Coded Computation over Heterogeneous Clusters

Amirhossein Reisizadeh, Saurav Prakash, Ramtin Pedarsani, Amir Salman Avestimehr

Abstract*—In large-scale distributed computing clusters, such as Amazon EC2, there are several types of "system noise" that can result in major degradation of performance: system failures, bottlenecks due to limited communication bandwidth, latency due to straggler nodes, etc. There have been recent results that demonstrate the impact of coding for efficient utilization of computation and storage redundancy to alleviate the effect of stragglers and communication bottlenecks in *homogeneous* clusters. In this paper, we focus on general *heterogeneous* distributed computing clusters consisting of a variety of computing machines with different capabilities. We propose a coding framework for speeding up distributed computing in heterogeneous clusters by trading redundancy for reducing the latency of computation. In particular, we propose *Heterogeneous Coded Matrix Multiplication (HCMM)* algorithm for performing distributed matrix multiplication over heterogeneous clusters that is provably asymptotically optimal for a broad class of processing time distributions. Moreover, we show that HCMM is unboundedly faster than *any uncoded* scheme that partitions the total work load among the workers. To demonstrate how the proposed HCMM scheme can be applied in practice, we provide results from numerical studies and Amazon EC2 experiments comparing HCMM with three benchmark load allocation schemes – Uniform Uncoded, Load-balanced Uncoded, and Uniform Coded. In particular, in our numerical studies, HCMM achieves speedups of up to 73%, 56% and 42% respectively over the three benchmark schemes mentioned above. Furthermore, we carry out experiments over Amazon EC2 clusters and demonstrate how HCMM can be combined with rateless codes with nearly linear decoding complexity. In particular, we show that HCMM combined with the Luby transform (LT) codes can significantly reduce the overall execution time. HCMM is found to be up to 61%, 46% and 36% faster than the aforementioned three benchmark schemes, respectively. Additionally, we provide a generalization to the problem of optimal load allocation in heterogeneous settings, where we take into account the monetary costs associated with distributed computing clusters. We argue that HCMM is asymptotically optimal for budget-constrained scenarios as well. In particular, we characterize the minimum possible expected cost associated with a computation task over a given cluster of machines. Furthermore, we develop a heuristic algorithm for (HCMM) load allocation for the distributed implementation of budget-limited computation tasks.

***Index Terms*—Coded computation, distributed computing, heterogeneous clusters.**

I. Introduction

A. Reisizadeh and R. Pedarsani are with the Department of Electrical and Computer Engineering, University of California, Santa Barbara, Santa Barbara, CA 93106 USA (e-mail: reisizadeh@ucsb.edu; ramtin@ece.ucsb.edu).

S. Prakash and A. S. Avestimehr are with the the Department of Electrical and Computer Engineering, University of Southern California, Los Angeles, CA 90089 USA (e-mail: sauravpr@usc.edu; avestimehr@ee.usc.edu).

A part of this work was presented in IEEE International Symposium on Information Theory, 2017 [1].

GENERAL distributed computing frameworks, such as MapReduce [2] and Spark [3], along with the availability of large-scale commodity servers, such as Amazon EC2, have made it possible to carry out large-scale data analytics at the production level. These "virtualized data centers" enjoy an abundance of storage space and computing power, and are cheaper to rent by the hour than maintaining dedicated data centers round the year. However, these systems suffer from various forms of "system noise" which reduce their efficiency: system failures, limited communication bandwidth, straggler nodes, etc.

The current state-of-the-art approaches to mitigate the impact of system noise in cloud computing environments involve creation of some form of "computation redundancy". For example, *replicating* the straggling task on another available node is a common approach to deal with stragglers [4], while partial data replication is also used to reduce the communication load in distributed computing [5]. However, there have been recent results demonstrating that *coding* can play a transformational role for creating and exploiting computation redundancy to effectively alleviate the impact of system noise. In particular, there have been two coding concepts proposed to deal with the communication and straggler bottlenecks in distributed computing.

The first coding concept introduced in [6]–[8] enables an inverse-linear tradeoff between computation load and communication load in distributed computing. This result implies that increasing the computation load by a factor of r (i.e. evaluating each computation at r carefully chosen nodes) can create novel coding opportunities that reduce the required communication load for computing by the same factor r. Hence, these codes can be utilized to pool the underutilized computing resources at network edge to slash the communication load of Fog computing [9]. Other related works tackling the communication bottleneck in distributed computation include [10]–[14].

In the second coding concept introduced in [10], an inverse-linear tradeoff between computation load and computation latency (i.e. the overall job response time) is established for distributed matrix multiplication in homogeneous computing environments. More specifically, this approach utilizes coding to effectively inject redundant computations to alleviate the effects of stragglers and speed up the computations. Hence, by utilizing more computation resources, this can significantly speed up distributed computing applications. A number of related works have been proposed recently to mitigate stragglers in distributed computation. In [15], the authors propose the use of redundant short dot products to speed up distributed computation of linear transforms. The work in [16] proposes coding schemes for mitigating stragglers in distributed batch gradient computation. Coding schemes for high-dimensional matrix-matrix multiplication have been developed in [17]–

[21]. Techniques for efficient straggler mitigation for matrix-vector computation in distributed wireless settings have been developed in [22]. In [23], the potential of the multicore nature of computing machines is studied. In [24], the authors propose an anytime approach to distributed computing, developing an approximate matrix multiplication scheme. The authors in [25] propose a novel encoding scheme for achieving large sparsity in the encoded matrix. Work in [26] develops a coding strategy for mitigating straggling decoders in cloud radio access network. Speeding up the computation of linear transformations with unreliable components is studied in [27]. Straggler mitigation through data encoding in distributed optimization is proposed in [28]. A coded scheme based on LT codes is proposed in [29] for multiplying a matrix by a set of vectors in a distributed computing environment. Addressing stragglers has attracted a lot of attention in the queuing-based frameworks for large-scale computation as well [30], [31]. These works utilize the technique of dynamically replicating the tasks in a careful manner to minimize run-time.

We extend the problem of distributed matrix multiplication in homogeneous clusters in [10] to heterogeneous environments. As discussed in [4], the computing environments in virtualized data centers are heterogeneous and algorithms based on homogeneous assumptions can result in significant performance reduction. In this paper, we focus on general *heterogeneous* distributed computing clusters consisting of a variety of computing machines with different capabilities. Specifically, we propose a coding framework for speeding up distributed matrix multiplication in heterogeneous clusters with straggling servers, named *Heterogeneous Coded Matrix Multiplication (HCMM)*. Matrix multiplication is a crucial computation module in many engineering and scientific disciplines. In particular, it is a fundamental component of many popular machine learning algorithms such as logistic regression, reinforcement learning and gradient descent-based algorithms. Implementations that speed up matrix multiplication would naturally speed up the execution of a wide variety of popular algorithms. Therefore, we envision HCMM to play a fundamental role in speeding up big data analytics in virtualized data centers by leveraging the wide range of computing capabilities provided by these heterogeneous environments.

We now describe the main ideas behind HCMM, which results in asymptotically optimal performance. In a coded implementation of distributed matrix-vector multiplication, each worker node is assigned the task of computing inner products of the assigned coded rows with the input vector, where the assigned coded rows are random linear combinations of the rows of the original matrix. Computation time at each worker is a random variable, which is first assumed to have shifted exponential distribution, and we later generalize it to shifted Weibull distribution. The master node receives the results from the worker nodes and aggregates them until it receives a decodable set of inner products and recovers the matrix-vector multiplication. We are interested in finding the optimal load allocation that minimizes the expected waiting time to complete this computation. However, due to heterogeneity, finding the exact solution to the optimization problem seems intractable.

As the main contribution of the paper, we propose an alternative optimization that focuses on maximizing the expected number of returned computation results from the workers. Apart from being computationally tractable, the alternative optimization asymptotically approximates the problem of finding the optimal computation load allocation. Specifically, we develop the HCMM algorithm that is derived as a solution to the alternative formulation, and prove it is asymptotically optimal. Furthermore, we prove that given a heterogeneous cluster of n workers, HCMM is $\Theta(\log n)$ times faster than uncoded schemes under the shifted exponential distribution for run-time. We further generalize the proposed HCMM algorithm to shifted Weibull model and provide similar unbounded gains over uncoded scenarios.

In addition to proving the asymptotic optimality of HCMM, we carry out numerical studies and experiments over Amazon EC2 clusters to demonstrate how HCMM can be used in practice. We compare HCMM with three benchmark schemes – Uniform Uncoded, Load-balanced Uncoded, and Uniform Coded. In our numerical analysis, HCMM results in significant speedups of up to 73%, 56% and 42% over the three aforementioned benchmark schemes, respectively. In experiments using Amazon EC2 clusters, we use the Luby transform (LT) codes for coding and demonstrate that HCMM combined with LT codes significantly reduces the overall execution time in comparison to uncoded and coded schemes. In particular, HCMM achieves gains of up to 61%, 46% and 36%, respectively over Uniform Uncoded, Load-balanced Uncoded and Uniform Coded. Furthermore, the overall computation load of HCMM is less than the one of Uniform Coded. Our results demonstrate that HCMM combines the benefits of both Load-balanced Uncoded and Uniform Coded schemes by achieving efficient load balancing along with minimal number of redundant computations.

Furthermore, we consider the problem of load allocation under budget constraints, considering an intuitive and convincing pricing model. In particular, we show that HCMM is the (asymptotically) optimal load allocation in feasible budget-constrained scenarios as well, and determine whether a budget-constrained computation task is feasible given a cluster of machines. We then develop a heuristic algorithm to find the (sub)optimal load allocations using the proposed HCMM scheme. The heuristic is based on the observation that given a computation task and a set of machines, decreasing the number of fastest machines participating in HCMM results in smaller average cost.

Notation. We denote by $[n]$ the set $\{1,\cdots,n\}$ for any $n\in\mathbb{N}$. For non-negative sequences $g(n)$ and $h(n)$, we denote $g(n)=\mathcal{O}\big(h(n)\big)$ if there exist constants $c>0$ and $n_0\in\mathbb{N}$ such that $g(n)\leq c\cdot h(n)$ for all $n>n_0$; and $g(n)=\Theta\big(h(n)\big)$ if $g(n)=\mathcal{O}\big(h(n)\big)$ and $h(n)=\mathcal{O}\big(g(n)\big)$. Moreover, we write $g(n)=o\big(h(n)\big)$ if $\lim_{n\to\infty} g(n)/h(n)=0$.

II. Problem Formulation and Main Results

In this section, we describe our computation model, the network model and the precise problem formulation. We then conclude with four theorems highlighting the main contributions of the paper.

A. Computation Model

We consider the problem of matrix-vector multiplication, in which given a matrix $\mathbf{A} \in \mathbb{R}^{r \times m}$ for some positive integers r and m, we want to compute the output $\mathbf{y} = \mathbf{A}\mathbf{x}$ for an input vector $\mathbf{x} \in \mathbb{R}^m$. Due to limited computing power, the computation cannot be carried out at a single server and a distributed implementation is required. As an example, consider a matrix $\mathbf{A}$ with an even number of rows and two computing nodes. The matrix can be divided into two equally tall matrices $\mathbf{A}_1$ and $\mathbf{A}_2$, and each will be stored in a different worker node. The master node receives the input $\mathbf{x}$ and broadcasts it to the two worker nodes. These nodes will then compute $\mathbf{y}_1 = \mathbf{A}_1\mathbf{x}$ and $\mathbf{y}_2 = \mathbf{A}_2\mathbf{x}$ locally and return their results to the master node, which combines them to obtain the intended outcome $\mathbf{y} = [\mathbf{y}_1; \mathbf{y}_2] = \mathbf{A}\mathbf{x}$. This example also illustrates an uncoded implementation of distributed computing, in which results from all the worker nodes are required to recover the final result.

We now present the formal definition of *Coded Distributed Computation*.

Definition 1. (Coded Distributed Computation) The coded distributed implementation of a computation task $f_{\mathbf{A}}(\cdot)$ is specified by:

- local data blocks $\langle \mathbf{A}_i \rangle_{i=1}^n$ and local computation tasks $\langle f^i_{\mathbf{A}_i}(\cdot) \rangle_{i=1}^n$;
- a decoding function that outputs $f_{\mathbf{A}}(\cdot)$ given the results from a decodable set of local computations.

For matrix-vector multiplication tasks in particular, local data blocks $\mathbf{A}_i \in \mathbb{R}^{\ell_i \times m}$ are matrices consisting of *coded* combinations of the rows in $\mathbf{A}$, for non-negative integers ℓ_i. To assign the computation tasks to each worker, we use random linear combinations of the r rows of the matrix $\mathbf{A}$, such that the master node can recover the result $\mathbf{A}\mathbf{x}$ from any r inner products received from the worker nodes with probability 1. As an example, if worker i is assigned a matrix-vector multiplication with matrix size $\ell_i \times m$, it will compute ℓ_i inner products of the assigned coded rows of $\mathbf{A}$ with $\mathbf{x}$. The master node shall wait for the first r inner products and will use them to decode the required output. In order to ensure the recovery of the output from any r inner products received from the workers, we pick the computation matrix assigned to worker i as $\mathbf{A}_i = \mathbf{S}_i\mathbf{A}$, where $\mathbf{S}_i \in \mathbb{R}^{\ell_i \times r}$ is the coding matrix with i.i.d. $\mathcal{N}(0, 1)$ entries. Worker i computes $\mathbf{A}_i\mathbf{x}$ and returns the result to the master node. Upon receiving r inner products, the aggregated results at the master will be in the form of $\mathbf{z} = \mathbf{S}_{(r)}\mathbf{A}\mathbf{x}$, where $\mathbf{S}_{(r)} \in \mathbb{R}^{r \times r}$ is the aggregated coding matrix, and it is full-rank with probability 1 [32]. Therefore, the master node can recover $\mathbf{A}\mathbf{x} = \mathbf{S}_{(r)}^{-1}\mathbf{z}$ with probability 1.[1,2]

B. Network Model

The network model is based on a master-worker setup illustrated in Fig. 1. The master node receives an input $\mathbf{x}$ and broadcasts it to all the workers. Each worker computes its assigned set of computations and unicasts the result to the master node. The master node aggregates the results from the worker nodes until it receives a decodable set of computations and recovers the output $\mathbf{A}\mathbf{x}$.

