloor r/\left(\binom{K}{\mu q}\cdot T\right)\right\rfloor$, thus assigning the first $\binom{K}{\mu q}\gamma$ rows of each partition to batches such that each batch is assigned $\gamma T$ rows. Let $d = r/\binom{K}{\mu q} - \gamma T$ be the number of rows that still need to be assigned to each batch. The $r/T - \binom{K}{\mu q}\gamma$ rows per partition not assigned yet are assigned in the second step as given in the algorithm below.

```
Input  : $\boldsymbol{P}$, $d$, $K$, $T$, and $\mu q$
for $0 \leq a < d\binom{K}{\mu q}$ do
    $i \leftarrow \lfloor a/d \rfloor + 1$
    $j \leftarrow (a \bmod T) + 1$
    $p_{i,j} \leftarrow p_{i,j} + 1$
end
return $\boldsymbol{P}$
```

### B. Branch-and-Bound Solver

The branch-and-bound solver finds an optimal solution by recursively branching at each batch for which there is more than one possible assignment and considering all options. For each branch, we lowerbound the value of the objective function of any assignment in that branch and only investigate branches with possibly better assignments.

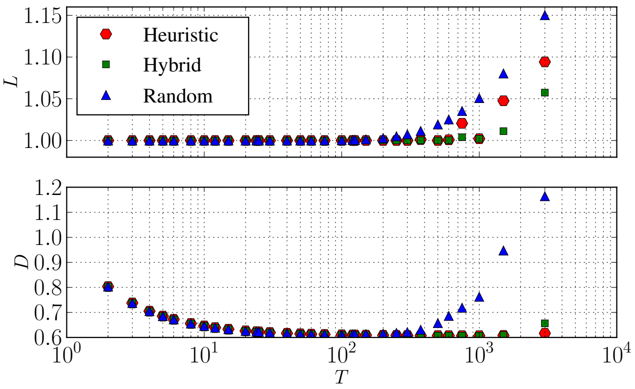

Fig. 3. The tradeoff between partitioning and performance for $m = 6000$, $n = 6000$, $K = 9$, $q = 6$, $N = 6$, and $\mu = 1/3$.

*1) Branch:* For the first row of $\boldsymbol{P}$ with remaining assignments, branch on every available assignment for that row.

*2) Bound:* We keep a record of all nonzero $U_{\mathcal{Q}}^{(S)}$ for all $\mathcal{Q}$ and $S$, and index them by the batches they are computed from. An assignment to a batch can at most reduce $L_{\text{BDC}}$ by $1/\left(mN\left|\mathbb{Q}^q\right|\right)$ for each nonzero $U_{\mathbb{Q}}^{(S)}$ indexed by that batch, and we lowerbound $L_{\text{BDC}}$ for a subtree by assuming that no $U_{\mathbb{Q}}^{(S)}$ will drop to zero for any subsequent assignment.

### C. Hybrid Solver

We first find a candidate solution using the heuristic solver and then iteratively improve it using the branch-and-bound solver. In particular, we decrement by 1 a random set of elements of $\boldsymbol{P}$ and then use the branch-and-bound solver to reassign the corresponding rows optimally. We repeat this process until the average improvement between iterations drops below some threshold.

## VI. Numerical Results

In Figs. 3 and 4, we plot the performance of the proposed block-diagonal coding scheme with assignment $\boldsymbol{P}$ given by the heuristic and the hybrid solvers. We normalize the performance by that of the unpartitioned scheme of [4]. We also give the average performance over 100 random assignments.

In Fig. 3, we plot the normalized communication load ($L$) and overall computational delay ($D$) as a function of the number of partitions $T$. The system parameters are $m = 6000$, $n = 6000$, $K = 9$, $q = 6$, $N = 6$, and $\mu = 1/3$. For up to $r/\binom{K}{\mu q} = 250$ partitions, the proposed scheme does not incur any loss in $D_{\mathsf{map}}$ and communication load with respect to the unpartitioned scheme. However, the proposed scheme yields a significantly lower $D$ compared to the scheme in [4] (about 40% speedup for $T > 50$). For heavy partitioning (around $T = 800$) a tradeoff between partitioning level, communication load, and map phase delay is observed. With 3000 partitions (the maximum possible), there is about a 10% increase in communication load. Note that the gain in overall computational delay saturates with the partitioning level, thus there is no reason to partition beyond a given level.

