Coded Computation over Heterogeneous Clusters

Amirhossein Reisizadeh*, Saurav Prakash†, Ramtin Pedarsani*, Amir Salman Avestimehr†
* University of California, Santa Barbara † University of Southern California

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. On the other hand, these systems enjoy abundance of redundancy – a vast number of computing nodes and large storage capacity. 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 with straggling servers 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. Moreover, if the number of worker nodes in the cluster is n, we show that HCMM is $\Theta(\log n)$ times faster than *any* uncoded scheme. We further provide numerical results demonstrating significant speedups of up to 49% and 34% for HCMM in comparison to the "uncoded" and "homogeneous coded" schemes, respectively. Additionally, we consider the problem of optimal load allocation subject to budget constraints, develop a heuristic algorithm for efficient load allocation, and demonstrate examples where the heuristic algorithm achieves the best allocation. Finally, we propose the use of LDPC codes instead of random linear codes, and describe how fast linear time decoding can be achieved by the use of LDPC codes.

I. Introduction

General distributed computing frameworks, such as MapReduce [1] and Spark [2], 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", which enjoy abundance of storage space and computing power, 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 (e.g., [3]), while partial data replication is also used to reduce the communication load in distributed computing (e.g., [4]). 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 [5], [6] 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. Hence, these codes can be utilized to pool the underutilized computing resources at network edge to slash the communication load of Fog computing [7].

The second coding concept introduced in [8] enables an inverse-linear tradeoff between computation load and computation latency (i.e., the overall job response time) for distributed matrix multiplication. 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, these can significantly speed up distributed computing applications. In a related work, the authors have proposed the use of redundant short dot products to speed up distributed computation of linear transforms [9]. In [10], the authors propose coding schemes for mitigating stragglers in distributed batch gradient computation.

However, the above coded computing approaches have been developed for homogeneous clusters. As discussed in [3], 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 in many engineering and scientific disciplines. In particular, it is a fundamental component of many popular machine learning algorithms such as logistic regression, reinforcement

This work is supported by a startup grant for Ramtin Pedarsani and NSF grants CCF-1408639, NETS-1419632, ONR award N000141612189, NSA grant, a research gift from Intel and by Defense Advanced Research Projects Agency (DARPA) under Contract No. HR001117C0053. 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.

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 multiplication, each worker node is assigned the task of computing inner products of the assigned coded rows with the input, where the assigned coded rows are random linear combinations of the rows of the original matrix. 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 multiplication. We are interested in finding the optimal load allocation that minimizes the expected 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 show that our proposed HCMM algorithm is provably asymptotically optimal. We also prove that HCMM is $\Theta(\log n)$ faster than the best uncoded scheme. Furthermore, in our numerical analysis, we demonstrate significant speedups of up to 49% and 34% for HCMM in comparison to the "uncoded" and "homogeneous coded" schemes, respectively.

We also provide two extensions of the HCMM algorithm. First, we consider the problem of load allocation under budget constraints, and develop a heuristic algorithm to find (sub)optimal load allocations. The heuristic is based on the lemma that for carrying out the computation task using a set of machines, the minimum (maximum) cost is induced by running the task only on any number of the slowest (fastest) machines. Moreover, this lemma determines whether a budget-constrained computation task is feasible given a set of machines. Second, we describe how linear time decoding can be achieved by using LDPC codes instead of random linear codes. Specifically, we demonstrate the use of iterative or *peeling* decoder for recovering the result of the matrix multiplication.

Notation. For a natural number n, $[n]$ represents the set $\{1, 2, \ldots, n\}$. For functions of n, we denote $f = \Theta(g)$ if there are positive constants c_1 and c_2 such that $c_1 \leq |f/g| \leq c_2$, and $f = \mathcal{O}(g)$ if there exists a positive constant c such that $|f/g| \leq c$. For functions f and g of n, $f = o(g)$ if $f/g \to 0$ as n goes to infinity.

II. Problem Setting

In this section, we describe our computation model, the network model and the precise problem formulation.

A. Computation Model

We consider the problem of matrix multiplication, in which given a matrix $\mathbf{A} \in \mathbb{R}^{r\times m}$ for some integers r, 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 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 and broadcasts it to the two worker nodes. These nodes will then compute $\mathbf{y}_1 = \mathbf{A}_1\mathbf{x}$ and $\mathbf{y}_2 = \mathbf{A}_2\mathbf{x}$ and return their results to the master node, which combines the results to obtain the intended outcome $\mathbf{y} = \mathbf{A}\mathbf{x}$. This example also illustrates an uncoded implementation of distributed computing, in which results from all the worker nodes are required to obtain the final result.

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

Definition 1. (Coded Distributed Computation) The coded distributed implementation of a computational task $f_{\mathbf{A}}(\cdot)$ is carried out in two steps:

- Local data blocks $\langle \mathbf{A}_i \rangle_{i=1}^n$ and local computational tasks $\left\langle f^i_{\mathbf{A}_i}(\cdot) \right\rangle_{i=1}^n$ where $\mathbf{A}_i \in \mathbb{R}^{\ell_i \times m}$ for some integers ℓ_i and m, and the rows in $\langle \mathbf{A}_i \rangle$ are coded combinations of the rows in $\mathbf{A}$.
- A decoding function that correctly outputs the correct result $f_{\mathbf{A}}(\cdot)$ when it receives results from a decodable set of local computations.

To assign the computation tasks to each worker, we use random linear combinations of the r rows of the matrix $\mathbf{A}$. As an example, if worker i's load of computation is to compute 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 can recover the result $\mathbf{A}\mathbf{x}$ from any r inner products received from the worker nodes with probability 1. Therefore, the master node does not need to wait for all the worker nodes to complete their computations, as any first r inner products can be used to decode the output. More precisely, let $\mathbf{S}_i \in \mathbb{R}^{\ell_i \times r}$ be the coding matrix for worker i, where the entries of $\mathbf{S}_i$ are i.i.d. $\mathcal{N}(0, 1)$. The computation matrix assigned to worker i is $\mathbf{A}_i = \mathbf{S}_i\mathbf{A}$, i.e. worker i computes $\mathbf{A}_i\mathbf{x}$ and returns the result to the master node. Upon receiving r inner products, the aggregated results 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 matrices, and it is full-rank with probability 1. Therefore, the master node can recover $\mathbf{A}\mathbf{x} = \mathbf{S}^{-1}_{(r)}\mathbf{z}$. [1]

[1]Maximum-Distance Separable (MDS) codes can also be used for assigning the local computation tasks. In this 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. Again, the output $\mathbf{A}\mathbf{x}$ can be recovered from the inner products of any r coded rows with the input vector $\mathbf{x}$.