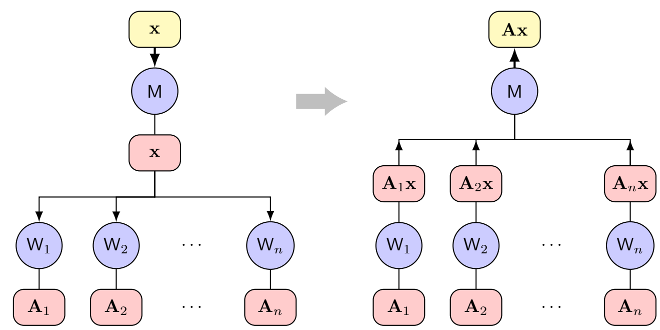


Fig. 1: Master-worker setup of the computing clusters: The master node receives the input vector $\mathbf{x}$ and broadcasts it to all the worker nodes. Upon receiving the input, worker node i starts computing the inner products of the input vector with the locally assigned rows, i.e., $\mathbf{y}_i = \mathbf{A}_i\mathbf{x}$, and unicasts the output vector $\mathbf{y}_i$ to the master node upon completing the computation. The results are aggregated at the master node until r inner products are received and the desired output $\mathbf{A}\mathbf{x}$ is recovered.

We denote by T_i the random variable representing the task run-time at node i and assume that the run-times $T_1, \cdots, T_n$ are mutually independent. We consider the distribution of run-time random variables to be exponential, and later generalize it to Weibull distribution. More specifically, we consider a 2-parameter shifted exponential distribution for the execution time of each worker, i.e., the CDF of execution time of worker node i, T_i, loaded with ℓ_i row vectors is as follows:

$$\Pr[T_i \le t] = 1 - e^{-\frac{\mu_i}{\ell_i}(t - a_i\ell_i)}, \tag{1}$$

for $t \ge a_i\ell_i$ and $i \in [n]$, where $a_i > 0$ is the shift parameter and $\mu_i > 0$ denotes the straggling parameter associated with worker node i. The shifted exponential model for computation time, which is the sum of a constant (deterministic) term and a variable (stochastic) term, is motivated by the distribution model proposed by authors in [33] for latency in querying data files from cloud storage systems. As demonstrated in [10] as

[1] Although we consider random linear coding in our theoretical analysis, other codes such as Maximum-Distance Separable (MDS) codes and Luby transform (LT) codes are compatible with HCMM as well, given a decodable set of results at the master. For example, in the MDS case, the entries in the coding matrix $\{\mathbf{S}_i\}_{i=1}^n$ are drawn from a finite field. Specifically, one can encode the rows of $\mathbf{A}$ using an $(\sum_{i=1}^n \ell_i, r)$ MDS code and assign ℓ_i coded rows to the worker node i. The output $\mathbf{A}\mathbf{x}$ can be recovered from the inner products of any r coded rows with the input vector $\mathbf{x}$. Furthermore, to implement the ideas developed in this work, we use LT codes in our experiments over Amazon EC2 clusters.

[2] Instead of i.i.d. Gaussian, we could use any continuous distribution for the random entries, since Schwartz-Zippel lemma ensures that such random matrix is full-rank with high probability

well as by our own experiments, exponential model provides a good fit for the distribution of computation times over cloud computing environments such as Amazon EC2 clusters. Moreover, these experiments confirm the assumption that as a first order approximation, both shift and mean parameters of the shifted exponential distributions linearly scale with the load size.

We further generalize the analysis to shifted Weibull distribution in Section IV, where we consider a 3-parameter shifted Weibull distribution for the execution time of each worker. That is, the CDF of task run-time at worker node i, loaded with ℓ_i row vectors is as follows:

$$\Pr[T_i \leq t] = 1 - e^{-\left(\frac{\mu_i}{\ell_i}(t - a_i\ell_i)\right)^{\alpha_i}}, \tag{2}$$

for $t \geq a_i\ell_i$ and $i \in [n]$, where $a_i > 0$ denotes the shift parameter, $\mu_i > 0$ is the straggling parameter and $\alpha_i > 0$ represents the shape parameter associated with worker i. A similar model has been considered in [34] as well.

C. Problem Formulation

We consider the problem of using a cluster of n worker nodes for distributedly computing the matrix-vector multiplication $\mathbf{Ax}$, where $\mathbf{A}$ is a size $r \times m$ matrix for positive integers r and m. Let $\boldsymbol{\ell} = (\ell_1, \cdots, \ell_n)$ be the load allocation vector where ℓ_i denotes the number of rows assigned to worker node i. Let T_{CMP} be the random variable denoting the waiting time for receiving a decodable set of results, i.e. at least r inner products. We aim at finding the optimal load allocation vector that minimizes the average waiting time by solving the following optimization problem:

$$\mathcal{P}_{\text{main}}: \quad \underset{\boldsymbol{\ell}}{\text{minimize}} \quad \mathbb{E}[T_{\mathsf{CMP}}]. \tag{3}$$

For a homogeneous cluster, to achieve a coded solution, one can divide $\mathbf{A}$ into k equal size submatrices, and apply an (n, k) MDS code to these submatrices. The master node can then obtain the final result from any k responses. In [10], the authors find the optimal k for minimizing the average running time for the shifted exponential run-time model.

For heterogeneous clusters, however, assigning equal loads to servers is clearly not optimal. Moreover, directly finding the optimal solution to $\mathcal{P}_{\text{main}}$ is hard. In homogeneous clusters, the problem of finding a sufficient number of inner products can be mapped to the problem of finding the waiting time for a set of fastest responses, and thus closed form expressions for the expected computation time can be found using order statistics of i.i.d. run-times. However, this is not straight-forward in heterogeneous clusters, where the load allocation is non-uniform. In Section III, we present an alternative formulation to $\mathcal{P}_{\text{main}}$ in (3), and show that the solution to the alternative formulation – which we shall name HCMM – is tractable and provably asymptotically optimal.

Assumptions. From now onward, we consider the practically relevant regime where the size of the problem scales linearly with the size of the network, while the computing power and the storage capacity of each worker node remain constant. Specifically, we assume $r = \Theta(n)$, $a_i = \Theta(1)$, $\mu_i = \Theta(1)$ and $\alpha_i = \Theta(1)$ for each worker i.

D. Main Results

Having set the model and formulation of the problem, we now present the main contributions of this paper. The following theorem characterizes the asymptotic optimality of HCMM for the shifted exponential run-time model.

Theorem 1. *Let T_{HCMM} be the random variable denoting the finish time of the HCMM algorithm and T_{OPT} be the random variable representing the finish time of the optimum algorithm obtained by solving $\mathcal{P}_{\text{main}}$. Then, for shifted exponential run-times in (1) with constant parameters $a_i = \Theta(1)$ and $\mu_i = \Theta(1)$ for each worker $i \in [n]$ and $r = \Theta(n)$, we have* $\lim_{n\to\infty} \mathbb{E}[T_{\mathsf{HCMM}}] = \lim_{n\to\infty} \mathbb{E}[T_{\mathsf{OPT}}]$.

Remark 1. Theorem 1 demonstrates that our proposed HCMM algorithm is asymptotically optimal as the number of workers n approaches infinity. In other words, the optimal computation load allocation problem $\mathcal{P}_{\text{main}}$ in (3) can be optimally solved using the proposed HCMM algorithm as n gets large.

Remark 2. We note that $\mathcal{P}_{\text{main}}$ in (3) is a hard combinatorial optimization problem since it will require checking all load combinations to minimize the overall expected execution time. The key idea in Theorem 1 is to consider an alternative formulation to (3) focusing on maximizing the expected number of returned computation results from the workers, i.e. maximizing the *aggregate return*. As we describe in Section III, the alternative optimization problem not only can be solved efficiently in a tractable way giving rise to HCMM algorithm, it also asymptotically approximates $\mathcal{P}_{\text{main}}$ and allows us to establish Theorem 1.

Remark 3. While Theorem 1 theoretically characterizes the optimality of our proposed scheme HCMM, we also demonstrate gains that one can get in practice. In particular, we carry out numerical studies and experiments over Amazon EC2 clusters that demonstrate that HCMM can provide significant gains in a wide variety of computing scenarios. In particular, we compare HCMM's performance with three benchmark load allocation policies – Uniform Uncoded, Load-balanced Uncoded, and Uniform Coded. In numerical studies, HCMM achieves speedups of up to 71% over Uniform Uncoded, up to 53% over Load-balanced Uncoded, and up to 39% over Uniform Coded. In EC2 experiments, HCMM combined with the Luby transform (LT) codes provides speedups of up to 61%, 46% and 36% over Uniform Uncoded, Load-balanced Uncoded and Uniform Coded, respectively.

Theorem 2. *Let T_{UC} denote the completion time of the uncoded distributed matrix multiplication algorithm. Then, for the shifted exponential run-times with constant parameters and $r = \Theta(n)$,*

$$\frac{\mathbb{E}[T_{\mathsf{UC}}]}{\mathbb{E}[T_{\mathsf{HCMM}}]} = \Theta\big(\log n\big).$$

Remark 4. As Theorem 2 shows, our proposed HCMM guarantees an improvement of $\Theta\big(\log n\big)$ in expected execution time over *any* uncoded scheme, including the one that optimally allocates the workers' loads. This result illustrates that by leveraging coded computing, one achieves the same order-wise gain over heterogeneous clusters as over homogeneous clusters [10].

Although Theorems 1 and 2 are based on the shifted exponential model (1) for run-time random variables for the workers, our analyses are general and can be extended to other models. The following two theorems generalize the results when the execution time of each worker follows the Weibull distribution as described in (2).

Theorem 3. *For the shifted Weibull distribution of run-times with constant parameters $a_i = \Theta(1)$, $\mu_i = \Theta(1)$ and $\alpha_i = \Theta(1)$ for each worker $i \in [n]$ and $r = \Theta(n)$, the proposed HCMM algorithm is asymptotically optimal, i.e.,* $\lim_{n\to\infty} \mathbb{E}[T_{\mathsf{HCMM}}] = \lim_{n\to\infty} \mathbb{E}[T_{\mathsf{OPT}}]$.

Theorem 4. *Under the Weibull distribution for run-times with constant parameters and $r = \Theta(n)$, the proposed HCMM scheme unboundedly outperforms the uncoded scheme, i.e.,*

$$\frac{\mathbb{E}[T_{\mathsf{UC}}]}{\mathbb{E}[T_{\mathsf{HCMM}}]} \geq \Theta\big((\log n)^{1/\tilde{\alpha}}\big),$$

where $\tilde{\alpha} = \max_{i\in[n]} \alpha_i$ is the largest shape parameter among the workers.

Remark 5. As stated in Theorem 4, HCMM provides an unbounded gain over *any* uncoded scheme – including the optimal uncoded load allocation – under the Weibull distribution for workers' run-times. Furthermore, our numerical simulations demonstrate speedups of up to 73%, 56% and 42% over Uniform Uncoded, Load-balanced Uncoded and Uniform Coded, respectively.

In the following section, we describe our alternative formulation based on aggregate return and describe our proposed HCMM algorithm that solves the alternative optimization.

III. The Proposed HCMM Scheme and Proofs of Theorems 1 and 2

In this section, we prove Theorems 1 and 2 for the exponential model (1). In particular, we start by describing the HCMM algorithm and show that it asymptotically achieves the optimal performance, as stated in Theorem 1, and lastly conclude the section by characterizing the gain of HCMM over uncoded scheme.

To derive HCMM, we start by reformulating $\mathcal{P}_{\text{main}}$ defined in (3) and show that the alternative formulation can be efficiently solved, as opposed to solving $\mathcal{P}_{\text{main}}$ that needs an exhaustive search over all possible load allocations. The solution to the alternative problem gives rise to HCMM. We will further prove the optimality of HCMM and compare its average run-time to uncoded schemes.

