A Droplet Approach Based on Raptor Codes for Distributed Computing with Straggling Servers

A Droplet Approach Based on Raptor Codes for Distributed Computing With Straggling Servers

Albin Severinson[†], Alexandre Graell i Amat[‡], Eirik Rosnes[†], Francisco Lázaro[§], and Gianluigi Liva[§]
[†]Simula UiB, Bergen, Norway
[‡]Department of Electrical Engineering, Chalmers University of Technology, Gothenburg, Sweden
[§]Institute of Communications and Navigation of DLR (German Aerospace Center), Munich, Germany

***Abstract*—We propose a coded distributed computing scheme based on Raptor codes to address the straggler problem. In particular, we consider a scheme where each server computes intermediate values, referred to as droplets, that are either stored locally or sent over the network. Once enough droplets are collected, the computation can be completed. Compared to previous schemes in the literature, our proposed scheme achieves lower computational delay when the decoding time is taken into account.**

I. Introduction

Modern computing systems often consist of several thousands of servers working in a highly coordinated manner [1]. These systems, referred to as warehouse-scale computers (WSCs) [2], differ from traditional datacenters in that servers rarely have fixed roles. Instead, a cluster manager dynamically assigns storage and computing tasks to servers [1]. This approach offers a high level of flexibility but also poses significant challenges. For example, so-called *straggling servers*, i.e., servers that experience transient delays, are a major issue in WSCs and may significantly slow down the overall computation [3].

Recently, an approach based on maximum distance separable (MDS) codes was proposed to alleviate the straggler problem for linear computations (e.g., multiplying a matrix with a vector) [4], [5]. In particular, redundancy is added to the computation in such a way that straggling servers can be treated as erasures when decoding the final output. Any partially computed results by the straggling servers are discarded. In [4], a single master node is responsible for decoding the final output. A more general framework was proposed in [5], where the work of decoding is distributed over the servers. Somewhat surprisingly, most previous works neglect the decoding complexity of the underlying code, which may have a significant impact on the overall computational delay [6], [7]. For the matrix multiplication problem, a coded scheme consisting of partitioning the source matrix and encoding each partition separately using shorter MDS codes was proposed in [6], [7] and shown to significantly reduce the overall computational delay compared to using a single MDS code when the decoding complexity is taken into account. Furthermore, it was shown in [7] that Luby Transform (LT) codes [8] may reduce the delay further in some cases.

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

Using LT codes for distributed computing has also been studied in [9], [10], where, assuming that a single master node is responsible for decoding the output, it was shown that these codes may bring some advantages. In [9], the problem of multiplying a matrix by a vector in an internet-of-things setting was considered. Specifically, a scheme based on LT codes where a device may dynamically assign computing tasks to its neighboring devices was proposed. It was shown that this scheme achieves low delay and high resource utilization even when the available computing resources vary over time. The scheme proposed in [10] extends the scheme in [4] by introducing LT codes and utilizing partial computations. The authors give bounds on the overall delay in this setting.

In this paper, we propose a coded computing scheme based on Raptor codes [11] for the problem of multiplying a matrix by a set of vectors. In particular, we consider standardized Raptor10 (R10) codes [12] as the underlying code. Similar to [10], the proposed scheme exploits partial computations, i.e., servers compute intermediate values, referred here to as droplets, that are either stored locally or transferred over the network. The computation can be completed once enough droplets have been collected. Unlike in [4], [5], [9], [10], we take the decoding time into account since it may contribute significantly to the overall computational delay [7]. Furthermore, the work of decoding the output is distributed over the servers in a similar fashion to the scheme in [5]. We show that this significantly reduces the overall computational delay compared to the scheme in [10] when the number of servers is large, and also outperforms other schemes in the literature. Interestingly, the proposed scheme based on R10 codes achieves an overall computational delay close to that of a scheme using an *ideal* rateless code with zero overhead and incurring no decoding delay. Furthermore, we provide an analytical approximation of the expected overall computational delay of the proposed scheme when the droplets are computed in an optimal order. We then give a heuristic for choosing the order in which each server computes values and show numerically that it achieves almost identical performance to optimal ordering. We also present an optimization problem for finding the optimal number of servers over which the decoding of the final output should be distributed.