B. Network Model

The network model is based on a master-worker setup (See Fig. 1 as an illustration.). 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 $\mathbf{Ax}$. The random variable T_i denotes the task runtime at node i and has a shifted exponential distribution:

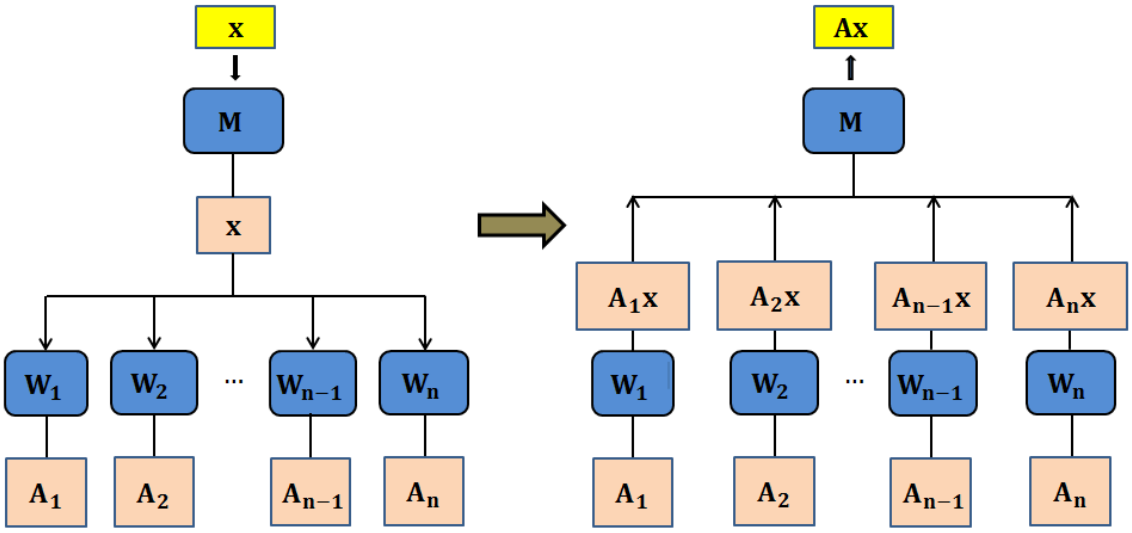


Fig. 1: **Master-worker setup of the computing cluster.** The master node receives the input vector $\mathbf{x}$ and broadcasts it to all the worker nodes. Upon receiving the input, worker node i starts computing the inner products of the input vector with the locally assigned rows, i.e. $\mathbf{y}_i = \mathbf{A}_i\mathbf{x}$, 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 $\mathbf{Ax}$ is recovered.

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

where μ_i is the straggling parameter, a_i is the shift parameter, and ℓ_i denotes the number of coded rows assigned to 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 [11] for latency in querying data files from cloud storage systems. We assume that the runtime random variables are mutually independent.

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 some integers r, 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 at least r inner products, i.e. a decodable set of results. We aim at solving the following optimization problem:

$$\mathcal{P}_{\text{main}}: \quad \underset{\boldsymbol{\ell}}{\text{minimize}} \quad \mathbb{E}[T_{\mathsf{CMP}}]. \tag{2}$$

For a homogeneous cluster, to achieve a coded solution, one can divide $\mathbf{A}$ into k equal parts vertically, and apply an (n, k) MDS code to these submatrices. The master node can then obtain the final result from any k responses. In [8], the authors find the optimal k for minimizing the average running time.

For the heterogeneous cluster, assigning equal loads to servers is clearly not optimal. Moreover, directly finding the optimal solution to the optimization problem $\mathcal{P}_{\text{main}}$ seems to be intractable. 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. However, this is not possible in heterogeneous clusters, where the load allocation is non-uniform. In Section III, we present an alternate formulation to $\mathcal{P}_{\text{main}}$ in (2), and show that the solution to the alternate formulation is tractable and provably asymptotically optimal. 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 i remain constant, i.e. $r = \Theta(n)$, $a_i = \Theta(1)$, and $\mu_i = \Theta(1)$.

III. Heterogeneous Coded Matrix Multiplication

As discussed in Subsection II-C, the goal is to solve $\mathcal{P}_{\text{main}}$, i.e. finding the optimal load allocation that minimizes the expected execution time, $\mathbb{E}[T_{\mathsf{CMP}}]$; however, due to the combinatorial nature of the problem, this needs an exhaustive search over all possible load allocations. In this section, we propose the HCMM algorithm, which is derived from solving an alternate problem formulation towards finding the optimal load allocation. We will further compare the average runtime of HCMM with the uncoded scheme.

A. Alternative Formulation 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 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 $i \in [n]$. Then the *aggregate return* at the master node at time t is:

$$X(t) = \sum_{i=1}^{n} X_i(t). \tag{3}$$

It is easy to show that the expected number of equations aggregated at the master node is:

$$\mathbb{E}\big[X(t)\big] = \sum_{i=1}^{n} \mathbb{E}\big[X_i(t)\big] = \sum_{i=1}^{n} \ell_i \left(1 - e^{-\frac{\mu_i}{\ell_i}(t - a_i\,\ell_i)}\right). \tag{4}$$

We propose the following two-step alternative formulation for $\mathcal{P}_{\text{main}}$ defined by (2). First, for a fixed time t, we maximize the aggregate return over different load allocations, i.e.

$$\mathcal{P}^{(1)}_{\text{alt}} : \boldsymbol{\ell}^*(t) = \arg\max_{\boldsymbol{\ell}} \mathbb{E}\big[X(t)\big], \tag{5}$$

and then, given load allocation $\boldsymbol{\ell}^*(t)$, 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.

$$\begin{aligned} \mathcal{P}^{(2)}_{\text{alt}} : \quad & \text{minimize} \quad t \\ & \text{subject to} \quad \Pr\big[X^*(t) < r\big] = o\left(\frac{1}{\log n}\right), \end{aligned} \tag{6}$$

where $X^*(t)$ is the aggregate return at time t for optimal loads obtained from $\mathcal{P}^{(1)}_{\text{alt}}$, i.e.

$$X^*(t) = \sum_{i=1}^{n} X_i^*(t) = \sum_{i=1}^{n} \ell_i^*(t)\,\mathbb{1}_{\{T_i < t\}}. \tag{7}$$

From now onwards, we denote the solution to the two step-problem as t^* and the corresponding load allocation $\boldsymbol{\ell}^*$ that is the load allocation of the HCMM algorithm. One of the main contributions of this paper is stated in the following theorem.

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,* $\lim_{n\to\infty} \mathbb{E}[T_{\mathsf{HCMM}}] = \lim_{n\to\infty} \mathbb{E}[T_{\mathsf{OPT}}]$.

Subsection III-C provides the proof of Theorem 1. Moreover, in Subsection III-D, we will compare the performance of HCMM to the optimal uncoded scheme and show that HCMM is $\Theta\big(\log n\big)$ times faster than the optimal uncoded scheme.

B. Solving the Alternate Formulation

We first proceed to solve $\mathcal{P}^{(1)}_{\text{alt}}$ in (5). Since there is no constraint on load allocations, this problem can be decomposed to n decoupled optimization problems, i.e.

$$\ell_i^*(t) = \arg\max_{\ell_i} \mathbb{E}\big[X_i(t)\big],\ i \in [n]. \tag{8}$$

One can easily check that $\mathbb{E}\big[X_i(t)\big]$ is concave in ℓ_i for $\ell_i \in (0, \infty)$, since

$$\frac{\partial^2}{\partial {\ell_i}^2} \mathbb{E}[X_i(t)] = -e^{-\frac{\mu_i}{\ell_i}(t - a_i\ell_i)} \frac{\mu_i^2 t^2}{{\ell_i}^3} < 0,$$

and the solution to (8) is obtained as follows,

$$\begin{aligned} \frac{\partial}{\partial \ell_i} \mathbb{E}\big[X_i(t)\big] &= 1 - e^{-\frac{\mu_i}{\ell_i}(t - a_i\ell_i)} \left(\frac{\mu_i t}{\ell_i} + 1\right) = 0, \\ \implies \ell_i^*(t) &= \frac{t}{\lambda_i}, \end{aligned} \tag{9}$$

where $\lambda_i = \Theta(1)$ is a constant independent of t and is the positive solution to the equation $e^{\mu_i x} = e^{a_i\mu_i}(\mu_i x + 1)$. Thus,