A. Alternative Formulation of $\mathcal{P}_{\text{main}}$ via Maximal Aggregate Return

Consider an n-tuple load allocation $\boldsymbol{\ell} = (\ell_1, \cdots, \ell_n)$ and let t be a feasible time for computation, i.e., $t \geq \max_i \{a_i\, \ell_i\}$. The number of equations received from worker $i \in [n]$ at the master node till time t is a random variable, $X_i(t) = \ell_i\, \mathbb{1}_{\{T_i \leq t\}}$, where T_i is the random execution time for machine i that is assigned the load ℓ_i and $\mathbb{1}_{\{.\}}$ denotes the indicator function. Then, the *aggregate return* at the master node at time t is:

$$X(t) = \sum_{i=1}^{n} X_i(t).$$

We propose the following two-step alternative formulation for $\mathcal{P}_{\text{main}}$ defined in (3). First, for a fixed feasible time t, we maximize the aggregate return over different load allocations, i.e., we solve

$$\mathcal{P}^{(1)}_{\text{alt}} : \boldsymbol{\ell}^*(t) = \arg\max_{\boldsymbol{\ell}} \mathbb{E}\big[X(t)\big]. \tag{4}$$

Then, given the load allocation $\boldsymbol{\ell}^*(t) = \big(\ell^*_1(t), \cdots, \ell^*_n(t)\big)$ obtained from $\mathcal{P}^{(1)}_{\text{alt}}$, we find the smallest time t such that with high probability, there is enough aggregate return by time t at the master node, i.e., we solve

$$\begin{aligned} \mathcal{P}^{(2)}_{\text{alt}} : \quad & \text{minimize} \quad t \\ & \text{subject to} \quad \Pr\big[X^*(t) < r\big] = o\left(\frac{1}{n}\right), \end{aligned} \tag{5}$$

where $X^*(t)$ is the aggregate return at time t for load allocation obtained from $\mathcal{P}^{(1)}_{\text{alt}}$, that is

$$X^*(t) = \sum_{i=1}^{n} X^*_i(t) = \sum_{i=1}^{n} \ell^*_i(t)\, \mathbb{1}_{\{T_i \leq t\}}.$$

From now onward, we denote the solution to $\mathcal{P}^{(2)}_{\text{alt}}$ by t^* and hence $\boldsymbol{\ell}^*(t^*)$ denotes the solution to the two-step alternative formulation in (4) and (5) which gives rise to our proposed HCMM scheme described next.

B. Solving the Alternative Formulation

Considering the exponential distribution for workers' run-times, we first proceed to solve $\mathcal{P}^{(1)}_{\text{alt}}$ in (4). The expected number of equations aggregated at the master node at time t is:

$$\mathbb{E}\big[X(t)\big] = \sum_{i=1}^{n} \mathbb{E}\big[X_i(t)\big] = \sum_{i=1}^{n} \ell_i \left(1 - e^{-\frac{\mu_i}{\ell_i}(t - a_i\, \ell_i)}\right).$$

Since there is no constraint on load allocations, $\mathcal{P}^{(1)}_{\text{alt}}$ can be decomposed to n decoupled optimization problems, i.e.,

$$\ell^*_i(t) = \arg\max_{\ell_i} \mathbb{E}\big[X_i(t)\big], \tag{6}$$

for all workers $i \in [n]$. The solution to (6) satisfies the following optimality condition:

$$\frac{\partial}{\partial \ell_i} \mathbb{E}\left[X_i(t)\right] = 1 - e^{-\frac{\mu_i}{\ell_i}(t - a_i \ell_i)} \left(\frac{\mu_i t}{\ell_i} + 1\right) = 0,$$

which yields

$$\ell^*_i(t) = \frac{t}{\lambda_i}, \tag{7}$$

where $\lambda_i = \Theta(1)$ is a constant independent of t and is the positive solution to the following equation:

$$e^{\mu_i \lambda_i} = e^{a_i \mu_i}(\mu_i \lambda_i + 1).$$

Algorithm 1 Heterogeneous Coded Matrix Multiplication (HCMM)

Input: computation time parameters (a_i, μ_i) for each worker i [3]
Output: computation load assigned to each worker i

procedure HCMM
  solve $\mathcal{P}_{\text{alt}}^{(1)}$ for any feasible t
  obtain $\ell_i^*(t) = \frac{t}{\lambda_i}$
  solve $\mathcal{P}_{\text{alt}}^{(2)}$ and obtain t^*
  return $\ell_i^*(t^*) = \frac{t^*}{\lambda_i}$ row vector computations for worker i

One can easily check that the condition $t \geq a_i \ell_i^*(t)$ holds for all i as well. Moreover, we denote by t^* the solution to $\mathcal{P}_{\text{alt}}^{(2)}$. Now, we define the HCMM load allocation as

$$\ell_i^*(t^*) = \frac{t^*}{\lambda_i}, \tag{8}$$

for all workers i. In the following, we formally define the HCMM algorithm which is basically the solution to $\mathcal{P}_{\text{alt}}$.

Remark 6. We note that in order to implement any load allocation scheme, each worker supposedly admits an integer number of rows as its associated computation load. However, the load allocation $\ell_i^*(t^*)$ given by HCMM scheme in Algorithm 1 is a real number for any worker i and therefore one needs to round the result before proceeding with experiments. In practical scenarios, $\ell_i^*(t^*)$ is fairly large, e.g. in the order of 100 row vectors. Therefore, the effect of rounding the load allocations shall be insignificant.

We now provide an approximation to t^* and show it asymptotically converges to t^*. The expected aggregate return at time t for optimal loads obtained in (7) is

$$\begin{aligned}\mathbb{E}\left[X^*(t)\right] &= \sum_{i=1}^{n} \ell_i^*(t)\left(1 - e^{-\frac{\mu_i}{\ell_i^*(t)}(t - a_i \ell_i^*(t))}\right) \\ &= \sum_{i=1}^{n} \frac{t}{\lambda_i}\left(1 - e^{-\frac{\mu_i}{t/\lambda_i}\left(t - \frac{a_i t}{\lambda_i}\right)}\right) \\ &= ts, \end{aligned} \tag{9}$$

where

$$s = \sum_{i=1}^{n} \frac{1}{\lambda_i}\left(1 - e^{-\mu_i \lambda_i (1 - \frac{a_i}{\lambda_i})}\right) = \sum_{i=1}^{n} \frac{\mu_i}{1 + \mu_i \lambda_i} = \Theta(n),$$

since $\mu_i = \Theta(1)$ and $\lambda_i = \Theta(1)$. Let τ^* be the solution to the following equation when solved for t:

$$\mathbb{E}\big[X^*(t)\big] = \sum_{i=1}^{n} \ell_i^*(t)\left(1 - e^{-\frac{\mu_i}{\ell_i^*(t)}(t - a_i \ell_i^*(t))}\right) = r. \tag{10}$$

In other words, τ^* is the time for which there are exactly r inner products – on average – aggregated at the master node, when the workers are loaded according to the loading obtained in (7). Using (7), (9) and (10), we find that

$$\tau^* = \frac{r}{s} = \Theta(1), \tag{11}$$

$$\ell_i^*(\tau^*) = \frac{\tau^*}{\lambda_i} = \frac{r}{s\lambda_i} = \Theta(1). \tag{12}$$

We now present the following lemma, which shows that τ^* converges to t^* for large n (see Appendix for proof).

Lemma 1. *Let t^* be the solution to the alternative formulation $\mathcal{P}_{\text{alt}}$ in (4-5) and τ^* be the solution to (10). Then,*

$$\tau^* \leq t^* \leq \tau^* + o(1).$$

C. Asymptotic Optimality of HCMM

In this subsection, we prove the asymptotic optimality of HCMM as claimed in Theorem 1.

Proof of Theorem 1. Consider the HCMM load assignment in (8). Let the random variable T_{HCMM} denote the finish time associated to this load allocation, i.e. the waiting time to receive at least r inner products from the workers. Let T_{max} be the random variable denoting the finish time of all the workers for the HCMM load assignment.

First, we show that

$$\mathbb{E}[T_{\mathsf{HCMM}}] \leq t^* + o(1).$$

Let us define two events $\mathcal{E}_1$ and $\mathcal{E}_2$ as follows:

$$\mathcal{E}_1 = \{T_{\mathsf{max}} > \Theta(n)\} \text{ and } \mathcal{E}_2 = \{T_{\mathsf{HCMM}} > t^*\}.$$

Conditioning on these events, we can write

$$\begin{aligned}\mathbb{E}[T_{\mathsf{HCMM}}] = {} & \mathbb{E}[T_{\mathsf{HCMM}} | \mathcal{E}_1] \Pr[\mathcal{E}_1] \\ & + \mathbb{E}[T_{\mathsf{HCMM}} | {\mathcal{E}_1}^c \cap \mathcal{E}_2] \Pr[{\mathcal{E}_1}^c \cap \mathcal{E}_2] \\ & + \mathbb{E}[T_{\mathsf{HCMM}} | {\mathcal{E}_1}^c \cap {\mathcal{E}_2}^c] \Pr[{\mathcal{E}_1}^c \cap {\mathcal{E}_2}^c]. \end{aligned} \tag{13}$$

We can write the second term in RHS of (13) as follows:

$$\begin{aligned} & \mathbb{E}[T_{\mathsf{HCMM}} | {\mathcal{E}_1}^c \cap \mathcal{E}_2] \Pr[{\mathcal{E}_1}^c \cap \mathcal{E}_2] \\ & \quad = \mathbb{E}[T_{\mathsf{HCMM}} | T_{\mathsf{max}} \leq \Theta(n), T_{\mathsf{HCMM}} > t^*] \\ & \qquad \times \Pr[T_{\mathsf{max}} \leq \Theta(n), T_{\mathsf{HCMM}} > t^*] \\ & \quad \leq \mathbb{E}[T_{\mathsf{max}} | T_{\mathsf{max}} \leq \Theta(n), T_{\mathsf{HCMM}} > t^*] \Pr[T_{\mathsf{HCMM}} > t^*] \\ & \quad \overset{(a)}{\leq} \Theta(n) \cdot o\left(\frac{1}{n}\right) \\ & \quad = o(1). \end{aligned} \tag{14}$$

To prove (a), we note that HCMM returns r inner products by time T_{HCMM}. Moreover, the aggregate return is increasing in time. Therefore,

$$\Pr[T_{\mathsf{HCMM}} > t^*] \leq \Pr[X^*(t^*) < r] = o\left(\frac{1}{n}\right).$$

Furthermore, we have

$$\begin{aligned} & \mathbb{E}[T_{\mathsf{max}} | T_{\mathsf{max}} \leq \Theta(n), T_{\mathsf{HCMM}} > t^*] \\ & \quad = \frac{1}{\Pr[T_{\mathsf{max}} \leq \Theta(n), T_{\mathsf{HCMM}} > t^*]} \\ & \qquad \times \int_{t_1=0}^{\Theta(n)} \int_{t_2=t^*}^{\infty} t_1 d \Pr[T_{\mathsf{max}} \leq t_1, T_{\mathsf{HCMM}} \leq t_2] \\ & \quad \leq \frac{\Theta(n)}{\Pr[T_{\mathsf{max}} \leq \Theta(n), T_{\mathsf{HCMM}} > t^*]} \\ & \qquad \times \int_{t_1=0}^{\Theta(n)} \int_{t_2=t^*}^{\infty} d \Pr[T_{\mathsf{max}} \leq t_1, T_{\mathsf{HCMM}} \leq t_2] \\ & \quad = \Theta(n). \end{aligned}$$

[3]For the shifted Weibull distribution, parameters (a_i, μ_i, α_i) are taken as inputs.

Moreover, the third term in RHS of (13) can be written as

$$\begin{aligned}
&\mathbb{E}[T_{\mathsf{HCMM}}|\mathcal{E}_1{}^c \cap \mathcal{E}_2{}^c] \Pr[\mathcal{E}_1{}^c \cap \mathcal{E}_2{}^c] \\
&\qquad = \mathbb{E}[T_{\mathsf{HCMM}}|T_{\mathsf{max}} \le \Theta(n), T_{\mathsf{HCMM}} \le t^*] \\
&\qquad\quad \times \Pr[T_{\mathsf{max}} \le \Theta(n), T_{\mathsf{HCMM}} \le t^*] \\
&\qquad \le \mathbb{E}[T_{\mathsf{HCMM}}|T_{\mathsf{max}} \le \Theta(n), T_{\mathsf{HCMM}} \le t^*] \\
&\qquad \overset{(b)}{\le} t^*, \qquad (15)
\end{aligned}$$

where proof of (b) is similar to proof of (a) in (14). Regarding the first term in RHS of (13), we have

$$\begin{aligned}
\mathbb{E}[T_{\mathsf{HCMM}}|\mathcal{E}_1] \Pr[\mathcal{E}_1] &= \mathbb{E}[T_{\mathsf{HCMM}}|T_{\mathsf{max}} > \Theta(n)] \\
&\quad \times \Pr[T_{\mathsf{max}} > \Theta(n)] \\
&\le \mathbb{E}[T_{\mathsf{max}}|T_{\mathsf{max}} > \Theta(n)] \\
&\quad \times \Pr[T_{\mathsf{max}} > \Theta(n)] \\
&= \int_{\Theta(n)}^{\infty} t f_{\mathsf{max}}(t)\, dt \\
&\overset{(c)}{\le} \int_{\Theta(n)}^{\infty} t n k_1 e^{-k_1 t} \left(1 - e^{-k_1 t}\right)^{n-1} dt \\
&\le \int_{\Theta(n)}^{\infty} n k_1 t e^{-k_1 t}\, dt \\
&\le \int_{\Theta(n)}^{\infty} \frac{1}{t^2}\, dt = o(1), \qquad (16)
\end{aligned}$$