II. System Model and Preliminaries

We consider the distributed matrix multiplication problem. Specifically, given an $m \times n$ matrix $\boldsymbol{A} \in \mathbb{F}_{2^u}^{m \times n}$ and N vectors $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N \in \mathbb{F}_{2^u}^{n}$, where $\mathbb{F}_{2^u}$ is an extension field of characteristic 2, 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, $S_1, \ldots, S_K$. More precisely, $\boldsymbol{A}$ is split into m/l disjoint submatrices, each consisting of l rows. The submatrices are then encoded using an $(r/l, m/l)$ linear code, resulting in r/l encoded submatrices, denoted by $\boldsymbol{C}_1, \ldots, \boldsymbol{C}_{r/l}$. We refer to l as the *droplet size*. Each of the r/l coded submatrices is stored at exactly one server such that each server stores ηm coded matrix rows, for some $\frac{1}{K} \leq \eta \leq 1$. Note that, overall, the K servers store a total of $r = \eta m K$ coded rows. We assume that η is selected such that ηm is a multiple of l. Finally, we denote by $\mathcal{C}_k$ the set of indices of the submatrices stored by server S_k.

A. Probabilistic Runtime Model

We assume that each server $S_1, \ldots, S_K$ becomes available and starts working on its assigned tasks after a random amount of time, which is captured by the random variables $H_1, \ldots, H_K$, respectively. We assume that $H_1, \ldots, H_K$ are independent and identically distributed (i.i.d) random variables with exponential probability density function

$$f_H(h) = \begin{cases} \frac{1}{\beta} \mathrm{e}^{-\frac{h}{\beta}} & h \geq 0 \\ 0 & h < 0 \end{cases},$$

where β is used to scale the tail of the distribution. The tail accounts for transient disturbances that are at the root of the straggler problem. We refer to β as the straggling parameter. As in [10], we assume that once a server becomes available it carries out each of its assigned tasks in a deterministic amount of time, denoted by σ. Let σ_{A} and σ_{M} be the time required to compute one addition and one multiplication, respectively, over $\mathbb{F}_{2^u}$. The parameter σ is then given by $\sigma = n_{\mathsf{A}}\sigma_{\mathsf{A}} + n_{\mathsf{M}}\sigma_{\mathsf{M}}$, where n_{A} and n_{M} are the required number of additions and multiplications, respectively, to complete each task. As in [12], we assume that σ_{A} is $\mathcal{O}(\frac{u}{64})$ and σ_{M} is $\mathcal{O}(u \log_2 u)$. Furthermore, we assume that the hidden coefficients are comparable and will thus not consider them.

We denote by $H_{(i)}$, $i = 1, \ldots, K$, the i-th order statistic, i.e., the i-th smallest variable of $H_1, \ldots, H_K$. $H_{(i)}$ is a gamma-distributed random variable with cumulative probability distribution function

$$F_{H_{(i)}}(h_{(i)}) \triangleq \Pr(H_{(i)} \leq h_{(i)}) = \begin{cases} \frac{\gamma(b, a h_{(i)})}{\Gamma(b)} & h_{(i)} \geq 0 \\ 0 & h_{(i)} < 0 \end{cases},$$

where Γ denotes the gamma function and γ the lower incomplete gamma function. The inverse scale factor a and shape parameter b of the gamma distribution are computed from its mean and variance as in [7]. The expectation of $H_{(i)}$, i.e., the expected delay until a total of i servers become available, is [13]

$$\mu(K, i) \triangleq \mathbb{E}\left[H_{(i)}\right] = \sum_{j=K-i+1}^{K} \frac{\beta}{j}.$$

Finally, we denote by h_i and $h_{(i)}$ the realizations of H_i and $H_{(i)}$, $i = 1, \ldots, K$, respectively.

B. Distributed Computing Model

We consider the coded computing framework introduced in [5], which extends the MapReduce framework [3]. The overall computation proceeds in two phases, the *map-shuffle* phase and the *reduce* phase, which are augmented to make use of the coded scheme proposed in [10] to alleviate the straggler problem. We assume that the input vectors $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$ are known to all servers at the start of the computation.