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

where

$$s = \sum_{i=1}^{n} \frac{1}{\lambda_i} \left(1 - e^{-\mu_i\lambda_i(1 - \frac{a_i}{\lambda_i})}\right) = \sum_{i=1}^{n} \frac{\mu_i}{1 + \mu_i\lambda_i} = \Theta(n). \tag{11}$$

Let τ^* be the solution to the following equation:

$$\mathbb{E}\big[X^*(t)\big] = \sum_{i=1}^{n} \ell_i^*(t) \left(1 - e^{-\frac{\mu_i}{\ell_i^*(t)}(t - a_i\ell_i^*(t))}\right) = r. \tag{12}$$

Using (10), we find that

$$\tau^* = \frac{r}{s} = \Theta(1), \tag{13}$$

$$\ell_i^*(\tau^*) = \frac{\tau^*}{\lambda_i} = \frac{r}{s\lambda_i}. \tag{14}$$

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

Lemma 1. *Let t^* be the solution to the alternative formulation $\mathcal{P}_{\text{alt}}$ in (5-6) and τ^* be the solution to (12). Then,*

$$\tau^* \leq t^* \leq \tau^* + o(1). \tag{15}$$

We prove this lemma in Appendix. In the next subsection, we prove that the solution to $\mathcal{P}_{\text{alt}}$ is asymptotically optimal.

C. Asymptotic Optimality

In this subsection, we prove Theorem 1. First, we show that

$$\mathbb{E}[T_{\mathsf{HCMM}}] \leq t^* + o(1). \tag{16}$$

Since t^* satisfies $\mathcal{P}_{\text{alt}}$, $\Pr[X^*(t^*) < r] = o(\frac{1}{\log n})$. Let us define a new random variable T_u as follows:

$$T_u = \begin{cases} T_{\mathsf{HCMM}} & \text{if } T_{\mathsf{HCMM}} \leq t^* \\ T_{\mathsf{max}} & \text{if } T_{\mathsf{HCMM}} > t^* \end{cases} \tag{17}$$

where T_{max} is the random variable denoting the finish time of all the workers. By definition, T_u is stochastically greater than T_{HCMM}. Therefore, $\mathbb{E}[T_{\mathsf{HCMM}}] \leq \mathbb{E}[T_u]$. Let the probability density of T_{max} be $f_{\mathsf{max}}(t)$, where $t \geq \max_i a_i\ell_i$, for $i \in [n]$. We now upper bound $\mathbb{E}[T_u]$ as follows.

$$\begin{aligned} \mathbb{E}[T_u] =& \Pr[T_{\mathsf{HCMM}} \leq t^*]\mathbb{E}[T_u|T_{\mathsf{HCMM}} \leq t^*] \\ &+ \Pr[T_{\mathsf{HCMM}} > t^*]\mathbb{E}[T_u|T_{\mathsf{HCMM}} > t^*] \\ =& \Pr[T_{\mathsf{HCMM}} \leq t^*]\mathbb{E}[T_{\mathsf{HCMM}}|T_{\mathsf{HCMM}} \leq t^*] \\ &+ \Pr[T_{\mathsf{HCMM}} > t^*]\mathbb{E}[T_{\mathsf{max}}|T_{\mathsf{max}} > t^*]. \end{aligned} \tag{18}$$

We can write

$$\mathbb{E}[T_{\mathsf{HCMM}}|T_{\mathsf{HCMM}} \leq t^*] \leq t^*, \tag{19}$$

and

$$\mathbb{E}[T_{\mathsf{max}}|T_{\mathsf{max}} > t^*] = \frac{\int_{t^*}^{\infty} t f_{\mathsf{max}}(t)\,dt}{\int_{t^*}^{\infty} f_{\mathsf{max}}(t)\,dt} \tag{20}$$

$$\leq \frac{\int_{\tau^*}^{\infty} t f_{\mathsf{max}}(t)\,dt}{\int_{\tau^*+o(1)}^{\infty} f_{\mathsf{max}}(t)\,dt} \tag{21}$$

$$\leq \frac{\int_{0}^{\infty} t f_{\mathsf{max}}(t)\,dt}{\int_{\tau^*+o(1)}^{\infty} f_{\mathsf{max}}(t)\,dt} \tag{22}$$

$$= \frac{\int_{0}^{\infty} t f_{\mathsf{max}}(t)\,dt}{\int_{\Theta(1)}^{\infty} f_{\mathsf{max}}(t)\,dt} \tag{23}$$

$$\leq \Theta(\log n), \tag{24}$$

where we use the result in (15) to derive (21), and the result in (13) to derive (23). For (24), consider another set of n machines replaced with a machine with parameters $(\hat{\mu}, \hat{a})$, where $\hat{\mu} = \min_i \mu_i$ and $\hat{a} = \max_i a_i$. Further, each machine is assigned $\max_i \ell_i^*(\tau^*)$ equations. Finishing time of the new scenario is stochastically larger. For the new scenario, one can show that $\mathbb{E}[T_{\mathsf{max}}] = \Theta(\log n)$. Using (19), (18) and (24), we have

$$\mathbb{E}[T_{\mathsf{HCMM}}] \leq \mathbb{E}[T_u] \leq t^* + o(1). \tag{25}$$

Next, we prove the following lower bound on the average completion time of the optimum algorithm

$$\mathbb{E}[T_{\mathsf{OPT}}] \geq t^* - \delta, \tag{26}$$

where $\delta = \Theta(\sqrt{\frac{\log n}{n}})$. To this end, we show the following two inequalities,

$$\mathbb{E}[T_{\mathsf{OPT}}] \overset{(a)}{\geq} \tau - \delta_1 \overset{(b)}{\geq} t^* - \delta_2 - \delta_1, \tag{27}$$

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

$$r - \mathbb{E}[X_{\mathsf{OPT}}(\tau - \delta_1)] = \sum_{i=1}^{n} \ell_{\mathsf{OPT},i}\big(\Pr[T_i < \tau] - \Pr[T_i < \tau - \delta_1]\big) \tag{28}$$

$$= \sum_{i=1}^{n} \ell_{\mathsf{OPT},i}\left(\frac{d}{d\tau}\Pr[T_i < \tau]\delta_1 + \mathcal{O}(\delta_1^2)\right)$$

$$= \sum_{i=1}^{n} \delta_1 \mu_i \exp\left(-\frac{\mu_i}{\ell_{\mathsf{OPT},i}}(\tau - a_i \ell_{\mathsf{OPT},i})\right) + \mathcal{O}(\log n)$$

$$= \Theta(n\delta_1). \tag{29}$$

The last equality follows the fact that $\ell_{\mathsf{OPT},i} = \Theta(1)$ [2]. By McDiarmid's inequality (See Appendix for its description),

$$\Pr[X_{\mathsf{OPT}}(\tau - \delta_1) \geq r] = \Pr\left[X_{\mathsf{OPT}}(\tau - \delta_1) - \mathbb{E}[X_{\mathsf{OPT}}(\tau - \delta_1)] \geq r - \mathbb{E}[X_{\mathsf{OPT}}(\tau - \delta_1)]\right]$$

$$\leq \exp\left(-\frac{2\big(\mathbb{E}[X_{\mathsf{OPT}}(\tau - \delta_1)] - r\big)^2}{\sum_i {\ell_{\mathsf{OPT},i}}^2}\right)$$