for some $k_1 = \Theta(1)$ and large enough n. To derive inequality (c), we find a stochastic upper bound on T_{max} by considering n i.i.d. copies of the worker run-times with largest shift and smallest straggling parameters that are also $\Theta(1)$, and use the PDF of the maximum of n i.i.d. exponential random variables. As we later use in the proof of Theorem 3, one can similarly write for the shifted Weibull distribution:

$$\begin{aligned}
&\mathbb{E}[T_{\mathsf{HCMM}}|\mathcal{E}_1] \Pr[\mathcal{E}_1] \\
&\qquad \le \int_{\Theta(n)}^{\infty} t f_{\mathsf{max}}(t)\, dt \\
&\qquad \le \int_{\Theta(n)}^{\infty} n k_1 k_2 t^{k_2} e^{-k_1 t^{k_2}} \left(1 - e^{-k_1 t^{k_2}}\right)^{n-1} dt \\
&\qquad \le \int_{\Theta(n)}^{\infty} n k_1 k_2 t^{k_2} e^{-k_1 t^{k_2}}\, dt \\
&\qquad \le \int_{\Theta(n)}^{\infty} \frac{1}{t^2}\, dt = o(1), \qquad (17)
\end{aligned}$$

for some constants k_1 and k_2. Therefore, using (14), (15) and (16) (or (17) for the shifted Weibull model) in (13) we have

$$\mathbb{E}[T_{\mathsf{HCMM}}] \le t^* + o(1).$$

Let $\boldsymbol{\ell}_{\mathsf{OPT}} = (\ell_{\mathsf{OPT},1}, \cdots, \ell_{\mathsf{OPT},n})$ denote the optimal load allocation corresponding to $\mathcal{P}_{\text{main}}$ in (3) and $X_{\mathsf{OPT}}(\cdot)$ represent the aggregate return under load allocation $\boldsymbol{\ell}_{\mathsf{OPT}}$. Now we prove the following lower bound on the average completion time of the optimum algorithm:

$$\mathbb{E}[T_{\mathsf{OPT}}] \ge t^* - o(1).$$

To this end, we show the following two inequalities,

$$\mathbb{E}[T_{\mathsf{OPT}}] \overset{(d)}{\ge} \tau - \delta_1 \overset{(e)}{\ge} t^* - \delta_2 - \delta_1,$$

where $\delta_1 = \Theta\left(\frac{\log n}{\sqrt{n}}\right)$, $\delta_2 = \Theta\left(\frac{\log n}{\sqrt{n}}\right)$ and τ is the solution to $\mathbb{E}[X_{\mathsf{OPT}}(\tau)] = r$. We have

$$\begin{aligned}
&r - \mathbb{E}[X_{\mathsf{OPT}}(\tau - \delta_1)] \\
&\qquad = \sum_{i=1}^{n} \ell_{\mathsf{OPT},i} \big(\Pr[T_i < \tau] - \Pr[T_i < \tau - \delta_1] \big) \\
&\qquad = \sum_{i=1}^{n} \ell_{\mathsf{OPT},i} \left(\frac{d}{d\tau} \Pr[T_i < \tau] \delta_1 + \mathcal{O}\left(\delta_1^2\right) \right) \\
&\qquad = \Theta(n\delta_1) + \mathcal{O}\left(n\delta_1^2\right) = \Theta(n\delta_1),
\end{aligned}$$

where we used the fact that $\ell_{\mathsf{OPT},i} = \Theta(1)$[4]. By McDiarmid's inequality (see Appendix for its description), we have

$$\begin{aligned}
&\Pr[X_{\mathsf{OPT}}(\tau - \delta_1) \ge r] \\
&\qquad = \Pr[X_{\mathsf{OPT}}(\tau - \delta_1) - \mathbb{E}[X_{\mathsf{OPT}}(\tau - \delta_1)] \\
&\qquad\qquad \ge r - \mathbb{E}[X_{\mathsf{OPT}}(\tau - \delta_1)]] \\
&\qquad \le \exp\left(- \frac{2\left(\mathbb{E}[X_{\mathsf{OPT}}(\tau - \delta_1)] - r\right)^2}{\sum_{i=1}^{n} \ell_{\mathsf{OPT},i}^2} \right) \\
&\qquad = e^{-\Theta\left(n\delta_1^2\right)} = o\left(\frac{1}{n}\right),
\end{aligned}$$

which implies inequality (d). We proceed to prove (e) by showing the following two inequalities,

$$\tau \ge \tau^*, \qquad (18)$$

$$\tau^* \ge t^* - \delta_2, \qquad (19)$$

where τ^* is obtained in (11). Given the fact that HCMM maximizes the expected aggregate return, we have

$$\mathbb{E}[X^*(t)] \ge \mathbb{E}[X_{\mathsf{OPT}}(t)],$$

for every feasible t, which implies (18). Moreover, Lemma 1 proves (19). All in all, we have

$$t^* - o(1) \le \mathbb{E}[T_{\mathsf{OPT}}] \le \mathbb{E}[T_{\mathsf{HCMM}}] \le t^* + o(1),$$

which yields $\lim_{n\to\infty} \mathbb{E}[T_{\mathsf{HCMM}}] = \lim_{n\to\infty} \mathbb{E}[T_{\mathsf{OPT}}]$ and the claim is concluded. □

D. Comparison with Uncoded Schemes

This subsection provides the proof of Theorem 2 by comparing the performance of HCMM to uncoded scheme. In an uncoded scheme, the redundancy factor is 1; thus, the master node has to wait for the results from all the worker nodes in order to complete the computation.

[4]We argue that the allocated loads in the optimum coded scheme are all $\Theta(1)$. Without loss of generality, suppose $\ell_{\mathsf{OPT},1} > \Theta(1)$ which implies $\lim_{n\to\infty} \Pr[T_1 < t] = 0$ for any $t = \Theta(1)$. We have already implemented HCMM, a (sub-)optimal algorithm achieving computation time $\tau^* = \Theta(1)$, therefore the optimal scheme should have a better finishing time $\tau \le \Theta(1)$. Now assume the load of machine 1 is replaced by $\tilde{\ell}_{\mathsf{OPT},1} = \Theta(1)$. Clearly, for any time $t = \Theta(1)$, the aggregate return for the new set of loads is larger than the former one by any $\Theta(1)$ time, almost surely. This is in contradiction to optimality assumption.

Proof of Theorem 2. We start by characterizing the expected run-time of the best uncoded scheme. Particularly, we show that

$$\mathbb{E}[T_{\mathsf{UC}}] = \Theta\big(\log n\big),$$

where T_{UC} denotes the completion time of the optimum uncoded distributed matrix multiplication algorithm. To do so, we start by showing that

$$\mathbb{E}[T_{\mathsf{UC}}] \geq c \log n,$$

for a constant c independent of n. For a set of machines with parameters $\{(a_i, \mu_i)\}_{i=1}^{n}$, let $\tilde{a} = \min_i a_i$ and $\tilde{\mu} = \max_i \mu_i$. Now, consider another set of n machines in which every machine is replaced with a faster machine with parameters $(\tilde{a}, \tilde{\mu})$. Since the computation times of the new set of machines are i.i.d., one can show that the optimal load allocation for these machines is uniform, i.e.,

$$\widetilde{\ell}_i^* = \frac{r}{n},$$

for every machine $i \in [n]$. Let $\{\widetilde{T}_i\}_{i=1}^{n}$ represent the i.i.d. shifted exponential random variables denoting the execution times for the new set of machines where each machine is loaded by $\widetilde{\ell}_i^* = \frac{r}{n}$. Therefore, the CDF of the completion time of each new machine can be written as

$$\begin{aligned}\Pr\left[\widetilde{T}_i \leq t\right] =& 1 - e^{-\frac{\tilde{\mu}}{\widetilde{\ell}_i^*}\left(t - \tilde{a}\widetilde{\ell}_i^*\right)} \\ =& 1 - e^{-\tilde{\mu}\frac{n}{r}\left(t - \tilde{a}\frac{r}{n}\right)},\end{aligned}$$

for $t \geq \frac{\tilde{a}r}{n}$ and the expected computation time can be written as

$$\mathbb{E}\left[\widetilde{T}_i\right] = \frac{r}{n}\left(\tilde{a} + \frac{1}{\tilde{\mu}}\right),$$

for all $i \in [n]$. Since the master needs to wait for all of the machines to return their results, the total run-time is $\widetilde{T}_{\mathsf{UC}} = \max_{i \in [n]} \widetilde{T}_i$. Therefore,

$$\mathbb{E}\left[\widetilde{T}_{\mathsf{UC}}\right] = \mathbb{E}\left[\max_{i \in [n]} \widetilde{T}_i\right] = \frac{\tilde{a}r}{n} + \frac{rH_n}{n\tilde{\mu}}, \tag{20}$$

where $H_n = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n}$ is the sum of the harmonic series. We can further bound (20) using the fact that

$$\frac{\tilde{a}r}{n} + \frac{rH_n}{n\tilde{\mu}} \geq \frac{\tilde{a}r}{n} + \frac{r}{n\tilde{\mu}} \log(n+1) \geq c \log n,$$

for a constant c independent of n, since $r = \Theta(n)$, $\tilde{a} = \Theta(1)$, and $\tilde{\mu} = \Theta(1)$ for all $i \in [n]$. All in all, we have the following lower bound on the optimal uncoded scheme:

$$\mathbb{E}[T_{\mathsf{UC}}] \geq \mathbb{E}\left[\widetilde{T}_{\mathsf{UC}}\right] \geq c \log n. \tag{21}$$

Now consider another set of n machines, where each machine is replaced with a slower one with parameters $(\hat{a}, \hat{\mu})$ for $\hat{a} = \max_i a_i$ and $\hat{\mu} = \min_i \mu_i$. By an argument similar to the one employed the lower bound, we can write

$$\mathbb{E}[T_{\mathsf{UC}}] \leq \frac{\hat{a}r}{n} + \frac{r}{n\hat{\mu}} H_n \leq C \log n, \tag{22}$$

for another constant C. From (21) and (22), one can conclude that

$$\mathbb{E}[T_{\mathsf{UC}}] = \Theta\big(\log n\big). \tag{23}$$

Further, by Theorem 1 and Lemma 1, we find that

$$\mathbb{E}[T_{\mathsf{HCMM}}] = \Theta(1). \tag{24}$$

Comparing (23) to (24) demonstrates that HCMM outperforms the best uncoded scheme by a factor of $\Theta(\log n)$, i.e.,

$$\frac{\mathbb{E}[T_{\mathsf{UC}}]}{\mathbb{E}[T_{\mathsf{HCMM}}]} = \Theta\big(\log n\big).$$

□

IV. Generalization to the Shifted Weibull Model and Proofs of Theorems 3 and 4

In this section, we consider the shifted Weibull distribution for the workers' execution times, which captures a broader class of run-time models than the exponential distribution. We particularly generalize our proposed HCMM algorithm to the class of shifted Weibull distributed run-times and prove Theorems 3 and 4. More specifically, we argue that asymptotic optimality of HCMM is derived similar to the shifted exponential case and further show that HCMM provides unbounded gain over uncoded schemes, asymptotically.

A random variable T has Weibull distribution with shape parameter $\alpha > 0$ and scale parameter $\mu > 0$, denoted by $T \sim \mathcal{W}(\alpha, \mu)$, if the CDF of T is of the following form:

$$\Pr[T \leq t] = 1 - e^{-(\mu t)^{\alpha}}, \quad t \geq 0.$$

The expected value of the Weibull distribution is known to be $\mathbb{E}[T] = \frac{1}{\mu}\Gamma(1+1/\alpha)$, where $\Gamma(\cdot)$ denotes the Gamma function.

As stated in Section II-B, we consider a 3-parameter shifted Weibull distribution for workers' run-times defined in (2). The mean value of the worker i's run-times is then $\mathbb{E}[T_i] = a_i\ell_i + \frac{\ell_i}{\mu_i}\Gamma(1 + 1/\alpha_i)$. Clearly, shifted exponential distribution is a special case of the shifted Weibull model when $\alpha_i = 1$. By slight reparameterizations, this model can be similarly applied to the HCMM algorithm proposed in Algorithm 1, meaning that the main and alternative optimization problems defined in (3), (4) and (5) can be similarly analyzed under the shifted Weibull model.