1) Map-Shuffle Phase: The servers compute coded intermediate values (droplets) which are later used to obtain the vectors $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_N$. Each droplet is the product between a submatrix stored by the server and an input vector $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$. The responsibility for decoding each of the vectors $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_N$ is assigned to one of the K servers. The computed droplets are then transferred over the network to the server responsible for decoding the corresponding output vector. We assume that the channel is error-free and that all transfers are unicast. The map-shuffle phase ends when all output vectors $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_N$ can be decoded with high probability (see Section III-B). At this point the computation enters the reduce phase. We denote the delay of the map-shuffle phase by D_{map} and its expectation by $\bar{D}_{\mathsf{map}}$.

2) Reduce Phase: The vectors $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_N$ are computed from the intermediate values. More specifically, each server uses the droplets computed locally or received over the network to decode the output vectors it has been assigned. Denote by σ_{reduce} the time required for one server to decode one output vector. The computational delay of the reduce phase, denoted by $\mathsf{D}_{\mathsf{reduce}}$, is deterministic and is given by $\mathsf{D}_{\mathsf{reduce}} = \frac{N}{q}\sigma_{\mathsf{reduce}}$, where q denotes the number of servers used in the reduce phase.

Definition 1. *The overall computational delay, D, is the sum of the map-shuffle and reduce phase delays, i.e.,*

$$D = D_{\mathsf{map}} + \mathsf{D}_{\mathsf{reduce}} \quad \textit{and} \quad \bar{D} \triangleq \mathbb{E}[D] = \bar{D}_{\mathsf{map}} + \mathsf{D}_{\mathsf{reduce}}.$$

C. Raptor Codes

Raptor codes [11] are built from the serial concatenation of an outer linear block code with an inner LT code. Raptor codes not only outperform LT codes in terms of probability of decoding failure but also exhibit a lower encoding and decoding complexity. Here we consider R10 codes, which are binary codes whose outer code is obtained as the serial concatenation of a low-density parity-check code with a high-density parity-check (HDPC) code [14]. R10 codes are tailored to an efficient maximum likelihood decoding algorithm known as *inactivation decoding* [11]. In particular, we consider R10 codes in their nonsystematic form.

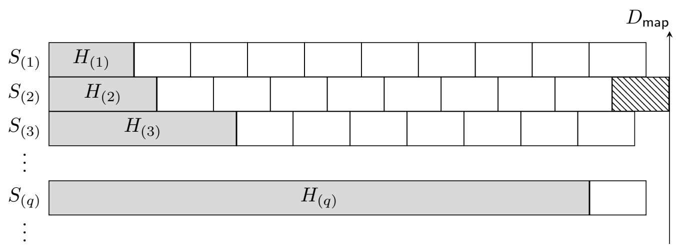

Fig. 1. Map-shuffle phase computation. Each server $S_{(k)}$, $k = 1, \dots, K$, computes droplets, illustrated by white squares, after an initial time $H_{(k)}$. The map-shuffle phase ends once enough droplets are collected and server $S_{(q)}$ has become available. It incurs a delay D_{map}. We depict the final droplet that is computed with a hash pattern.

III. Proposed Coded Computing Scheme

In this section, we introduce the proposed coded computing scheme. The main idea is that each server computes multiple intermediate values. More specifically, each server S_k, $k = 1, \dots, K$, computes droplets $\boldsymbol{z}_j^{(i)} = \boldsymbol{C}_i \boldsymbol{x}_j$ by multiplying the coded submatrices $\boldsymbol{C}_i$, $i \in \mathcal{C}_k$, it stores locally with the N input vectors $\boldsymbol{x}_1, \dots, \boldsymbol{x}_N$. The indices $i \in \mathcal{C}_k$ and $j \in \{1, \dots, N\}$ should be carefully chosen to minimize the computational delay. We consider this in Section III-A. The time required for a server to compute a droplet, denoted by σ_{d}, is $\sigma_{\mathsf{d}} = l\left((n-1)\sigma_{\mathsf{A}} + n\sigma_{\mathsf{M}}\right)$ since it requires computing l inner products, each requiring $n-1$ additions and n multiplications.