$$= \exp(-\Theta(n\delta_1^2)) = o\left(\frac{1}{\log n}\right),$$

implying inequality (a). We proceed to prove (b) by showing the following two inequalities,

$$\tau^* \geq t^* - \delta_2, \tag{30}$$

$$\tau \geq \tau^*, \tag{31}$$

where τ^* is obtained in (13). By McDiarmid's inequality,

$$\Pr\left[X^*(\tau^* + \delta_2) \leq r\right] \leq \exp\left(-\frac{2\big(\mathbb{E}[X^*(\tau^* + \delta_2)] - r\big)^2}{\sum_i \ell_i^{*2}(\tau^* + \delta_2)}\right)$$

$$= \exp\left(-\frac{2\big((\tau^* + \delta_2)s - r\big)^2}{\sum_i \ell_i^{*2}(\tau^* + \delta_2)}\right)$$

$$= \exp\left(-\frac{2\delta_2^2 s^2}{\big((\frac{r}{s})^2 + \delta_2^2 + 2\delta_2 \frac{r}{s}\big)\sum_i \lambda_i^2}\right)$$

$$= e^{-\Theta(\log n)} = o\left(\frac{1}{\log n}\right),$$

implying (30). Now, given the fact that HCMM maximizes the expected aggregate return, we have

$$\mathbb{E}[X^*(t)] > \mathbb{E}[X_{\mathsf{OPT}}(t)], \tag{32}$$

for every t, which implies (31). Thus,

$$\lim_{n\to\infty} \mathbb{E}[T_{\mathsf{HCMM}}] = \lim_{n\to\infty} \mathbb{E}[T_{\mathsf{OPT}}] = \tau^* = \frac{r}{\sum_{i=1}^{n} \frac{\mu_i}{1+\mu_i\lambda_i}}.$$

D. Comparison with Uncoded Scheme

In this subsection, we present a comparison between the performance of HCMM and the best 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. Next, we state and prove the following lemma regarding the computation time of the optimal uncoded scheme.

Lemma 2. *Let $T^{\mathsf{UC}}_{\mathsf{OPT}}$ denote the completion time of the optimum uncoded distributed matrix multiplication algorithm. Then,*

$$\mathbb{E}[T^{\mathsf{UC}}_{\mathsf{OPT}}] = \Theta\big(\log n\big). \tag{33}$$

Proof. We show that

$$\mathbb{E}[T^{\mathsf{UC}}_{\mathsf{OPT}}] > c \log n, \tag{34}$$

for a constant c independent of n. For a set of machines with parameters $\{(\mu_i, a_i)\}_{i=1}^n$, let $\tilde{\mu} = \max_i \mu_i$ and $\tilde{a} = \min_i a_i$. Now, consider another set of n machines in which every machine is replaced with a faster machine with parameters $(\tilde{\mu}, \tilde{a})$.

[2]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 $\tau \leq \Theta(1)$. 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.

Since the computation times of the machines are i.i.d., one can show that the optimal load allocation for these machines is uniform, i.e.

$$\widetilde{\ell}_i^* = \frac{r}{n}, \tag{35}$$

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

$$\Pr(\widetilde{T}_i \leq t) = 1 - e^{-\frac{\tilde{\mu}}{\widetilde{\ell}_i^*}(t - \tilde{a}\widetilde{\ell}_i^*)} \tag{36}$$

$$= 1 - e^{-\tilde{\mu}\frac{n}{r}(t - \tilde{a}\frac{r}{n})}, \text{ for } t \geq \frac{\tilde{a}r}{n}, \tag{37}$$

and the expected computation can be written as

$$\mathbb{E}[\widetilde{T}_i] = \frac{r}{n}\left(\frac{1}{\tilde{\mu}} + \tilde{a}\right), \tag{38}$$

for all $i \in [n]$. The computation time for the new set of machines is

$$\mathbb{E}[\widetilde{T}_{\mathsf{OPT}}] = \mathbb{E}[\max_i \widetilde{T}_i] = \frac{\tilde{a}r}{n} + \frac{rH_n}{n\tilde{\mu}}, \tag{39}$$

where $H_n = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n} > \log(n+1)$ is the sum of harmonic series.

It is easy to see that the expected completion time of the optimal uncoded scheme is larger than the $\mathbb{E}[\widetilde{T}_{\mathsf{OPT}}]$, since all the machines are replaced with faster ones. Therefore,

$$\mathbb{E}[T_{\mathsf{OPT}}^{\mathsf{UC}}] \geq \mathbb{E}[\widetilde{T}_{\mathsf{OPT}}] > \frac{\tilde{a}r}{n} + \frac{r}{n\tilde{\mu}}\log(n+1) > c\log(n), \tag{40}$$

for a constant c independent of n. This follows from the fact that $r = \Theta(n)$, $a_i = \Theta(1)$, and $\mu_i = \Theta(1)$ for $i \in [n]$.

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

$$\mathbb{E}[T_{\mathsf{OPT}}^{\mathsf{UC}}] \leq \frac{\hat{a}r}{n} + \frac{r}{n\hat{\mu}}H_n < C\log n, \tag{41}$$

for another constant C. From (40) and (41), one can conclude that

$$\mathbb{E}[T_{\mathsf{OPT}}^{\mathsf{UC}}] = \Theta\big(\log n\big). \tag{42}$$

□

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

$$\mathbb{E}[T_{\mathsf{HCMM}}] = \Theta(1). \tag{43}$$

Comparing (42) and (43) demonstrates that HCMM outperforms the best uncoded scheme by a factor of $\Theta(\log n)$, i.e.

$$\frac{\mathbb{E}[T_{\mathsf{OPT}}^{\mathsf{UC}}]}{\mathbb{E}[T_{\mathsf{HCMM}}]} = \Theta\big(\log n\big). \tag{44}$$

IV. Numerical Analysis

We now present numerical results evaluating the performance of HCMM. Specifically, we consider two benchmark load allocation schemes for performance comparison:

1) **Uncoded Load Balanced (ULB)**: In load balancing, the number of inner products computed by a worker node is proportional to its speed i.e. $\ell_i \propto \mu_i$.
2) **Coded Equal Allocation (CEA)**: 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. In other words, CEA gives the best load allocation among all equal load allocation schemes.

We consider the practical problem of computing 500 inner products using 100 workers for three scenarios:

1) **Scenario 1 (2-mode heterogeneity)**: μ_i is 1 for 50 workers, and 3 for the other 50 workers.
2) **Scenario 2 (3-mode heterogeneity)**: μ_i is 3 for 50 workers, 1 for 25 workers, and 9 for 25 workers.
3) **Scenario 3 (Random heterogeneity)**: μ_i is uniformly distributed in $\{1, 3, 9\}$.

In all scenarios, we assume $a_i\mu_i = 1$ for each worker.

Fig. 2 illustrates the gain in the performance of HCMM over ULB and CEA for the three scenarios of cluster heterogeneity. HCMM gives performance improvement of 49% over ULB and $25\% - 34\%$ over CEA. Moreover, HCMM outperforms CEA with much smaller storage redundancy. For HCMM, the storage redundancy is approximately 1.46, while for CEA, the optimal redundancy factor varies in the range of $1.5 - 4.4$.