As in the exponential case, we begin by maximizing the expected aggregate return at the master node ($\mathcal{P}_{\text{alt}}^{(1)}$) under the shifted Weibull distribution, which is given by

$$\mathbb{E}\left[X(t)\right] = \sum_{i=1}^{n} \mathbb{E}\left[X_i(t)\right] = \sum_{i=1}^{n} \ell_i \left(1 - e^{-\left(\frac{\mu_i}{\ell_i}(t - a_i\ell_i)\right)^{\alpha_i}}\right).$$

The optimal load allocation that maximizes the individual expected aggregate returns at each worker (and thus the total aggregate return) can be found by solving the following equation:

$$\begin{aligned}&\frac{\partial}{\partial \ell_i}\mathbb{E}\left[X_i(t)\right] \\ &\quad = 1 - e^{-\left(\frac{\mu_i}{\ell_i}(t - a_i\ell_i)\right)^{\alpha_i}} \left(1 + \frac{{\mu_i}^{\alpha_i}\alpha_i t}{\ell_i}\left(\frac{t}{\ell_i} - a_i\right)^{\alpha_i - 1}\right) \\ &\quad = 0.\end{aligned} \tag{25}$$

Solving (25) for ℓ_i yields $\ell_i^*(t) = \frac{t}{\lambda_i}$ where the constant $\lambda_i > a_i$ is the positive solution to

$$e^{{\mu_i}^{\alpha_i}(\lambda_i - a_i)^{\alpha_i}} = 1 + \alpha_i {\mu_i}^{\alpha_i}\lambda_i(\lambda_i - a_i)^{\alpha_i - 1}.$$

Similar to Section III, we can define s as follows,

$$
\begin{aligned}
s &= \frac{\mathbb{E}\left[X^*(t)\right]}{t} \\
&= \frac{1}{t}\sum_{i=1}^{n} \ell_i^*(t)\left(1 - e^{-\left(\frac{\mu_i}{\ell_i^*(t)}(t - a_i\ell_i^*(t))\right)^{\alpha_i}}\right) \\
&= \sum_{i=1}^{n} \frac{1}{\lambda_i}\left(1 - e^{-\left(\mu_i\lambda_i\left(1-\frac{a_i}{\lambda_i}\right)\right)^{\alpha_i}}\right) \\
&= \sum_{i=1}^{n} \frac{\alpha_i {\mu_i}^{\alpha_i}(\lambda_i - a_i)^{\alpha_i - 1}}{1 + \alpha_i {\mu_i}^{\alpha_i}\lambda_i(\lambda_i - a_i)^{\alpha_i - 1}} \\
&= \Theta(n).
\end{aligned}
$$

The last equality uses the fact that all the distribution parameters are constants. The expected aggregate return with optimal loads, $\mathbb{E}\left[X^*(t)\right]$, equals to r at time $t = \tau^*$. Thus, $\tau^* = \frac{r}{s} = \Theta(1)$ and $\ell_i^*(\tau^*) = \frac{\tau^*}{\lambda_i} = \frac{r}{s\lambda_i} = \Theta(1)$.

Proof of Theorem 3. With the aforementioned reparametrizations of λ_i, s and τ^*, the HCMM algorithm defined in Algorithm 1 is identically applicable to the Weibull model. Proof of the asymptotic optimality of HCMM under the Weibull distribution follows the similar steps as in the proof for the exponential case in Section III-C (unless specifically justified, e.g. (17)). We avoid rewriting these steps for the purpose of readability of the paper, but we note that the concentration inequalities used to establish the proof of Theorem 1 can be applied to a wide class of distributions including the Weibull distribution. □

As an implication of Theorem 3, the induced expected execution time by HCMM algorithm is asymptotically constant, that is $\mathbb{E}[T_{\mathsf{HCMM}}] = \Theta(1)$; which was also the case for shifted exponential distribution. To compare with the uncoded scenario, we start by the following lemma which characterizes the extreme value of a sequence of Weibull random variables.

Lemma 2. *Let $\{T_i\}_{i=1}^{\infty}$ be a sequence of i.i.d. $\mathcal{W}(\alpha, \mu)$ random variables and $T_n^* = \max_{i\in[n]} T_i$ denote the maximum of the first n variables. Then,*

$$
\mathbb{E}\left[T_n^*\right] \geq \Theta\left((\log n)^{1/\alpha}\right).
$$

Proof. Consider the sequence of maximums $\{T_i^*\}_{i=1}^{\infty}$. From Markov's inequality, we have $\frac{\mathbb{E}[T_n^*]}{t_n} \geq \Pr[T_n^* \geq t_n]$, for any $t_n > 0$ and $n \in \mathbb{N}$. Pick $t_n = \frac{1}{\mu}\big(\log n\big)^{1/\alpha}$. Therefore,

$$
\begin{aligned}
\frac{\mathbb{E}[T_n^*]}{\frac{1}{\mu}\big(\log n\big)^{1/\alpha}} &\geq \Pr\left[T_n^* \geq \frac{1}{\mu}\big(\log n\big)^{1/\alpha}\right] \\
&= 1 - \Pr\left[T_n^* < \frac{1}{\mu}\big(\log n\big)^{1/\alpha}\right] \\
&= 1 - \prod_{i=1}^{n} \Pr\left[T_i < \frac{1}{\mu}\big(\log n\big)^{1/\alpha}\right] \\
&= 1 - \left(1 - e^{-\log n}\right)^n \\
&= 1 - \left(1 - \frac{1}{n}\right)^n.
\end{aligned}
$$

Therefore,

$$
\lim_{n\to\infty} \frac{\mathbb{E}[T_n^*]}{\frac{1}{\mu}\big(\log n\big)^{1/\alpha}} \geq \lim_{n\to\infty} 1 - \left(1 - \frac{1}{n}\right)^n = 1 - \frac{1}{e} > 0.63,
$$

which implies $\mathbb{E}[T_n^*] \geq \Theta\left((\log n)^{1/\alpha}\right)$. □

Now we complete the proof of Theorem 4.

Proof of Theorem 4. Recall that T_{UC} denotes the completion time of the optimum uncoded distributed matrix multiplication algorithm across n workers parametrized by tuples $\{(a_i, \mu_i, \alpha_i,)\}_{i=1}^{n}$. To bound the mean of T_{UC}, assume that every machine is replaced with a stochastically faster machine with parameters $(\tilde{a}, \tilde{\mu}, \tilde{\alpha})$ where $\tilde{a} = \min_i a_i$, $\tilde{\mu} = \max_i \mu_i$ and $\tilde{\alpha} = \max_i \alpha_i$, i.e., the expected run-time of the latter scenario is no greater than that of the former one. For the new set of n identical machines, the optimal loading is uniform, i.e., $\widetilde{\ell}_i^* = \frac{r}{n}$. Let $\{\widetilde{T}_i\}_{i=1}^{n}$ denote the i.i.d. shifted Weibull run times for new set of machines which have CDFs of the form

$$
\begin{aligned}
\Pr\left[\widetilde{T}_i \leq t\right] &= 1 - e^{-\left(\frac{\tilde{\mu}}{\widetilde{\ell}_i^*}\left(t - \tilde{a}\widetilde{\ell}_i^*\right)\right)^{\tilde{\alpha}}} \\
&= 1 - e^{-\left(\tilde{\mu}\frac{n}{r}\left(t - \tilde{a}\frac{r}{n}\right)\right)^{\tilde{\alpha}}},
\end{aligned}
$$

for $t \geq \frac{\tilde{a}r}{n}$. The mean of computation time for the new set of machines is

$$
\mathbb{E}\left[\widetilde{T}_{\mathsf{UC}}\right] = \mathbb{E}\left[\max_{i\in[n]} \widetilde{T}_i\right] = \frac{\tilde{a}r}{n} + \mathbb{E}\left[\max_{i\in[n]} \widetilde{\widetilde{T}}_i\right],
$$

where $\widetilde{\widetilde{T}}_i = \widetilde{T}_i - \frac{\tilde{a}r}{n}$ are i.i.d. $\mathcal{W}(\tilde{\alpha}, \tilde{\mu}\frac{n}{r})$ for all workers $i \in [n]$. Using Lemma 2, we can write

$$
\mathbb{E}[T_{\mathsf{UC}}] \geq \mathbb{E}\left[\widetilde{T}_{\mathsf{UC}}\right] \geq \frac{\tilde{a}r}{n} + \Theta\left((\log n)^{1/\tilde{\alpha}}\right) = \Theta\left((\log n)^{1/\tilde{\alpha}}\right).
$$

Comparing the best uncoded scheme with the proposed coded algorithm demonstrates that HCMM outperforms the best uncoded scheme by a factor of at least $\Theta\left((\log n)^{1/\tilde{\alpha}}\right)$, i.e.,

$$
\frac{\mathbb{E}[T_{\mathsf{UC}}]}{\mathbb{E}[T_{\mathsf{HCMM}}]} \geq \Theta\left((\log n)^{1/\tilde{\alpha}}\right).
$$

□

V. Numerical Studies and Experiments using Amazon EC2 Machines

In this section, we present our results both from simulations as well as from experiments over Amazon EC2 clusters. These results demonstrate how HCMM can provide significant speedups in comparison to state-of-the-art load allocation schemes.

A. Numerical Analysis

We now present numerical results evaluating the performance of HCMM. We consider both the shifted exponential model in (1) and the shifted Weibull model in (2) for run-time distributions in our simulations, assuming the unit seconds per row (s/row) for a and $1/\mu$. The underlying computation task is to compute $r = 10000$ inner products using a heterogeneous cluster of $n = 100$ workers, where different scenarios for

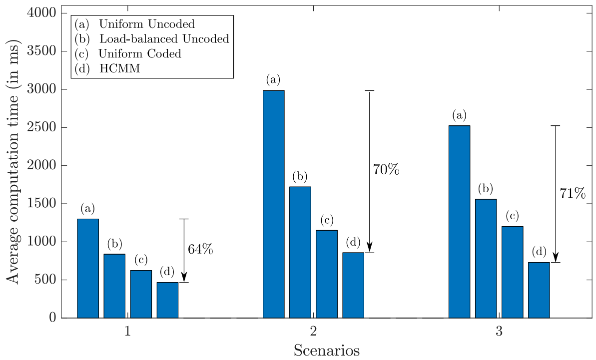


Fig. 2: Illustration of the performance gain of HCMM over the three benchmark schemes for the exponential run-time model. Among the three scenarios, HCMM achieves a performance improvement of up to 71% over Uniform Uncoded, up to 53% over Load-balanced Uncoded, and up to 39% over Uniform Coded. Furthermore, the coding redundancy $\sum_{i=1}^{n} \ell_i/r$ for the three scenarios is in the range of $1.41 - 1.46$ for HCMM and in the range of $2.3 - 2.8$ for Uniform Coded. This demonstrates the efficient utilization of resources by HCMM.

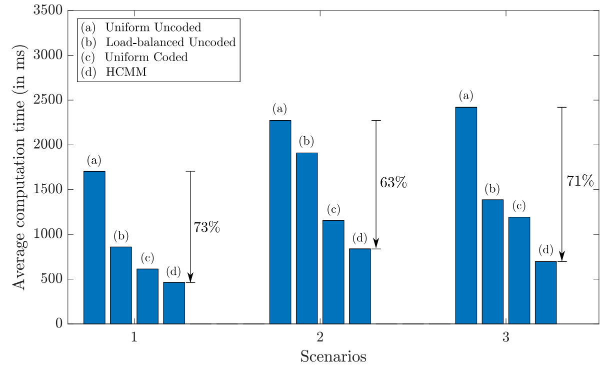


Fig. 3: Illustration of the performance gain of HCMM over the three benchmark schemes for Weibull model for run-time. Among the three scenarios, HCMM achieves a performance improvement of up to 73% over Uniform Uncoded, up to 56% over Load-balanced Uncoded, and up to 42% over Uniform Coded. Furthermore, the coding redundancy $\sum_{i=1}^{n} \ell_i/r$ for the three scenarios is in the range of $1.30 - 1.42$ for HCMM and in the range of $2.0 - 2.5$ for Uniform Coded. This demonstrates the efficient utilization of resources by HCMM.

heterogeneity are considered. For each scenario under consideration, we implement the following load allocation schemes[5]:

1) **Uniform Uncoded**: Each worker is assigned an equal number of rows, i.e., $\ell_i = r/n$ for all workers i.
2) **Load-balanced Uncoded**: Each worker is assigned a load which is inversely proportional to its expected time for computing one inner product, i.e., for the shifted exponential model, $\ell_i \propto \mu_i/(a_i\mu_i + 1)$, while for the shifted Weibull model, $\ell_i \propto \mu_i/(a_i\mu_i + \Gamma(1 + 1/\alpha_i))$ for all workers i. Furthermore, we set $\sum_{i=1}^{n} \ell_i = r$.
3) **Uniform Coded**: Equal number of coded rows are assigned to each worker. Redundancy is numerically optimized for minimizing the average computation time for receiving results of at least r inner products at the master node.
4) **HCMM**: Each worker is assigned the asymptotically optimal load allocation derived in Section III-B, i.e., $\ell_i = \tau^*/\lambda_i$ for each worker i according to (8) and (12).

For simulations under the shifted exponential model, we consider the following three scenarios:

- **Scenario 1 (2-mode heterogeneity)**: $(a_i, \mu_i) = (1, 1)$ for 50 workers, and $(a_i, \mu_i) = (4, 0.5)$ for the other 50 workers.
- **Scenario 2 (3-mode heterogeneity)**: $(a_i, \mu_i) = (1, 0.5)$ for 25 workers, $(a_i, \mu_i) = (4, 2)$ for 25 workers, and $(a_i, \mu_i) = (12, 0.25)$ for the remaining 50 workers.
- **Scenario 3 (Random heterogeneity)**: For each worker i, parameters a_i and μ_i are sampled from the sets $\{1, 4, 12\}$, $\{0.5, 2, 0.25\}$, respectively and all uniformly at random.

The following three scenarios are considered for simulations under the shifted Weibull distribution for run-times:

[5]For each scheme, the load number for each worker is approximated to the nearest larger integer using the `ceil()` function. For the practical large load regime considered in simulations, this rounding step has negligible impact on load allocation and on the overall results.