Denote by $S_{(1)}$ the first server to become available, and similarly denote by $S_{(k)}$, $k = 1, \dots, K$, the k-th server to become available. We assume that server $S_{(k)}$ computes droplets at a constant rate after a delay $H_{(k)}$. For example, server $S_{(k)}$ computes p droplets after a total delay of $H_{(k)} + p\sigma_{\mathsf{d}}$. This process is depicted in Fig. 1. We evenly and randomly split the indices of the N output vectors $\boldsymbol{y}_1, \dots, \boldsymbol{y}_N$ into $q \leq K$ disjoint sets $\mathcal{W}_1, \dots, \mathcal{W}_q$. Each of the q fastest servers $S_{(k)}$, $k = 1, \dots, q$, is responsible for decoding the N/q output vectors with indices in $\mathcal{W}_k$. Furthermore, we denote by $\tilde{\mathcal{W}}_k$ the set containing the indices of the vectors that server $S_{(k)}$ is not yet able to decode due to an insufficient number of droplets. At the start of the map-shuffle phase, $\tilde{\mathcal{W}}_k = \mathcal{W}_k$. The map-shuffle phase ends when servers $S_{(1)}, \dots, S_{(q)}$ have collected enough droplets to decode the output vectors they are responsible for, i.e., when $\left|\tilde{\mathcal{W}}_k\right| = 0$, $k = 1, \dots, q$. At this point servers $S_{(1)}, \dots, S_{(q)}$ simultaneously enter the reduce phase. The remaining $K - q$ servers are unused for the rest of the computation. A strategy for choosing q to minimize the expected computational delay, $\bar{D}$, is discussed in Section IV-A.

A. Droplet Order

For each droplet $\boldsymbol{z}_j^{(i)}$ computed by server S_k in the map-shuffle phase, the server has to choose the indices $i \in \mathcal{C}_k$ and $j \in \{1, \dots, N\}$ the droplet is computed from. Furthermore, the choice of i and j may have a large impact on the computational delay. In particular, if j is chosen such that it is not needed to decode any of the output vectors, i.e., j is not in any of the sets $\tilde{\mathcal{W}}_k$, $k = 1, \dots, q$, the resulting droplet is effectively wasted. Hence, i and j should be carefully chosen. We consider two scenarios. In the first scenario, i and j are chosen optimally, i.e., all servers have perfect knowledge of $\tilde{\mathcal{W}}_1, \dots, \tilde{\mathcal{W}}_q$. This gives a lower bound on the achievable computational delay. In a second, more practical scenario, i and j are chosen in a round-robin fashion. Specifically, for each server S_k we generate a number j from $\{1, \dots, N\}$ uniformly at random. Next, for each droplet $\boldsymbol{z}_j^{(i)}$ computed by server S_k we let $j = j + 1 \bmod N$. We remark that the optimal order requires each server to have global knowledge of all previously computed droplets over all servers, whereas the round-robin strategy only requires each server to have knowledge of the droplets it has computed locally. In Section V, we show numerically that the round-robin strategy achieves almost identical performance to the optimal strategy, the latter being infeasible in practice. In both cases we assume that the same pair of indices i, j is never chosen twice. Since each submatrix $\boldsymbol{C}_i$ is stored at exactly one server, this does not require any additional synchronization between servers. A server that has exhausted all possible combinations of i and j halts and performs no further computations in the map-shuffle phase.

B. Code Design

The decoding complexity and failure probability of Raptor codes depend on the number of droplets available to the decoder, $\frac{m}{l}(1+\epsilon)$, for some $\epsilon \geq 0$. We refer to ϵ as the overhead. Furthermore, we denote by $P_{\mathsf{f}}(\epsilon)$ the decoding failure probability when the overhead is ϵ. In general, increasing ϵ reduces the probability of decoding failure $P_{\mathsf{f}}(\epsilon)$ and the decoding complexity, leading to a lower decoding time σ_{reduce}. For example, the decoding failure probability for R10 codes roughly halves with every additional droplet available when the number of source symbols is close to 1000 [11]. However, a larger overhead ϵ also increases the computational delay due to computing the required droplets in the map-shuffle phase. We thus need to balance the computational delay of the reduce phase against that of the map-shuffle phase to achieve a low overall computational delay.