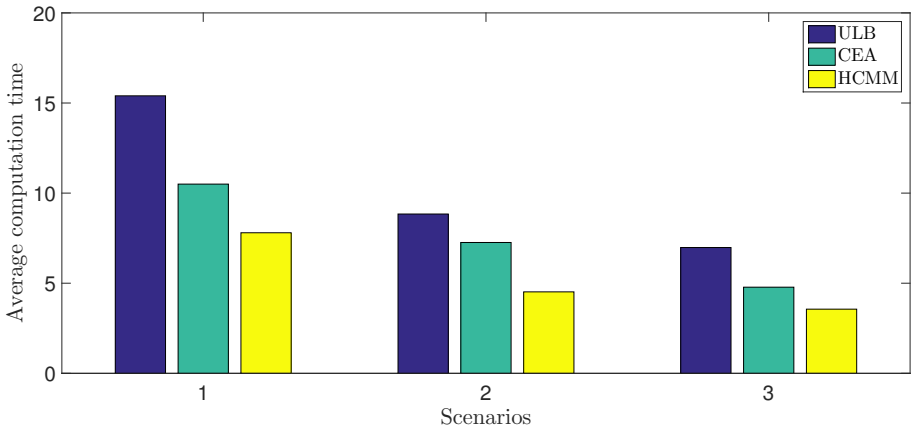


Fig. 2: **Illustration of the performance gain of HCMM over ULB and CEA for the three scenarios.** HCMM achieves a performance improvement of 49% over ULB and $25\% - 34\%$ over CEA.

V. Optimal Load Allocation under Budget Constraint

In this section, we consider the optimization problem in (2) 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, etc. Different cloud computing platforms employ different pricing policies. Table I summarizes the cost per hour of using Amazon EC2 clusters with different parameters [12]. In this section, we take into account the monetary constraint in the optimization problem in (2) and provide a heuristic algorithm towards finding the optimal load allocation under cost budget constraint.

TABLE I: Amazon EC2 Pricing for Linux

machine	vCPU	ECU	Memory (GiB)	Instance Storage (GB)	price
t2.nano	1	variable	0.5	EBS Only	$0.0077 per Hour
t2.micro	1	variable	1	EBS Only	$0.015 per Hour
t2.small	1	variable	2	EBS Only	$0.031 per Hour
t2.medium	2	variable	4	EBS Only	$0.061 per Hour
t2.large	2	variable	8	EBS Only	$0.122 per Hour
t2.xlarge	4	variable	16	EBS Only	$0.244 per Hour
t2.2xlarge	8	variable	32	EBS Only	$0.488 per Hour
m3.medium	1	3	3.75	1 x 4 SSD	$0.077 per Hour
m3.large	2	6.5	7.5	1 x 32 SSD	$0.154 per Hou
m3.xlarge	4	13	15	2 x 40 SSD	$0.308 per Hour
m3.2xlarge	8	26	30	2 x 80 SSD	$0.616 per Hour

For a computation task and a given set of machines, the goal is to minimize the expected run-time while satisfying the budget constraint C:

$$\begin{aligned} \mathcal{P}_{\text{main-constrained}}: \quad & \underset{\ell}{\text{minimize}} \quad \mathbb{E}[T_{\mathsf{CMP}}] \\ & \text{subject to} \quad \sum_{i=1}^{N} c_i \mathbb{1}_{\{l_i>0\}} \mathbb{E}[T_{\mathsf{CMP}}] \leq C, \end{aligned} \tag{45}$$

where c_i represents the cost per time unit of using machine $i \in [N]$. Based on existing pricing policies for cloud computation platforms (see Table I), we assume that using a machine with parameters $\{\mu, a\}$ will cost $c = \kappa\mu^{\alpha}$ per time unit, approximately, where $\alpha \geq 1$ and κ is a constant. Notice that parameter μ can roughly represent the speed of the processing machine. In the following, we analyze the problem of optimal load allocation for coded computation subject to cost constraint along with the simplifying assumption that $a_i\mu_i = 1$ for every machine $1 \leq i \leq n$. We assume that there are K types of machines parametrized with $\{\mu_i, a_i\}_{i=1}^{K}$, and $n_i, i \in [K]$ of each type is available to run a distributed computation task, where $n = \sum_{i=1}^{K} n_i$ is the total number of available machines. We first investigate the extreme allocation scenarios, for which the load allocation induces minimum (or maximum) possible cost. The following lemma states a useful observation regarding the minimum (or maximum) cost for carrying out the computation.

Lemma 3. *Given a computation task and a set of machines, the minimum (or maximum) cost is induced by running the task only on any number of the slowest (or fastest) machines.*

Proof. Assume that $\mu_1 < \cdots < \mu_K$. Suppose that we use n_i number of type i machine. Then, using the result of Theorem 1, the induced (optimal) cost can be written as

$$\texttt{cost}(\text{HCMM}\{n_1, \cdots, n_K\}) = \kappa r \frac{\lambda}{\lambda+1} \frac{\sum_{i=1}^{K} n_i \mu_i^{\alpha}}{\sum_{i=1}^{K} n_i \mu_i}, \tag{46}$$

where λ is the positive solution to the equation $e^{\lambda-1} = \lambda + 1$. We can write

$$\texttt{cost}(\text{HCMM}\{n_1, \cdots, n_K\}) = \kappa r \frac{\lambda}{\lambda+1} \frac{\sum_{i=1}^{K} n_i \mu_i^{\alpha}}{\sum_{i=1}^{K} n_i \mu_i} \geq \frac{\lambda}{\lambda+1} \kappa r \mu_1^{\alpha-1} = \texttt{cost}(\text{HCMM}\{n_1\}) = C_m. \tag{47}$$

By the same argument, we can write

$$\texttt{cost}(\text{HCMM}\{n_1, \cdots, n_K\}) \leq \frac{\lambda}{\lambda+1} \kappa r \mu_K^{\alpha-1} = \texttt{cost}(\text{HCMM}\{n_K\}) = C_M. \tag{48}$$

□

Lemma 3 states that if the available budget is less than C_m, then it is impossible to run the task on the given set of machines while satisfying the budget constraint. Likewise, if the available budget is greater than or equal to C_M, then one can simply run HCMM and obtain the optimal load allocation (which employs all the available machines).

Note that for the case of $\alpha = 1$, $C_m = C_M$. Hence, if the budget $C \geq C_m = C_M$, then HCMM using all machines provides the optimal load allocation that also satisfies the cost constraint; otherwise the optimization problem is infeasible. For the case of $\alpha > 1$, finding the optimal load allocation for a feasible budget-constrained problem requires combinatorial search over all possible allocations. However, as Lemma 3 suggests, using faster machines induces a larger cost. Further, when $a_i \mu_i = 1$ for each server i, the solution τ^* to (13) for carrying out the computation using n_i machines of type i, $1 \leq i \leq n$, can be found to be as follows:

$$\mathbb{E}[T_{\mathsf{HCMM}}] \approx \tau^* = \frac{r(\gamma+1)}{\sum_{i=1}^{K} n_i \mu_i} \tag{49}$$

where $\gamma \approx 2.145$. Therefore, the time of computation increases if we decrease the number of machines. This is the motivation behind our heuristic algorithm for an efficient search which we describe next.

First, we run the HCMM algorithm using all machines. Then, it calculates the corresponding cost according to (46). If $\texttt{cost} > C$, it starts to decrease the number of fastest machines, i.e. $n_K \leftarrow n_K - 1$, and runs HCMM again. While $\texttt{cost} > C$, the algorithm keep 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. The answer is thus $(n_1, \cdots, n_j, n'_{j+1}, 0, \cdots, 0)$ which is the first tuple that satisfies the cost constraint. Therefore, the search space complexity of the algorithm is $\mathcal{O}(n_1 + \cdots + n_K) = \mathcal{O}(n)$ which is much more efficient than the exhaustive search where the complexity is $\mathcal{O}(n_1 \cdots n_K)$. The following pseudo-code summarizes the heuristic algorithm.