- **Scenario 1 (2-mode heterogeneity)**: $(a_i, \mu_i, \alpha_i) = (1, 1, 1.2)$ for 50 workers, and $(a_i, \mu_i, \alpha_i) = (4, 0.5, 0.8)$ for the other 50 workers.
- **Scenario 2 (3-mode heterogeneity)**: $(a_i, \mu_i, \alpha_i) = (1, 0.5, 0.9)$ for 25 workers, $(a_i, \mu_i, \alpha_i) = (4, 2, 1.2)$ for 25 workers, and $(a_i, \mu_i, \alpha_i) = (12, 0.25, 1.5)$ for the remaining 50 workers.
- **Scenario 3 (Random heterogeneity)**: For each worker i, parameters a_i, μ_i and α_i are sampled from the sets $\{1, 4, 12\}$, $\{0.5, 2, 0.25\}$ and $\{0.9, 1.2, 1.5\}$, respectively and all uniformly at random.

Fig. 2 and 3 illustrate the performance comparison of the four schemes for the two run-time models. We make the following conclusions from the results.

- HCMM significantly outperforms the benchmark load allocation schemes. In particular, for the shifted exponential model, HCMM provides speedups of up to 71% over Uniform Uncoded, up to 53% over Load-balanced Uncoded, and up to 39% over Uniform Coded, among the three scenarios. When the machine run-time is assumed to have a shifted Weibull distribution, among the three scenarios HCMM results in gains of up to 73%, 56% and 42% over Uniform Uncoded, Load-balanced Uncoded, and Uniform Coded respectively.
- The coding redundancy $\sum_{i=1}^{n} \ell_i/r$ for Uniform Coded is higher in comparison to the one for HCMM. In particular, for simulations under the shifted exponential model, the coding redundancy for the three scenarios is in the range of $2.3 - 2.8$ for Uniform Coded and in the range of $1.41 - 1.46$ for HCMM. For simulations under the shifted Weibull distribution, the coding redundancy is in the range of $2.0 - 2.5$ for Uniform Coded, while for HCMM, it is in the range $1.30 - 1.42$. This demonstrates that HCMM leads to a better utilization of computing resources.
- Both Load-balanced Uncoded and Uniform Coded im-

prove upon the performance of Uniform Uncoded. In Load-balanced Uncoded scheme, assigning larger loads to faster machines leads to better performance, while for Uniform Coded, repeated computations lead to better performance as the master does not need to wait for all the results. HCMM provides the best expected execution time among the four schemes as it combines the gains of Load-balanced Uncoded and Uniform Coded by employing efficient load balancing along with minimal number of redundant computations.

Next, we present the results from our experiments over Amazon EC2 clusters. These results show agreement with our numerical studies.

B. Experiments using Amazon EC2 machines

We use Python with `mpi4py` package [35] to implement our developed HCMM scheme over Amazon EC2 clusters. To emulate the straggler effects in large-scale systems [36], we inject artificial delays.[6] This is achieved by selecting some workers to be stragglers at the beginning of experiments and slowing down each such worker by making it wait for 3 times the amount of time it spends in computation before it sends its results to the master. This is done using the `sleep()` function in `time` package. For each scenario, the choice of stragglers is made by drawing a sample from the Bernoulli(0.5) distribution for each worker, i.e., each worker is chosen to be a straggler with probability 0.5.

In line with our simulation studies, we compare the performance of HCMM with the three benchmark load allocation schemes. For Load-balanced Uncoded, the number of uncoded rows ℓ_i assigned to a worker i is proportional to the number of virtual CPUs, and the loads are normalized to have a sum equal to r. For the encoding and the decoding steps for Uniform Coded as well as HCMM, we utilize the Luby transform (LT) codes with peeling decoder which provides nearly linear decoding complexity [38]. Utilization of LT codes for distributed computing is proposed in [39] as well. However, they perform a homogeneous load allocation by assigning an equal number of rows of the encoded data matrix to each worker and hence do not capture the heterogeneity of the computing cluster in distributing the encoded data matrix. Towards this end, we relax our goal of recovering all the inner products from any r of the coded inner products to recovering all the inner products from any $r' = r(1+\epsilon)$ coded inner products with high probability. Ideally, we would like to have $\epsilon > 0$ to be as small as possible. In our experiments, we keep $r = 10000$, and based on the results in [39], we use the robust Soliton degree distribution with $(c, \delta) = (0.03, 0.1)$ and select $\epsilon = 0.13$, where c is a tuning parameter and δ is a bound on the probability of failure of decoding from a certain number of received coded inner products (see [39] for details). Therefore, for both HCMM and Uniform Coded, we design the load allocation such that the master needs to wait only for

[6]Artificial delays are injected since stragglers are rarely observed in small clusters in Amazon EC2. Though other emerging platforms such as federated learning, computation with deadline, mobile edge computing, fog computing, etc., still suffer from stragglers where our ideas can be employed [37].

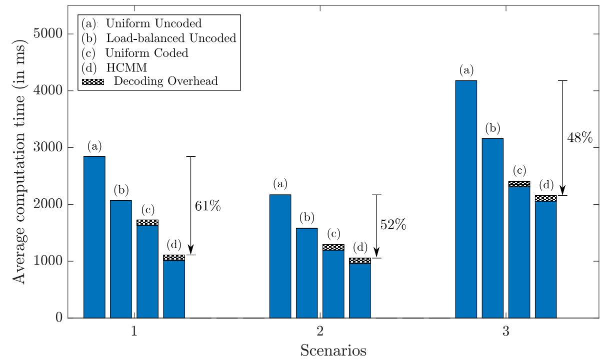


Fig. 4: Illustration of the performance gain of HCMM over the three benchmark schemes. Among the three scenarios, HCMM achieves a performance improvement of up to 61% over Uniform Uncoded, up to 46% over Load-balanced Uncoded, and up to 36% over Uniform Coded. Furthermore, the coding redundancy $\sum_{i=1}^{n} \ell_i/r$ for the three scenarios is approximately 1.4 for HCMM and in the range of $2.12 - 2.26$ for Uniform Coded. Therefore, HCMM gives the best overall execution time among the four scenarios with minimal coding overhead.

$r' = 11300$ coded inner products. The total computation time is equal to the waiting time for $r' = 11300$ results plus the average time for decoding the $r = 10000$ inner products from the received $r' = 11300$ coded inner products.[7] For HCMM, we use the shifted exponential distribution for estimating the computation model for each worker.

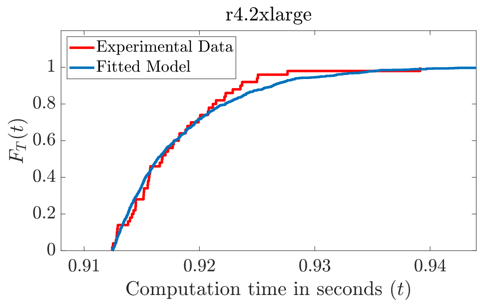
For performance comparison of the four schemes, we consider the following three computing scenarios:

- **Scenario 1**: Each row has 500000 elements. We use a heterogeneous cluster of 11 machines – one master of instance type **m4.xlarge**, four workers of instance type **r4.2xlarge**, and six workers of instance type **r4.xlarge**.
- **Scenario 2**: Each row has 500000 elements. We use a heterogeneous cluster of 16 machines – one master of instance type **m4.xlarge**, six workers of instance type **r4.2xlarge**, and nine workers of instance type **r4.xlarge**.
- **Scenario 3**: Each row has 1000000 elements. We use the same heterogeneous cluster as in the previous scenario.

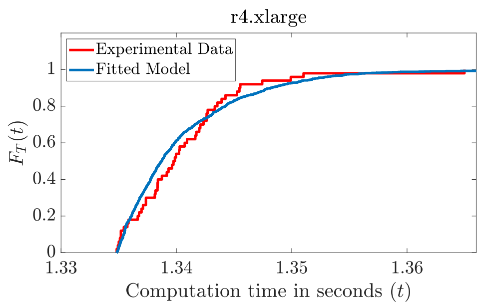
Fig. 4 provides a performance comparison of HCMM with the benchmark load allocation schemes for the three scenarios, where the decoding time is taken into account as well. Fig. 5 presents the typical cumulative distribution functions for the instances used in the experiments. We make the following conclusions from the results:

- As demonstrated in Fig. 5, the shifted exponential model is a good first order fit for the run-times of the workers.
- HCMM achieves significant speedups over the benchmark load allocation policies. In particular, HCMM combined with LT codes provides gains in the overall execution time of up to 61% over Uniform Uncoded, up to 46% over Load-balanced Uncoded, and up to 36% over Uniform Coded.

[7]The average time for decoding $r = 10000$ inner products from any $r(1+\epsilon)$ coded inner products is obtained using a **m4.xlarge** instance.



(a) $(a, 1/\mu) = (1.37 \times 10^{-3}\text{s/row}, 8.25 \times 10^{-6}\text{s/row})$



(b) $(a, 1/\mu) = (2.00 \times 10^{-3}\text{s/row}, 8.72 \times 10^{-6}\text{s/row})$

Fig. 5: Typical empirical cumulative distribution functions for two instances used in Scenario 3 of our experiments. The measurements were taken in the absence of any manual delay. As demonstrated here, shifted exponential distribution is a good model for the task execution time in EC2 machines.

TABLE I: Total computation load ($\sum_{i=1}^{n} \ell_i$) of HCMM and Uniform Coded

Scenario	n	HCMM	Uniform Coded
1	10	11397	22600
2	15	11402	21201
3	15	11403	21201

- As presented in Table I, HCMM has significantly lower total computation load compared to Uniform Coded. Hence, HCMM leads to efficient utilization of the computing resources, combining the benefits of both Load-balanced Uncoded and Uniform Coded schemes.

These results demonstrate that HCMM can provide significant speedups in large-scale computing environments.

VI. Generalization to Computing Scenarios under Budget Constraints

In this section, we consider the optimization problem in (3) under the shifted exponential distribution with a monetary constraint for carrying out the overall computation. Running computation tasks on a commodity server costs depending on several factors including CPU, memory, ECU, storage, bandwidth, etc. Different cloud computing platforms employ different pricing policies, and these need to be taken into account for developing efficient task allocation and execution algorithms [40]–[44]. For example, Table II summarizes the cost per hour of using Amazon EC2 clusters with different

TABLE II: Amazon EC2 Pricing for Linux

machine	vCPU	ECU	Memory (GiB)	Instance Storage (GB)	price (/Hour)
m3.medium	1	3	3.75	1×4 SSD	\$0.077
m3.large	2	6.5	7.5	1×32 SSD	\$0.154
m3.xlarge	4	13	15	2×40 SSD	\$0.308
m3.2xlarge	8	26	30	2×80 SSD	\$0.616

parameters (at the time of writing this manuscript) [45]. In this section, we take into account the monetary constraint in the optimization problem in (3) and provide a heuristic algorithm towards finding the optimal load allocation under cost budget constraint.

We now present the precise problem formulation we are interested in. For a computation task and a given set of N machines, the goal is to minimize the expected run-time while satisfying the budget constraint C, that is

$$\begin{aligned} \mathcal{P}_{\text{main-constrained}}: \quad & \underset{\ell}{\text{minimize}} \quad \mathbb{E}[T_{\mathsf{CMP}}] \\ & \text{subject to} \quad \sum_{i=1}^{N} c_i \mathbb{1}_{\{\ell_i > 0\}} \mathbb{E}[T_{\mathsf{CMP}}] \leq C, \end{aligned} \tag{26}$$

where c_i represents the cost per time unit of using machine $i \in [N]$. According to the pricing polices provided by AWS, e.g. Table II, a linear model for cost (versus performance parameters) is intuitive and convincing. Considering the last two rows of Table II for instance, doubling the parameters results in doubled cost. To be general, we model the computation cost of a single machine as $c = \kappa\mu^{\gamma}$ per unit of time, which captures a convex dependency of the speed parameter μ for constants $\gamma \geq 1$ and $\kappa > 0$.

We assume that there are K types of machines parameterized with $\{(a_k, \mu_k)\}_{k=1}^{K}$, and $N_k, k \in [K]$ of each type is available to run a distributed computation task, where $N = \sum_{k=1}^{K} N_k$ is the total number of available machines. We also assume that $\mu_1 \leq \cdots \leq \mu_K$ and $a_1\mu_1 = \cdots = a_K\mu_K = \xi$ for a constant ξ.[8] As we showed in Theorem 1, HCMM is asymptotically optimal (i.e. optimal within a vanishing deviation) regarding the average run-time. In this section, we also consider the asymptotic regime, i.e. for large enough number of machines and hence HCMM attains the optimality per $\mathcal{P}_{\text{main}}$ in (3).

The following lemma states a useful observation regarding the solutions to the constrained problem $\mathcal{P}_{\text{main-constrained}}$ and the minimum possible cost for carrying out a computation task.

Lemma 3. *HCMM is the (asymptotic) solution to the feasible $\mathcal{P}_{\text{main-constrained}}$. Moreover, given a computation task and a set of machines, decreasing the number of fastest (slowest) machines in HCMM, results in smaller (greater) expected cost. And, the minimum (maximum) cost of HCMM is induced by running the task only on any number of the slowest (fastest) machines.*

[8]The latter assumption can be intuitively justified as follows. If a machine is c times more powerful than another machine, as the first order estimation, one can assume that both the shift (a_k) and the straggling parameter (μ_k) of the computation are c times stronger.