We denote by ϵ_{min} the minimum overhead before decoding is attempted. R10 codes are fully specified, hence the only free parameter is ϵ_{min}. In [11], it is observed that the decoding complexity of Raptor codes drops sharply when the number of droplets available to the decoder is increased to be slightly larger than the number of HDPC symbols. Hence, we choose the minimum overhead ϵ_{min} such that the number of droplets available to the decoder is close to the number of source droplets m/l plus twice the number of HDPC symbols. For comparison purposes, in Section V we also consider LT codes with a robust Soliton distribution [8], whose parameters are optimized as described in [7]. In particular, we choose a minimum overhead ϵ_{min} and a target failure probability $P_{\mathsf{f,target}}$ and optimize the parameters of the distribution to minimize the decoding complexity under the constraint $P_{\mathsf{f,target}} \approx P_{\mathsf{f}}(\epsilon_{\mathsf{min}})$. Note that the overhead ϵ required for decoding may be larger than ϵ_{min}. We take this into account by simulating the overhead needed given that decoding failed at an overhead of ϵ_{min}.

IV. Computational Delay Analysis

In this section, we analyze the computational delay of the proposed coded computing scheme and provide an approximation of $\bar{D}_{\mathsf{map}}$. Let V_p be the random variable associated with the time until p droplets are computed over K servers, where we assume that p is chosen such that decoding succeeds with high probability, and $\bar{V}_p$ its expectation, $\bar{V}_p \triangleq \mathbb{E}[V_p]$. Then, $D_{\mathsf{map}} = \max(V_p, H_{(q)})$. For the analysis, we assume that each server is always able to compute droplets needed by some server until the end of the map-shuffle phase. This assumption is valid if the code rate m/r is low enough. Furthermore, we assume that the droplet order is optimal (see Section III-A). Finally, we explain how to choose the number of servers q to split the output vectors over to minimize the expected computational delay.

Denote by P_t the number of droplets computed over K servers at time t.

Proposition 1. *The expectation of P_t is*

$$\bar{P}_t \triangleq \mathbb{E}\left[P_t\right] = K \int_0^t \left\lfloor \frac{t-h}{\sigma_{\mathsf{d}}} \right\rfloor \frac{1}{\beta} \mathrm{e}^{-\frac{h}{\beta}} \, \mathrm{d}h. \qquad (1)$$

Using the fact that $x - 1 \leq \lfloor x \rfloor \leq x$ in (1) and computing the resulting integrals, $\bar{P}_t$ can be lower and upperbounded as

$$K\left(\frac{(\beta+\sigma_{\mathsf{d}})\mathrm{e}^{-\frac{t}{\beta}}}{\sigma_{\mathsf{d}}} + \frac{t}{\sigma_{\mathsf{d}}} - \frac{\beta}{\sigma_{\mathsf{d}}} - 1\right) \leq \bar{P}_t \leq K\left(\frac{\beta \mathrm{e}^{-\frac{t}{\beta}}}{\sigma_{\mathsf{d}}} + \frac{t}{\sigma_{\mathsf{d}}} - \frac{\beta}{\sigma_{\mathsf{d}}}\right). \qquad (2)$$