Algorithm 1

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

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

1) **Scenario 1**: Two types of machines are available, one parametrized by $(a_1, \mu_1) = (0.5, 2)$, another parametrized by $(a_2, \mu_2) = (0.25, 4)$. For each type, we have 10 machines available. Further, the available budget is $C = 860$. Using Lemma 3, the minimum and maximum induced costs are $C_m = 640$ and $C_M = 1280$. As $C_m \leq C \leq C_M$, there exists a solution to 45. Applying the proposed heuristic search, it takes 9 iterations (see figures 3 and 4), to arrive at the load allocation $(n_1, n_2) = (10, 2)$ which corresponds to the expected $\texttt{cost} = 822.9$ and expected time $\mathbb{E}[T_{\mathsf{HCMM}}] = 11.4286$.

2) **Scenario 2**: Three types of machines are available. These are parametrized by $(a_1, \mu_1) = (1, 1)$, $(a_2, \mu_2) = (0.5, 2)$ and $(a_3, \mu_3) = (0.125, 8)$. Again, 10 machines of each one are available. Further, the available budget is $C = 1500$. Using Lemma 3, the minimum and maximum induced costs for the task of computing $r = 100$ equations are $C_m = 640$

```
cost =

   1.0e+03 *

         0    1.2800    1.2800    1.2800    1.2800    1.2800    1.2800    1.2800    1.2800    1.2800    1.2800
    0.6400    1.0667    1.1520    1.1886    1.2089    1.2218    1.2308    1.2373    1.2424    1.2463    1.2495
    0.6400    0.9600    1.0667    1.1200    1.1520    1.1733    1.1886    1.2000    1.2089    1.2160    1.2218
    0.6400    0.8960    1.0057    1.0667    1.1055    1.1323    1.1520    1.1671    1.1789    1.1886    1.1965
    0.6400    0.8533    0.9600    1.0240    1.0667    1.0971    1.1200    1.1378    1.1520    1.1636    1.1733
    0.6400    0.8229    0.9244    0.9891    1.0338    1.0667    1.0918    1.1116    1.1276    1.1409    1.1520
    0.6400    0.8000    0.8960    0.9600    1.0057    1.0400    1.0667    1.0880    1.1055    1.1200    1.1323
    0.6400    0.7822    0.8727    0.9354    0.9813    1.0165    1.0442    1.0667    1.0852    1.1008    1.1141
    0.6400    0.7680    0.8533    0.9143    0.9600    0.9956    1.0240    1.0473    1.0667    1.0831    1.0971
    0.6400    0.7564    0.8369    0.8960    0.9412    0.9768    1.0057    1.0296    1.0496    1.0667    1.0814
    0.6400    0.7467    0.8229    0.8800    0.9244    0.9600    0.9891    1.0133    1.0338    1.0514    1.0667
```

Fig. 3: Total cost associated with every pair of (n_1, n_2); $0 \leq n_1, n_2 \leq 10$.

```
Et =

         0   80.0000   40.0000   26.6667   20.0000   16.0000   13.3333   11.4286   10.0000    8.8889    8.0000
  160.0000   53.3333   32.0000   22.8571   17.7778   14.5455   12.3077   10.6667    9.4118    8.4211    7.6190
   80.0000   40.0000   26.6667   20.0000   16.0000   13.3333   11.4286   10.0000    8.8889    8.0000    7.2727
   53.3333   32.0000   22.8571   17.7778   14.5455   12.3077   10.6667    9.4118    8.4211    7.6190    6.9565
   40.0000   26.6667   20.0000   16.0000   13.3333   11.4286   10.0000    8.8889    8.0000    7.2727    6.6667
   32.0000   22.8571   17.7778   14.5455   12.3077   10.6667    9.4118    8.4211    7.6190    6.9565    6.4000
   26.6667   20.0000   16.0000   13.3333   11.4286   10.0000    8.8889    8.0000    7.2727    6.6667    6.1538
   22.8571   17.7778   14.5455   12.3077   10.6667    9.4118    8.4211    7.6190    6.9565    6.4000    5.9259
   20.0000   16.0000   13.3333   11.4286   10.0000    8.8889    8.0000    7.2727    6.6667    6.1538    5.7143
   17.7778   14.5455   12.3077   10.6667    9.4118    8.4211    7.6190    6.9565    6.4000    5.9259    5.5172
   16.0000   13.3333   11.4286   10.0000    8.8889    8.0000    7.2727    6.6667    6.1538    5.7143    5.3333
```

Fig. 4: Expected time associated with every pair of (n_1, n_2); $0 \leq n_1, n_2 \leq 10$.

and $C_M = 7680$ 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 $\texttt{cost} = 1483.6$ and the expected time $\mathbb{E}[T_{\mathsf{HCMM}}] = 43.6$.

VI. LDPC Codes for computation

In this section, we propose the use of low-density parity check (LDPC) codes for distributed coded computation. As discussed in Section III, the HCMM algorithm obtains an asymptotically optimal load allocation for carrying out the computation of r equations over n servers. In Section II-A, we proposed the use of random linear combinations of the r rows in order to obtain the $\sum_{i=1}^{n} \ell_i$ rows which are assigned to the n worker nodes. Random linear coding ensures that the original r inner products can be recovered from any r coded computations with probability 1. However, decoding of such codes is equivalent to inverting an $r \times r$ matrix which has a complexity of $\mathcal{O}(r^3)$ and thus can become critical as r becomes large. We propose the use of LDPC codes which allow for linear time decoding complexity. Further, encoding complexity is not significant for the computation since load assignment is done prior to the distributed computation. Therefore, we concentrate only on the complexity of decoding. 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 $r(1+\delta)$ coded inner products with high probability. Ideally, we would like to have $\delta > 0$ to be as small as possible.

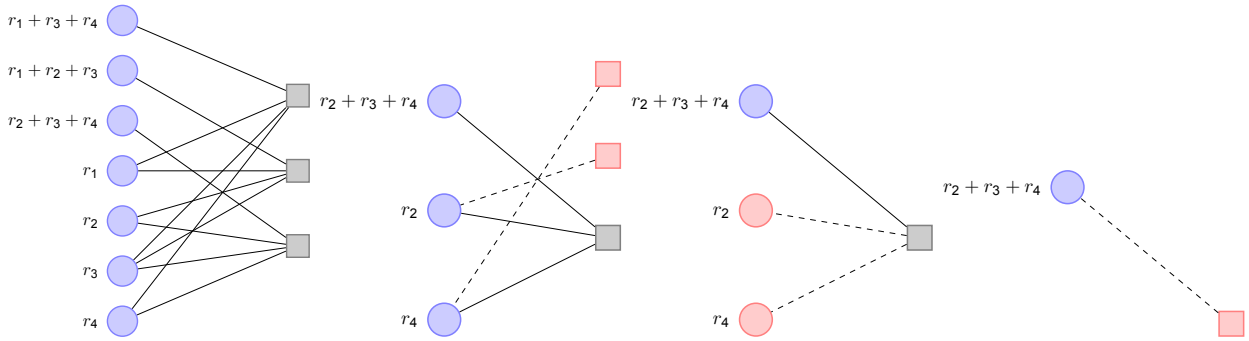
LDPC codes [13] are based on sparse binary parity check matrices. For a binary erasure channel (BEC), each symbol in the codeword is erased with probability p independently. Iterative or peeling decoder can be used to recover the erasures in the received codeword. We illustrate this process through the following example.