In Fig. 4, we plot the normalized performance for a constant $\mu q = 2$, $n = 10000$, $\mu m = 2000$, $m/T = 10$ rows per partition, and code rate $m/r = 2/3$ as a function of the number of servers, $K$. The results shown are averages over 1000 randomly generated realizations of $\mathcal{G}$ as it is computationally unfeasible to evaluate the performance exhaustively in this case. The heuristic solver outperforms the random assignments for small $K$, but as $K$ grows the performance of both solvers converge. The delay is an order of magnitude lower than that of the scheme in [4] for the largest system considered.

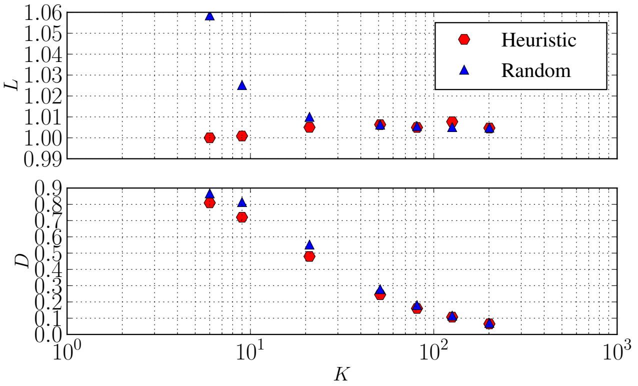

Fig. 4. Performance dependence on system size for $\mu q = 2$, $n = 10000$, $\mu m = 2000$, $m/T = 10$ rows per partition, and code rate $m/r = 2/3$.

## VII. Conclusion

We introduced a block-diagonal coding scheme for distributed matrix multiplication based on partitioning the matrix into smaller submatrices. Compared to earlier (MDS) schemes, the proposed scheme yields a significantly lower computational delay with no increase in communication load up to a level of partitioning. For instance, for a square matrix with 6000 rows and columns, the proposed scheme reduces the computational delay by about 40% when the number of partitions $T > 50$. We have also considered fountain codes, and will present the results in an extension of this paper.

## References

[1] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," in *Proc. 6th Symp. Operating Systems Design & Implementation*, San Francisco, CA, Dec. 2004, pp. 137–149.

[2] S. Li, M. A. Maddah-Ali, and A. S. Avestimehr, "Coded MapReduce," in *Proc. Annual Allerton Conf. Commun., Control, and Computing*, Monticello, IL, Sep./Oct. 2015, pp. 964–971.

[3] K. Lee, M. Lam, R. Pedarsani, D. Papailiopoulos, and K. Ramchandran, "Speeding up distributed machine learning using codes," in *Proc. IEEE Int. Symp. Inf. Theory*, Barcelona, Spain, Jul. 2016, pp. 1143–1147.

[4] S. Li, M. A. Maddah-Ali, and A. S. Avestimehr, "A unified coding framework for distributed computing with straggling servers," in *Proc. Workshop Network Coding and Appl.*, Washington, DC, Dec. 2016.

[5] H. Ishii and R. Tempo, "The PageRank problem, multiagent consensus, and web aggregation: A systems and control viewpoint," *IEEE Control Systems*, vol. 34, no. 3, pp. 34–53, Jun. 2014.

[6] G. Garrammone, "On decoding complexity of Reed-Solomon codes on the packet erasure channel," *IEEE Commun. Lett.*, vol. 17, no. 4, pp. 773–776, Apr. 2013.

[7] Z. Li, J. Higgins, and M. Clement, "Performance of finite field arithmetic in an elliptic curve cryptosystem," in *Proc. Int. Symp. Model. Anal. Simul. Comput. Telecommun. Syst.*, Cincinnati, OH, Aug. 2001, pp. 249–256.

[8] B. C. Arnold, N. Balakrishnan, and H. N. Nagaraja, *A First Course in Order Statistics*, 2nd ed. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2008.

[9] A. Severinson, "Coded Computing Tools in Python," Aug. 2017. [Online]. Available: https://doi.org/10.5281/zenodo.844866