Proof. We first argue that if the budget-constrained problem defined in $\mathcal{P}_{\text{main-constrained}}$ is feasible, then HCMM determines the asymptotically optimal load allocation. Consider a set of N machines and assume that M of them are assigned non-zero loads in the optimal budget-constrained scheme. Now, one can run HCMM load allocation over the set of these M machines and according to asymptotic optimality results, HCMM asymptotically attains the optimal run-time while satisfying the budget constraint.

Now assume that n_k number of type $k \in [K]$ machine is used. Then, by assigning the loads obtained from HCMM and the result of Theorem 1, the induced expected cost (for large number of machines) can be written as

$$\begin{aligned}\operatorname{cost}\big(\text{HCMM}(n_1, \cdots, n_K)\big) &= \tau^* \sum_{k=1}^{K} n_k c_k \\ &= \frac{r}{s} \sum_{k=1}^{K} n_k c_k \\ &= \frac{r}{\sum_{k=1}^{K} \frac{n_k \mu_k}{1+\mu_k \lambda_k}} \sum_{k=1}^{K} n_k \kappa \mu_k^{\gamma} \\ &= \kappa r x_\xi \frac{\sum_{k=1}^{K} n_k \mu_k^{\gamma}}{\sum_{k=1}^{K} n_k \mu_k}, \end{aligned} \tag{27}$$

where $x_\xi = 1+\mu_k\lambda_k$ is the solution to the equation $e^{x_\xi - \xi - 1} = x_\xi$ for all machine type $k \in [K]$. In another scenario, assume that we remove one machine of type K (the fastest machine type) and run HCMM accordingly, i.e. n_k of type $k \in [K-1]$ and $n_K - 1$ of type K. The expected cost of this scenario can be written as follows:

$$\begin{aligned}&\operatorname{cost}\big(\text{HCMM}(n_1, \cdots, n_K - 1)\big) \\ &\quad = \kappa r x_\xi \frac{\sum_{k=1}^{K-1} n_k \mu_k^{\gamma} + (n_K - 1)\mu_K^{\gamma}}{\sum_{k=1}^{K-1} n_k \mu_k + (n_K - 1)\mu_K} \\ &\quad \overset{(f)}{\leq} \kappa r x_\xi \frac{\sum_{k=1}^{K} n_k \mu_k^{\gamma}}{\sum_{k=1}^{K} n_k \mu_k} \\ &\quad = \operatorname{cost}\big(\text{HCMM}(n_1, \cdots, n_K)\big), \end{aligned} \tag{28}$$

where inequality (f) can be easily verified given that $\mu_1 \leq \cdots \leq \mu_K$. We can iteratively apply the same argument and conclude that the minimum expected cost is achieved when only the slowest machines are used, that is

$$C_{\text{min}} \coloneqq \operatorname{cost}\big(\text{HCMM}(n_1, 0, \cdots, 0)\big) = \kappa r x_\xi \mu_1^{\gamma - 1}, \tag{29}$$

for any $1 \leq n_1 \leq N_1$. Similar to (28), one can show that reducing the number of participating slowest machines increases the induced expected cost of HCMM, that is

$$\begin{aligned}&\operatorname{cost}\big(\text{HCMM}(n_1 - 1, \cdots, n_K)\big) \\ &\qquad \geq \operatorname{cost}\big(\text{HCMM}(n_1, \cdots, n_K)\big). \end{aligned} \tag{30}$$

Therefore, applying (30) iteratively shows that the maximum expected cost occurs when only the fastest machines are employed, that is

$$C_{\text{max}} \coloneqq \operatorname{cost}\big(\text{HCMM}(0, \cdots, 0, n_K)\big) = \kappa r x_\xi \mu_K^{\gamma - 1},$$

for any $1 \leq n_K \leq N_K$. □

Lemma 3 implies that if the available budget C is less than C_{min} defined in (29), then $\mathcal{P}_{\text{main-constrained}}$ is infeasible and it is impossible to run the task on the given set of machines while satisfying the budget constraint. Moreover, reducing one machine from the available set of fastest machines along with HCMM results in a lower expected cost; and reducing the number of participating slowest machines results in a larger expected cost.

Now that HCMM asymptotically solves the feasible budget-constrained problem in (26), i.e. for $C \geq C_{\text{min}}$, finding the optimal number of machines of each type to use in HCMM requires combinatorial search over all possible allocations. However, as Lemma 3 suggests, using faster machines induces a larger cost. Further, the computation time increases if we decrease the number of machines. This is the motivation behind our heuristic algorithm for an efficient search to find the number of machines of each type to include in HCMM, which we describe next.

Algorithm 2 Heuristic Search

procedure HEURISTIC SEARCH
    $(n_1, \cdots, n_K) \leftarrow (N_1, \cdots, N_K)$
top:
    Run HCMM with $(n_1, \cdots, n_K)$
    if $\operatorname{cost}\big(\text{HCMM}(n_1, \cdots, n_K)\big) > C$ **then**
        $n_j \leftarrow n_j - 1$ where $j = \max\{k : n_k > 0\}$
        goto *top*
    else
        return $(n_1, \cdots, n_K)$

First, Algorithm 2 runs HCMM algorithm using all machines, i.e., $n_k = N_k$ for each $k \in [K]$. Then, it calculates the corresponding cost according to (27). If the cost is $> C$, it starts to decrease the number of available fastest machines, i.e. $n_K \leftarrow n_K - 1$, and runs HCMM again. While the cost is $> C$, the algorithm keeps decreasing the number of used fast machines till $n_K = 0$. Then, the algorithm sets $n_K = 0$ and starts decreasing n_{K-1} and so on, until a feasible cost is achieved. Thus, the algorithm returns $(N_1, \cdots, N_j, n_{j+1}, 0, \cdots, 0)$ which is the first tuple that satisfies the cost constraint. Therefore, the search space complexity of the heuristic is $\mathcal{O}(N_1 + \cdots + N_K) = \mathcal{O}(N)$ which is more efficient than the exhaustive search where the complexity is $\mathcal{O}(N_1 \cdots N_K)$. The pseudo-code in Algorithm 2 summarizes the heuristic.

Example. In this example, we consider two different scenarios to demonstrate the application of the proposed heuristic search algorithm. For the cost model, we assume $\gamma = 2$ and $\kappa = 1$, i.e. $c = \mu^2$. Further, we consider the task of computing $r = 100$ equations.

- **Scenario 1:** Two types of machines are available parameterized by $(a_1, \mu_1) = (0.5, 2)$ and $(a_2, \mu_2) = (0.25, 4)$, assuming 10 machines available of each type. Further, the available budget is $C = 860$. Using Lemma 3, the minimum and maximum induced costs are $C_{\text{min}} = 629.2$ and $C_{\text{max}} = 1258.4$. As $C \geq C_{\text{min}}$, there exists an HCMM load allocation which is asymptotically optimal per (26).

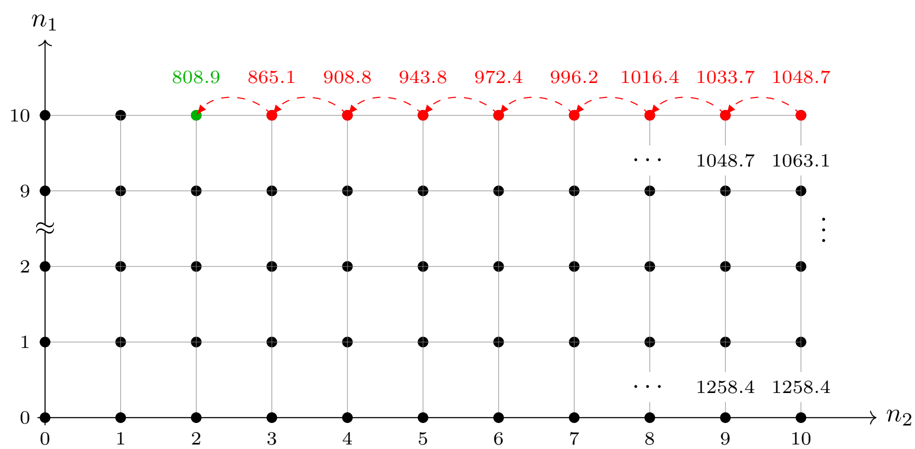


Fig. 6: Total cost associated with every pair of (n_1, n_2); $0 \leq n_1, n_2 \leq 10$.

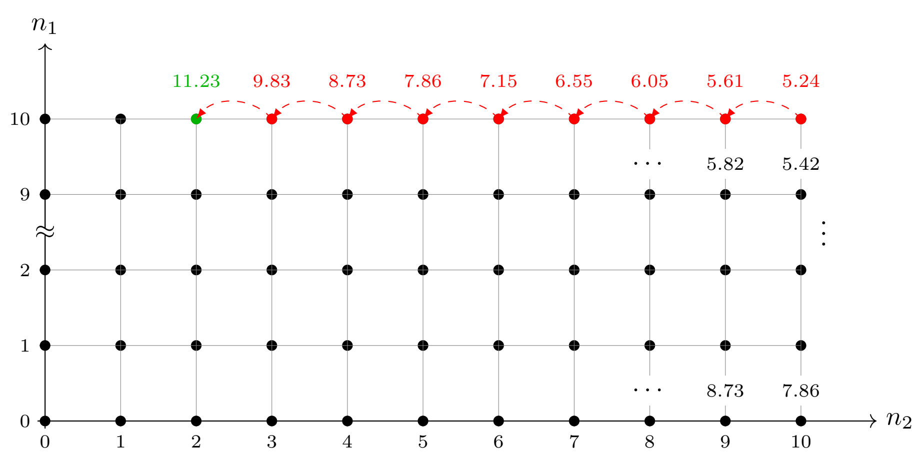


Fig. 7: Expected time associated with every pair of (n_1, n_2); $0 \leq n_1, n_2 \leq 10$.

Applying the proposed heuristic search, it takes 9 iterations (see Fig. 6 and 7) to arrive at the load allocation $(n_1, n_2) = (10, 2)$ which corresponds to the expected cost 808.9 and average execution time $\mathbb{E}[T_{\mathsf{HCMM}}] = 11.23$.

- **Scenario 2:** Three types of machines are available which are parameterized by $(a_1, \mu_1) = (1, 1)$, $(a_2, \mu_2) = (0.5, 2)$ and $(a_3, \mu_3) = (0.125, 8)$, assuming 10 machines of each type available. Further, the available budget is $C = 475$. Using Lemma 3, the minimum and maximum induced costs for the task of computing $r = 100$ equations are $C_{\text{min}} = 314.6$ and $C_{\text{max}} = 2516.8$ respectively. It takes 15 iterations for the proposed heuristic search algorithm to arrive at the tuple $(n_1, n_2, n_3) = (10, 6, 0)$. This corresponds to the expected cost 486.2 and the average time $\mathbb{E}[T_{\mathsf{HCMM}}] = 14.3$.

VII. Conclusion

In this paper, we proposed a coding framework for distributed matrix-vector multiplication in heterogeneous cloud computing environments. In particular, we considered two distributions for machines' run-times, i.e. shifted exponential and shifted Weibull and tackled the intractable problem of minimizing the average run-time of a computation task over all possible load allocations by proposing a tractable alternative formulation. The solution to the alternative problem established our proposed HCMM load allocation scheme which we proved to be asymptotically optimal. We also demonstrated the speedup of HCMM over three benchmark load allocation schemes and presented both the numerical and the experimental results. Experiments over Amazon EC2 clusters demonstrate that HCMM combined with LT codes and peeling decoders can provide significant gains in the average overall execution time. Moreover, we argued that HCMM is the asymptotically optimal allocation in budget-constrained scenarios as well, which led to providing a heuristic search in order to find a (sub)optimal load-machine assignment for a given set of machines while satisfying a pre-defined budget constraint.

VIII. Acknowledgment

This material is based upon work supported by Defense Advanced Research Projects Agency (DARPA) under Contract No. HR001117C0053, ARO award W911NF1810400, NSF grants CCF-1703575, ONR Award No. N00014-16-1- 2189, CCF-1763673, CCF-1755808 and the UC Office of President under grant No. LFR-18-548175. The views, opinions, and/or findings expressed are those of the author(s) and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government.

REFERENCES

[1] A. Reisizadeh, S. Prakash, R. Pedarsani, and S. Avestimehr, "Coded computation over heterogeneous clusters," in *IEEE International Symposium on Information Theory (ISIT), 2017*, pp. 2408–2412.

[2] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[3] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: cluster computing with working sets," *HotCloud*, vol. 10, pp. 10–10, 2010.

[4] M. Zaharia, A. Konwinski, A. D. Joseph, R. H. Katz, and I. Stoica, "Improving mapreduce performance in heterogeneous environments," in *OSDI*, vol. 8, p. 7, 2008.

[5] Q. Pu, G. Ananthanarayanan, P. Bodik, S. Kandula, A. Akella, P. Bahl, and I. Stoica, "Low latency geo-distributed data analytics," *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 4, pp. 421–434, 2015.

[6] S. Li, M. A. Maddah-Ali, and A. S. Avestimehr, "Coded MapReduce," in *Communication, Control, and Computing (Allerton), 2015 53rd Annual Allerton Conference on*, pp. 964–971, IEEE, 2015.

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

[8] S. Li, S. Supittayapornpong, M. A. Maddah-Ali, and S. Avestimehr, "Coded terasort," in *Parallel and Distributed Processing Symposium Workshops (IPDPSW), 2017 IEEE International*, pp. 389–398, IEEE, 2017.

[9] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*, pp. 13–16, ACM, 2012.