Let $\sigma_{\bar{P}}$ denote the time at which an average number of $\bar{P}$ droplets have been computed over K servers. By inverting the upper and lower bounds on $\bar{P}_t$ in (2), $\sigma_{\bar{P}}$ can be bounded as

$$\sigma_{\bar{P}}^{\mathsf{L}} \triangleq \beta + \frac{\bar{P}\sigma_{\mathsf{d}}}{K} + \beta W_0\left(-\mathrm{e}^{-\frac{\bar{P}\sigma_{\mathsf{d}}}{K\beta}-1}\right) \leq \sigma_{\bar{P}} \leq \beta+\sigma_{\mathsf{d}}+\frac{\bar{P}\sigma_{\mathsf{d}}}{K}+\beta W_0\left(-\frac{\mathrm{e}^{-\frac{K(\beta+\sigma_{\mathsf{d}})+\bar{P}\sigma_{\mathsf{d}}}{K\beta}}(\beta+\sigma_{\mathsf{d}})}{\beta}\right) \triangleq \sigma_{\bar{P}}^{\mathsf{U}},$$

where $W_0(\cdot)$ is the principal branch of the Lambert W function, i.e., $W_0(x)$ is the solution of $x = z\mathrm{e}^z$.

Now, let G_t be the random variable associated with the number of servers that are available at time t. We provide the following heuristic approximation of $\bar{D}_{\mathsf{map}}$,

$$\bar{D}_{\mathsf{map}} \approx \bar{V}_p + \sum_{j=1}^{q-1} \Pr(G_t = j)\mu(K-j, q-j), \qquad (3)$$

where the summation accounts for the delay due to waiting for server $S_{(q)}$. We have numerically verified that the approximation holds. Furthermore, we have observed that $\bar{V}_p \approx \sigma_p$ and $\sigma_p \approx \frac{1}{2}\left(\sigma_p^{\mathsf{L}} + \sigma_p^{\mathsf{U}}\right)$. Finally, assuming that decoding is possible with p droplets, the expected overall computational delay is

$$\bar{D} \approx \frac{N}{q}\sigma_{\mathsf{reduce}} + \bar{V}_p + \sum_{j=1}^{q-1} \Pr(G_t = j)\mu(K-j, q-j). \qquad (4)$$

A. Straggler Mitigation

The map-shuffle phase ends when all output vectors can be decoded and when the servers $S_{(1)}, \ldots, S_{(q)}$ are available, i.e., $D_{\mathsf{map}} = \max(V_p, H_{(q)})$. Since $\Pr(H_{(q)} > V_p)$ is always nonzero, choosing a small q lowers the expected delay of the map phase. On the other hand, choosing a large q reduces the delay of the reduce phase $\mathsf{D}_{\mathsf{reduce}} = \frac{N}{q}\sigma_{\mathsf{reduce}}$, as the decoding is distributed over more servers. Thus, we need to balance the delay of the map-shuffle and reduce phases by choosing q carefully. In particular, we optimize the value of q to minimize the overall computational delay in (4), where we use the approximation $\bar{V}_p \approx \sigma_p \approx \frac{1}{2}\left(\sigma_p^{\mathsf{L}} + \sigma_p^{\mathsf{U}}\right)$. We remark that (4) as a function of q is convex as it is the sum of the approximation of $\bar{D}_{\mathsf{map}}$ in (3) and $\mathsf{D}_{\mathsf{reduce}}$, which are strictly increasing and decreasing, respectively, in q for $\sigma_{\mathsf{reduce}} > 0$. For $\sigma_{\mathsf{reduce}} = 0$, (4) is minimized for $q = 1$.

V. Numerical Results

In Fig. 2, we give the expected computational delay of the proposed scheme, normalized by that of the uncoded scheme, as a function of the system size. In particular, we fix the code rate to $m/r = 1/3$ and the problem size divided by the number of servers to $mnN/K = 10^7$ ($\pm 10\%$ to find valid parameters) and scale the system size with K. Motivated by machine learning applications, where the number of rows and columns often represent the number of samples and features, respectively, we set $m = 1000n$. We also set $N = 10K$. Since R10 codes are optimized for code lengths close to 1024 [14], we choose the droplet size l such that $900 < m/l < 1100$ (the interval is required to find valid parameters). The overhead is 2% and 30% for R10 and LT codes, respectively. Finally, the straggling parameter β is equal to the total time required to compute the multiplications $\boldsymbol{A}\boldsymbol{x}_1, \ldots, \boldsymbol{A}\boldsymbol{x}_N$ divided by the number of servers, i.e., $\beta = \sigma_{\mathsf{K}} = (m(n-1)\sigma_{\mathsf{A}} + mn\sigma_{\mathsf{M}})N/K$.