Example 2. Suppose that $N = \sum_{i=1}^{n} \ell_i = 7$ and we want to compute the inner products r_1, r_2, r_3 and r_4 of a matrix A having four rows (r_1, r_2, r_3, r_4) with an input x. We use a $(7, 4)$ LDPC code with the following parity check matrix:

$$H = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{bmatrix}.$$

To determine the coded inner products, we find the corresponding generator matrix of the code in standard form:

$$G = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}.$$



(a) Sparse graph for the parity check matrix (b) Stage 1 (c) Stage 2 (d) Stage 3

Fig. 5: Iterative decoding for Example 2.

The bi-partite graph for the parity check matrix is shown in Fig. 5a, where the codeword is obtained as follows:

$$\mathbf{c} = \mathbf{r}\, G \tag{50}$$

where $\mathbf{r} = (r_1, r_2, r_3, r_4)$ is the vector of the original inner products that need to be recovered. Then, ℓ_i equations are mapped on the worker node i, where ℓ_i is the HCMM load allocation for server i. A straggler is equivalent to having erasures in the received inner products, corresponding to the inner products assigned to the straggler node. Therefore, the master node needs to wait until it receives a sufficient number of inner products so that the erasures can be recovered with high probability.

Referring to the tanner graph in Fig. 5a for the ongoing example, sum of all the messages at the message nodes adjacent to a check node should be 0, in order to satisfy the check constraints of parity check matrix. Before starting the decoding process, let each check node store a value 0. In our example, each message is either 0 or 1; therefore all the additions are modulo 2. If a message node is recovered, all edges corresponding to the adjacent check nodes are removed and the message value is added to the values at the adjacent check nodes. When a check node has degree 1, the adjacent message node is assigned the current value at the check node and the edge between them is removed. For the ongoing example, assume that the master node receives the inner products corresponding to the first, second, fourth and sixth nodes. The graph induced by removing the edges corresponding to the received inner products is shown in Fig. 5b. Here, the degrees of the check nodes 1 and 2 are 1; thus the inner products corresponding to the fifth and the seventh nodes can also be recovered. Then the edges corresponding to these nodes are removed which results in Fig. 5d. This makes the degree of the check node 3 as 1; thus the inner product corresponding to the message node 3 can also be recovered. Thus, all the inner products can be recovered.

The above example illustrates the peeling decoding algorithm for recovering erasures in LDPC codes. The algorithm comprises of two main operations that are carried out repeatedly:

- For each recovered message node, add the message value to the check node value for each adjacent check node, and remove the edges incident to the recovered message node.
- Find all the check nodes with degree 1. For each such node, assign the negative of the value at the node to the adjacent message node and remove the edge between them. This is done in order to satisfy the constraint that the sum of values at the message nodes adjacent to any check node is 0. Thus, the message nodes adjacent to the degree 1 check nodes are recovered.

The decoding algorithm terminates, when no more edges can be peeled from the graph. Further, those left nodes with all edges peeled are the recovered left nodes. As the number of operations that this algorithm performs is proportional to the number of edges in the graph, the time complexity is linear in the number of edges in the graph. For a sparse bi-regular graph, the number of edges E is proportional to the number of message nodes N. In our problem of distributed computation, N is proportional to the number of servers n. Therefore, the decoding complexity is $\mathcal{O}(n)$.

The success of the iterative decoding depends on the structure of the adjacency graph induced by the parity check matrix. Consider the bi-partite graph with the message nodes on the left and the check nodes on the right. The left (right) degree i of an edge is the degree of the left (right) node to which it is attached. Let λ_i and ρ_i denote the probabilities that an edge is connected to a node of degree i on the left and right respectively. Then the degree sequences of the graph are characterized by the vectors $(\lambda_1, \lambda_2, \ldots)$ and $(\rho_1, \rho_2, \ldots)$. These can be succinctly represented using the following generating functions:

$$\lambda(x) = \sum_i \lambda_i x^{i-1}$$
$$\rho(x) = \sum_i \rho_i x^{i-1}$$

Let p be the probability that a particular message is erased. We now analyze the decoding process using the density evolution for message passing. In any iteration m, first all the message nodes transmit 'recovered' (or R) or 'not recovered' (or NR) to their adjacent check nodes. A message node sends an R to the adjacent check nodes if the message at that node was either not erased or has been already recovered in a previous iteration of message passing, otherwise it sends NR. Thereafter, each check node sends R to an adjacent message node if it received R from all the remaining message nodes, otherwise it sends NR. Therefore, an erased message node is recovered if it receives R from any of the adjacent check nodes. Let p_m denote the probability that the messages passed from the message nodes to the check nodes in iteration m is NR. Similarly, let q_m be the probability that messages passed from the check nodes to the message nodes is NR. The corresponding probabilities for the $(m+1)^{th}$ iteration need to be updated based on the messages received in the m^{th} iteration. A message node will send NR to a check node if it was erased and it received NR from the remaining check nodes. Therefore, under the assumption that the incoming messages are statistically independent, $p_{m+1} = p\, q_m^{i-1}$. Similarly, a check node sends R to an adjacent message node if it received R from all the remaining adjacent message nodes in that iteration. Therefore, given that a check node has degree i and the incoming messages are statistically independent, the probability that a check node sends NR is $q_{m+1} = 1 - (1 - p_{m+1})^{i-1}$. Using the law of total probability and the degree sequences $(\lambda_1, \lambda_2, \ldots)$ and $(\rho_1, \rho_2, \ldots)$, the following recursion relation is obtained:

$$p_{m+1} = p\, \lambda(1 - \rho(1 - p_m))$$

where $\lambda(x)$ and $\rho(x)$ are the generating functions. For successful decoding, i.e. for recovering all the message nodes, $p_{m+1} < p_m$ so that $p_m \to 0$ as $m \to \infty$. This gives the following density evolution equation [14]:

$$p\lambda(1 - \rho(1 - x)) < x. \tag{51}$$

where $x \in (0, p)$.

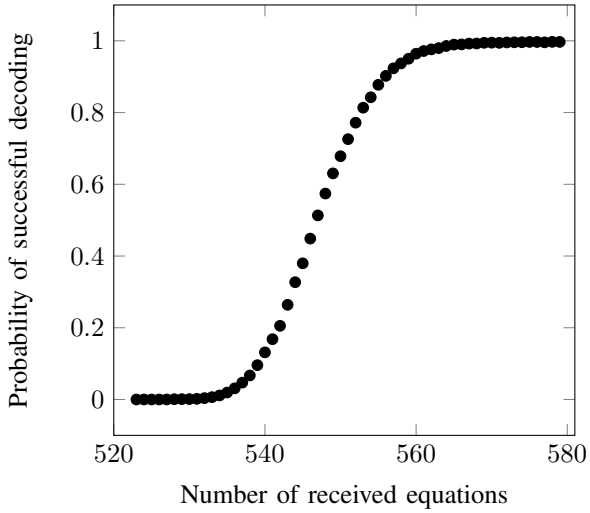


Fig. 6: **Probability of successful decoding vs. number of received equations.** Here, $r = 504$ and $\sum_{i=1}^{n} l_i = 756$. Decoding is successful with high probability if the number of received results is greater than 570.