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

[11] L. Song, C. Fragouli, and T. Zhao, "A pliable index coding approach to data shuffling," *arXiv preprint arXiv:1701.05540*, 2017.

[12] M. Kiamari, C. Wang, and A. S. Avestimehr, "On heterogeneous coded distributed computing," in *GLOBECOM 2017-2017 IEEE Global Communications Conference*, pp. 1–7, IEEE, 2017.

[13] M. Attia and R. Tandon, "Information theoretic limits of data shuffling for distributed learning," *arXiv preprint arXiv:1609.05181*, 2016.

[14] Y. H. Ezzeldin, M. Karmoose, and C. Fragouli, "Communication vs distributed computation: an alternative trade-off curve," in *Information Theory Workshop (ITW), 2017 IEEE*, pp. 279–283, IEEE, 2017.

[15] S. Dutta, V. Cadambe, and P. Grover, "Short-dot: Computing large linear transforms distributedly using coded short dot products," in *Advances In Neural Information Processing Systems*, pp. 2092–2100, 2016.

[16] R. Tandon, Q. Lei, A. G. Dimakis, and N. Karampatziakis, "Gradient coding: avoiding stragglers in distributed learning," in *International Conference on Machine Learning*, pp. 3368–3376, 2017.

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

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

[19] K. Lee, C. Suh, and K. Ramchandran, "High-dimensional coded matrix multiplication," in *Information Theory (ISIT), 2017 IEEE International Symposium on*, pp. 2418–2422, IEEE, 2017.

[20] S. Wang, J. Liu, N. Shroff, and P. Yang, "Fundamental limits of coded linear transform," *arXiv preprint arXiv:1804.09791*, 2018.

[21] Q. Yu, M. A. Maddah-Ali, and A. S. Avestimehr, "Straggler mitigation in distributed matrix multiplication: Fundamental limits and optimal coding," *arXiv preprint arXiv:1801.07487*, 2018.

[22] A. Reisizadeh and R. Pedarsani, "Latency analysis of coded computation schemes over wireless networks," in *Communication, Control, and Computing (Allerton), 2017 55th Annual Allerton Conference on*, pp. 1256–1263, IEEE, 2017.

[23] K. Lee, R. Pedarsani, D. Papailiopoulos, and K. Ramchandran, "Coded computation for multicore setups," in *Information Theory (ISIT), 2017 IEEE International Symposium on*, pp. 2413–2417, IEEE, 2017.

[24] N. S. Ferdinand and S. C. Draper, "Anytime coding for distributed computation," in *Communication, Control, and Computing (Allerton), 2016 54th Annual Allerton Conference on*, pp. 954–960, IEEE, 2016.

[25] G. Suh, K. Lee, and C. Suh, "Matrix sparsification for coded matrix multiplication," in *Communication, Control, and Computing (Allerton), 2017 55th Annual Allerton Conference on*, pp. 1271–1278, IEEE, 2017.

[26] M. Aliasgari, J. Kliewer, and O. Simeone, "Coded computation against straggling decoders for network function virtualization," in *2018 IEEE International Symposium on Information Theory (ISIT)*, pp. 711–715, IEEE, 2018.

[27] Y. Yang, P. Grover, and S. Kar, "Computing linear transformations with unreliable components," *IEEE Transactions on Information Theory*, vol. 63, no. 6, pp. 3729–3756, 2017.

[28] C. Karakus, Y. Sun, S. Diggavi, and W. Yin, "Straggler mitigation in distributed optimization through data encoding," in *Advances in Neural Information Processing Systems*, pp. 5440–5448, 2017.

[29] A. Severinson, E. Rosnes, *et al.*, "Block-diagonal and LT codes for distributed computing with straggling servers," *arXiv preprint arXiv:1712.08230*, 2017.

[30] M. F. Aktas, P. Peng, and E. Soljanin, "Effective straggler mitigation: which clones should attack and when?," *ACM SIGMETRICS Performance Evaluation Review*, vol. 45, no. 2, pp. 12–14, 2017.

[31] D. Wang, G. Joshi, and G. Wornell, "Using straggler replication to reduce latency in large-scale parallel computing," *ACM SIGMETRICS Performance Evaluation Review*, vol. 43, no. 3, pp. 7–11, 2015.

[32] M. Rudelson and R. Vershynin, "Non-asymptotic theory of random matrices: extreme singular values," in *Proceedings of the International Congress of Mathematicians 2010 (ICM 2010) (In 4 Volumes) Vol. I: Plenary Lectures and Ceremonies Vols. II–IV: Invited Lectures*, pp. 1576–1602, World Scientific, 2010.

[33] G. Liang and U. C. Kozat, "Tofec: achieving optimal throughput-delay trade-off of cloud storage using erasure codes," in *IEEE INFOCOM 2014-IEEE Conference on Computer Communications*, pp. 826–834, IEEE, 2014.

[34] S. Dutta, V. Cadambe, and P. Grover, "Coded convolution for parallel and distributed computing within a deadline," in *Information Theory (ISIT), 2017 IEEE International Symposium on*, pp. 2403–2407, IEEE, 2017.

[35] L. Dalcín, R. Paz, and M. Storti, "MPI for Python," *Journal of Parallel and Distributed Computing*, vol. 65, no. 9, pp. 1108–1115, 2005.

[36] J. Dean and L. A. Barroso, "The tail at scale," *Communications of the ACM*, vol. 56, no. 2, pp. 74–80, 2013.

[37] S. Li, M. A. Maddah-Ali, and A. S. Avestimehr, "Coding for distributed fog computing," *IEEE Communications Magazine*, vol. 55, no. 4, pp. 34–40, 2017.

[38] D. J. MacKay and D. J. Mac Kay, *Information theory, inference and learning algorithms*. Cambridge university press, 2003.

[39] A. Mallick, M. Chaudhari, and G. Joshi, "Rateless codes for near-perfect load balancing in distributed matrix-vector multiplication," *arXiv preprint arXiv:1804.10331*, 2018.

[40] S. Su, J. Li, Q. Huang, X. Huang, K. Shuang, and J. Wang, "Cost-efficient task scheduling for executing large programs in the cloud," *Parallel Computing*, vol. 39, no. 4-5, pp. 177–188, 2013.

[41] E. Deelman, G. Singh, M. Livny, B. Berriman, and J. Good, "The cost of doing science on the cloud: the montage example," in *High Performance Computing, Networking, Storage and Analysis, 2008. SC 2008. International Conference for*, pp. 1–12, IEEE, 2008.

[42] D. Kondo, B. Javadi, P. Malecot, F. Cappello, and D. P. Anderson, "Cost-benefit analysis of cloud computing versus desktop grids.," in *IPDPS*, vol. 9, pp. 1–12, 2009.

[43] S. Yi, A. Andrzejak, and D. Kondo, "Monetary cost-aware checkpointing and migration on amazon cloud spot instances," *IEEE Transactions on Services Computing*, vol. 5, no. 4, pp. 512–524, 2012.

[44] M. Malawski, G. Juve, E. Deelman, and J. Nabrzyski, "Algorithms for cost-and deadline-constrained provisioning for scientific workflow ensembles in iaas clouds," *Future Generation Computer Systems*, vol. 48, pp. 1–18, 2015.

[45] [Online], "Amazon EC2 pricing." https://aws.amazon.com/ec2/pricing/. Accessed date: July 5th, 2017.

APPENDIX

McDiarmid's Inequality: Let $X_1, \cdots, X_n$ be independent random variables taking values in $\mathcal{X}$. Further, let the function

$f : \mathcal{X}^n \to \mathbb{R}$ be L_i-Lipschitz for all $i \in [n]$, that is

$$|f(x_1, \cdots, x_i, \cdots, x_n) - f(x_1, \cdots, x_i', \cdots, x_n)| \leq L_i,$$

for any $x_1, \cdots, x_n, x_i' \in \mathcal{X}$ and $i \in [n]$. Then, for any $\epsilon > 0$,

$$\Pr\Big[f(X_1, \cdots, X_n) - \mathbb{E}[f(X_1, \cdots, X_n)] \geq \epsilon\Big] \leq \exp\left(-\frac{2\epsilon^2}{\sum_{i=1}^n L_i^2}\right),$$

$$\Pr\Big[\mathbb{E}[f(X_1, \cdots, X_n)] - f(X_1, \cdots, X_n) \geq \epsilon\Big] \leq \exp\left(-\frac{2\epsilon^2}{\sum_{i=1}^n L_i^2}\right).$$

For each i, the aggregate return at time t satisfies $X_i(t) \in \{0, \ell_i\}$. Therefore, we can use McDiarmid's inequality as follows:

$$\Pr\left[X(t) - \mathbb{E}[X(t)] \geq \epsilon\right] \leq \exp\left(-\frac{2\epsilon^2}{\sum_{i=1}^n \ell_i^2}\right),$$

$$\Pr\left[\mathbb{E}[X(t)] - X(t) \geq \epsilon\right] \leq \exp\left(-\frac{2\epsilon^2}{\sum_{i=1}^n \ell_i^2}\right),$$

for any $\epsilon > 0$. Now, we proceed to the proof of Lemma 1.

Proof of Lemma 1. Let $t = \tau^* + \delta$ for some $\delta = \Theta\left(\frac{\log n}{\sqrt{n}}\right)$ and $\epsilon = \delta^2$. The claim is that $\Pr\left[X^*(t) \leq r - \epsilon\right] = o\left(\frac{1}{n}\right)$. From McDiarmid's inequality, we have

$$\begin{aligned}
\Pr\left[X^*(t) \leq r - \epsilon\right] &\leq \exp\left(-\frac{2\big(\mathbb{E}[X^*(t)] - r + \epsilon\big)^2}{\sum_i \ell_i^{*2}(t)}\right) \\
&= \exp\left(-\frac{2\big(ts - r + \epsilon\big)^2}{\sum_i \ell_i^{*2}(t)}\right) \\
&= \exp\left(-\frac{2\delta^2 s^2 + 2\delta^4 + 4\delta^3 s}{\left((\frac{r}{s})^2 + \delta^2 + 2\delta\frac{r}{s}\right)\sum_i \lambda_i^2}\right) \\
&\overset{(g)}{=} e^{-\Theta\left(n\delta^2\right)} = o\left(\frac{1}{n}\right).
\end{aligned}$$

In above, equality (g) follows from the fact that $r = \Theta(n)$, $s = \Theta(n)$, $\lambda_i = \Theta(1)$, $\delta = \Theta\left(\frac{\log n}{\sqrt{n}}\right)$, and therefore $\sum_i \lambda_i^2 = \Theta(n)$ and $s^2 = \Theta(n^2)$. Moreover, if $t^* < \tau^*$, with a positive probability there are less than r equations at the master node by time t^* which is a contradiction. Therefore,

$$\tau^* \leq t^* \leq \tau^* + \delta.$$

□

Amirhossein Reisizadeh received his B.S. degree form Sharif University of Technology, Tehran, Iran in 2014 and an M.S. degree from University of California, Los Angeles (UCLA) in 2016, both in Electrical Engineering. He is currently pursuing his Ph.D. in Electrical and Computer Engineering at University of California, Santa Barbara (UCSB). He is interested in using information and coding-theoretic concepts to develop fast and efficient algorithms for large-scale machine learning, distributed computing and optimization.

Saurav Prakash received his Bachelor of Technology degree in Electrical Engineering from the Indian Institute of Technology (IIT), Kanpur, India in 2016 and is currently pursuing his Ph.D. in Electrical and Computer Engineering from the University of Southern California (USC), Los Angeles. Saurav received the Annenberg Graduate Fellowship in 2016. He was one of the Viterbi-India fellows in summer 2015. His interests include information theory and data analytics with applications in large-scale machine learning and edge computing.

Ramtin Pedarsani is an Assistant Professor in ECE Department at the University of California, Santa Barbara. He received the B.Sc. degree in electrical engineering from the University of Tehran, Tehran, Iran, in 2009, the M.Sc. degree in communication systems from the Swiss Federal Institute of Technology (EPFL), Lausanne, Switzerland, in 2011, and his Ph.D. from the University of California, Berkeley, in 2015. His research interests include machine learning, information and coding theory, networks, and transportation systems. Ramtin is a recipient of the IEEE international conference on communications (ICC) best paper award in 2014.

A. Salman Avestimehr is a Professor at the Electrical and Computer Engineering Department of University of Southern California. He received his Ph.D. in 2008 and M.S. degree in 2005 in Electrical Engineering and Computer Science, both from the University of California, Berkeley. Prior to that, he obtained his B.S. in Electrical Engineering from Sharif University of Technology in 2003. He was an Assistant Professor at the ECE school of Cornell University from 2009 to 2013. He was also a postdoctoral scholar at the Center for the Mathematics of Information (CMI) at Caltech in 2008. His research interests include information theory, coding theory, and large-scale distributed computing and machine learning.

Dr. Avestimehr has received a number of awards for his research, including an Information Theory Society and Communication Society Joint Paper Award, a Presidential Early Career Award for Scientists and Engineers (PECASE) from the White House, a Young Investigator Program (YIP) award from the U. S. Air Force Office of Scientific Research, a National Science Foundation CAREER award, the David J. Sakrison Memorial Prize, and several Best Paper Awards at Conferences. He has been an Associate Editor for IEEE Transactions on Information Theory. He is currently a general Co-Chair of the 2020 International Symposium on Information Theory (ISIT).