In the figure, we plot the overall computational delay given by (4) using the approximation $\bar{V}_p \approx \sigma_p \approx \frac{1}{2}\left(\sigma_p^{\mathsf{L}} + \sigma_p^{\mathsf{U}}\right)$ for the proposed scheme with an underlying R10 code (blue line with circle markers) and LT code (magenta line with diamond markers), and for the scheme assuming an ideal rateless code (black solid line). We also show simulated performance for the R10-based scheme with optimal droplet ordering and with a round-robin (rr) ordering. We observe that the round-robin strategy achieves a computational delay within 1% of that of the optimal strategy. Furthermore, (4) accurately predicts the overall computational delay with an error of at most about 1% compared to both the optimal and the round-robin ordering. The proposed scheme with R10 codes achieves a significantly lower delay than the scheme with LT codes. Interestingly, the delay for the scheme based on R10 codes is very close (at most 3.7% higher) to that of an ideal rateless code.

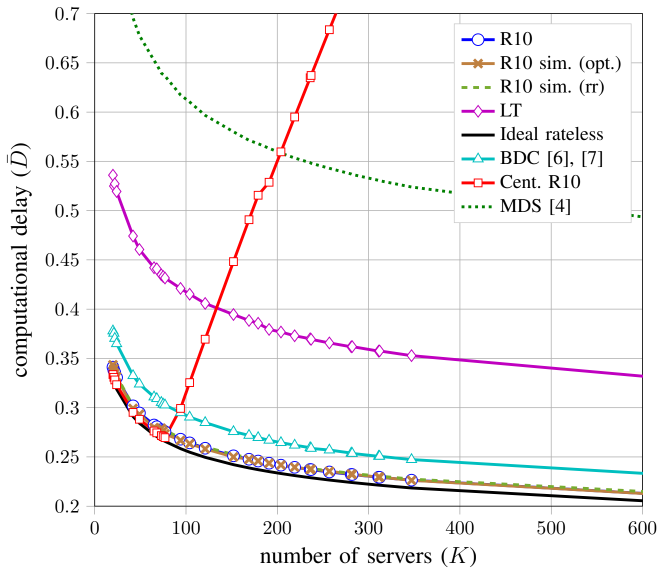

Fig. 2. Performance dependence on system size for $mnN/K \approx 10^7$, $n = m/1000$, $N = 10K$, $m/r = 1/3$, $m/l \approx 1024$, and $\beta = \sigma_{\mathsf{K}}$.

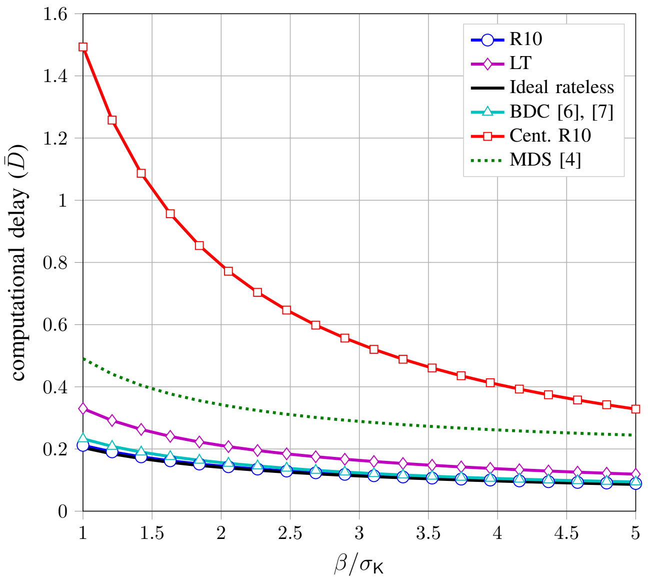

Fig. 3. Performance dependence on the straggling parameter β for $K = 625$, $m = 33333$, $n = 33$, $N = 6250$, $l = 32$, and $m/r = 1/3$.