We now demonstrate the use of bi-regular LDPC codes for the coding design of Scenario 1 (2-mode heterogeneity) described in Section IV. The task is to compute $r = 500$ inner products. For this scenario, the HCMM gives a load allocation with redundancy ≈ 1.5. One can use a $(3, 9)$ bi-regular parity check matrix for decoding and the corresponding generator matrix for encoding. This procedure gives a rate $2/3$ code. We used a $(504, 756)$ code for our purpose. The regular bi-partite graph for the parity check matrix has $\lambda(x) = x^2$ and $\rho(x) = x^8$. Using eq. [51], the upper bound for the probability of erasure p turns out to be ≈ 0.3 for successful decoding with high probability. Thus, the decoding process is successful with high probability if we receive more than $0.7 \times 756 \approx 529$ inner products. Fig. 6 depicts the probability of success of decoding obtained numerically. As the success probability is almost 1 when the number of received inner products is greater than 570, the decoding can be started after waiting for 570 inner products.

We now demonstrate how LDPC codes result in efficient decoding of the coded inner products when the number of assigned equations is large. Let $(\sum_{i=1}^{n} \ell_i / r) = 1.5$. For random linear codes, any r results are sufficient for decoding with probability 1. Using LDPC codes with $(3, 9)$ bi-regular parity check matrix, i.e. where each message node has a degree 3 and each check node has a degree 9, any $1.14\, r$ results is enough to decode the required inner products with high probability for large $\sum_{i=1}^{n} \ell_i$. Fig. 7 demonstrates the benefit of using LDPC codes over random linear codes as $\sum_{i=1}^{n} \ell_i$ increases. The linear decoding complexity of LDPC codes therefore compensates for the delay caused by waiting for an extra $0.14\, r$ results. Therefore, LDPC codes significantly outperform random linear codes in overall decoding time as the total number of assigned equations increases.

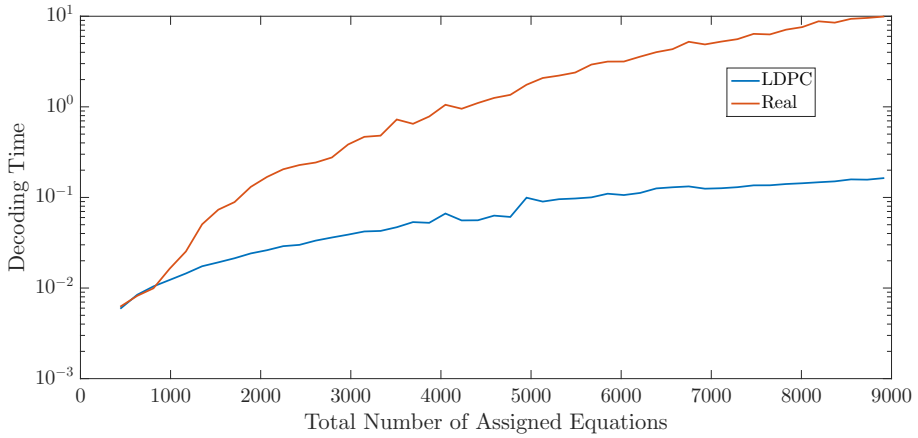


Fig. 7: **Comparison of the decoding times of LDPC codes and random linear codes over real field.** For LDPC decoding to begin, the minimum number of equations required to begin the decoding procedure was set to $1.14 \times r$, whereas for random linear codes, one can recover the original r inner products from any r results from the servers. Decoding for random linear codes is equivalent to inverting matrix which has $\mathcal{O}(r^3)$ complexity, while for LDPC code, the iterative peeling decoding procedure has an $\mathcal{O}(r)$ complexity.[3]

VII. Conclusion

In this paper, we proposed a coding framework for distributed computation in heterogeneous cloud computing environments. In particular, we proposed an alternate formulation to the main problem of minimizing the average computation time for distributed matrix multiplication. The HCMM algorithm, which is derived as a solution to the alternate formulation, asymptotically minimizes the average computation time. We also demonstrated the performance gains of HCMM over two benchmark load allocation schemes. Moreover, a heuristic search was provided in order to find the (sub)optimal load-machine assignment for a given set of machines and a budget constraint. Finally, we analyzed the use of LDPC codes that achieve a linear-time decoding complexity as opposed to high complexity random linear codes.

References

[1] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.
[2] 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.
[3] 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.
[4] 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.
[5] S. Li, M. A. Maddah-Ali, and A. S. Avestimehr, "Coded MapReduce," *53rd Allerton Conference*, Sept. 2015.
[6] S. Li, M. A. Maddah-Ali, Q. Yu, and A. S. Avestimehr, "A fundamental tradeoff between computation and communication in distributed computing," *arXiv preprint arXiv:1604.07086*, 2016.
[7] 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.
[8] K. Lee, M. Lam, R. Pedarsani, D. Papailiopoulos, and K. Ramchandran, "Speeding up distributed machine learning using codes," *arXiv preprint arXiv:1512.02673*, 2015.
[9] 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.
[10] R. Tandon, Q. Lei, A. G. Dimakis, and N. Karampatziakis, "Gradient coding," *arXiv preprint arXiv:1612.03301*, 2016.
[11] 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.
[12] https://aws.amazon.com/ec2/pricing/.
[13] A. Shokrollahi, "Ldpc codes: An introduction," *Digital Fountain, Inc., Tech. Rep*, p. 2, 2003.
[14] M. G. Luby, M. Mitzenmacher, M. A. Shokrollahi, D. A. Spielman, and V. Stemann, "Practical loss-resilient codes," in *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pp. 150–159, ACM, 1997.

Appendix

McDiarmid's Inequality: For any bounded function $f = (f_1, \cdots, f_n)$ and $\epsilon > 0$, if f_i is c_i-Lipschitz for each i and U_is are independent, then

$$\Pr\left[f(U_1, \cdots, U_n) - \mathbb{E}[f(U_1, \cdots, U_n)] > \epsilon\right] \leq \exp\left(-\frac{2\epsilon^2}{\sum_{i=1}^{n} c_i^2}\right), \tag{52}$$

$$\Pr\left[\mathbb{E}[f(U_1, \cdots, U_n)] - f(U_1, \cdots, U_n) > \epsilon\right] \leq \exp\left(-\frac{2\epsilon^2}{\sum_{i=1}^{n} c_i^2}\right). \tag{53}$$

[3]The simulations were carried out using MATLAB on a personal desktop.

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:

$$\Pr\Big[X(t) - \mathbb{E}[X(t)] > \epsilon\Big] \leq \exp\left(-\frac{2\epsilon^2}{\sum_{i=1}^n \ell_i^2}\right), \tag{54}$$

$$\Pr\Big[\mathbb{E}[X(t)] - X(t) > \epsilon\Big] \leq \exp\left(-\frac{2\epsilon^2}{\sum_{i=1}^n \ell_i^2}\right), \tag{55}$$

for any $\epsilon > 0$. Under the assumptions $r = \Theta(n)$, $\sum_{i=1}^n \ell_i = \Theta(n)$, and $\sum_{i=1}^n \ell_i^2 = \Theta(n)$, we have

$$\Pr\Big[X(t) - \mathbb{E}[X(t)] > \epsilon\Big] = o(1),$$

if $\lim_{n\to\infty} n/\epsilon^2 = 0$, which implies that $\epsilon > \Theta(\sqrt{n})$.

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

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

The above equations follow from the fact that $r = \Theta(n)$, $s = \Theta(n)$, $\lambda_i = \Theta(1)$, $\delta = \Theta\big(\sqrt{\frac{\log n}{n}}\big)$, and therefore, $\sum \lambda_i^2 = \Theta(n)$ 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,

$$\tau^* \leq t^* \leq \tau^* + \delta. \tag{56}$$