For comparison purposes, we also plot in the figure the delay of the block-diagonal coding (BDC) scheme in [6], [7], the MDS coding scheme proposed in [4] that does not utilize partial computations, and the scheme proposed in [10] (augmented with R10 codes). We refer to it as the centralized R10 (cent. R10) scheme, since a central master node is responsible for decoding all output vectors. For small K, the delay is limited by the time needed to compute droplets. However, for $K \gtrsim 90$ the master node of the centralized scheme can no longer decode the output vectors quickly enough, causing a high overall computational delay. Thus, for $K \gtrsim 90$ the scheme in [10] (now with R10 codes), incurs a delay significantly higher than that of the proposed scheme. The proposed scheme also yields a significantly lower computational delay than that of the scheme in [4]. Finally, the delay of the BDC scheme in [6], [7] is about 10% higher compared to the proposed scheme based on R10 codes.

In Fig. 3, we give the expected computational delay as a function of the straggling parameter β for $K = 625$, $m = 33333$, $n = 33$, $N = 6250$, $l = 32$, and $m/r = 1/3$. Since l is not a divisor of m, $\boldsymbol{A}$ is zero-padded with 11 all-zero rows. The performance of the centralized scheme approaches that of our scheme as β grows since the average rate at which droplets are computed decreases with β. The scheme based on R10 codes operates close to an ideal rateless code for all values of β considered.

VI. Conclusion

We introduced a coded computing scheme based on Raptor codes for distributed matrix multiplication where each server computes several intermediate values and where the work of decoding the output is distributed among servers. Compared to previous schemes, the proposed scheme yields significantly lower computational delay when the number of servers is large. For instance, the delay is less than half when the number of servers is 200. Furthermore, the performance of the scheme based on R10 codes is close to that of an ideal rateless code.

References

[1] A. Verma, L. Pedrosa, M. R. Korupolu, D. Oppenheimer, E. Tune, and J. Wilkes, “Large-scale cluster management at Google with Borg,” in *Proc. European Conf. Computer Systems*, Bordeaux, France, Apr. 2015.

[2] L. A. Barroso and U. Hölzle, *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines*. Morgan & Claypool Publishers, 2009.

[3] J. Dean and S. Ghemawat, “MapReduce: Simplified data processing on large clusters,” in *Proc. Conf. Symp. Operating Systems Design & Implementation*, San Francisco, CA, Dec. 2004, p. 10.

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

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

[6] A. Severinson, A. Graell i Amat, and E. Rosnes, “Block-diagonal coding for distributed computing with straggling servers,” in *Proc. IEEE Inf. Theory Work.*, Kaohsiung, Taiwan, Nov. 2017, pp. 464–468.

[7] ——, “Block-diagonal and LT codes for distributed computing with straggling servers,” Dec. 2017. [Online]. Available: https://arxiv.org/abs/1712.08230v2

[8] M. Luby, “LT codes,” in *Proc. IEEE Symp. Foundations Computer Science*, Vancouver, BC, Canada, Nov. 2002, pp. 271–280.

[9] Y. Keshtkarjahromi, Y. Xing, and H. Seferoglu, “Dynamic heterogeneity-aware coded cooperative computation at the edge,” Jan. 2018. [Online]. Available: https://arxiv.org/abs/1801.04357v2

[10] A. Mallick, M. Chaudhari, and G. Joshi, “Rateless codes for near-perfect load balancing in distributed matrix-vector multiplication,” Apr. 2018. [Online]. Available: https://arxiv.org/abs/1804.10331v2

[11] A. Shokrollahi and M. Luby, “Raptor codes,” *Foundations and Trends in Commun. and Inf. Theory*, vol. 6, no. 34, pp. 213–322, May 2011.

[12] J. Edmonds and M. Luby, “Erasure codes with a hierarchical bundle structure,” *IEEE Trans. Inf. Theory*, 2017, to appear.

[13] 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.

[14] M. Luby, A. Shokrollahi, M. Watson, and T. Stockhammer, “Raptor Forward Error Correction Scheme for Object Delivery,” Internet Requests for Comments, RFC Editor, RFC 5053, Oct. 2007.