Coded Computing for Distributed Graph Analytics

Saurav Prakash[†‡], Amirhossein Reisizadeh[*‡], Ramtin Pedarsani[*], Salman Avestimehr[†]
[†] University of Southern California [*] University of California, Santa Barbara
[‡] Authors have equal contribution

***Abstract*—Many distributed graph computing systems have been developed recently for efficient processing of massive graphs. These systems require many messages to be exchanged among computing machines at each step of the computation, making *communication bandwidth* a major performance bottleneck. We present a *coded computing* framework that systematically injects redundancy in the computation phase to enable coding opportunities in the communication phase thus reducing the communication load substantially. Specifically, we propose coded schemes that enable an inverse-linear trade-off (asymptotically) between *computation load* and *average communication load* for Erdös-Rényi (ER) random graph. The proposed scheme for ER graph is shown to be optimal asymptotically as the graph size $n \to \infty$. For finite n, we demonstrate via numerical analysis that for a given computation load r, i.e. when each graph vertex is carefully stored at r servers, the proposed scheme slashes the average communication load by (nearly) r.**

I. Introduction

Graphs are widely used to identify and incorporate the relationship patterns and anomalies inherent in real-life datasets. Their growing scale and importance have prompted the development of various large-scale distributed graph processing frameworks, such as Pregel [1] and GraphLab [2]. The underlying theme in these systems is the "think like a vertex" approach [3] where the computation at each vertex requires only the data available in the neighborhood of the vertex. This approach significantly improves performance in comparison to general-purpose distributed processing systems (e.g., Dryad [4], MapReduce [5]), which do not leverage the underlying structure of graphs.

These distributed graph processing systems, however, require many messages to be exchanged among computing machines (servers) during job execution. As a result, communication bandwidth is a common bottleneck in parallel computations over graphs [6], accounting for more than 50% of the overall execution time in representative cases [7].

We develop a new approach that leverages coding to reduce the communication load in distributed graph processing. Motivated by the "think like a vertex" approach, we describe a mathematical model for MapReduce computations on graphs and show how carefully injecting redundancy in *Map* phase results in significant reduction in the communication load during *Shuffle* phase. The idea is to leverage the underlying graph model and create coded messages that simultaneously satisfy the data demand of multiple computing machines in *Reduce* phase.

Our work is rooted in the recent development of a coding framework for general MapReduce computations that establishes an inverse-linear trade-off between computation and communication – Coded Distributed Computing (CDC) [8]. CDC achieves the communication bandwidth gain of r, when each Map computation is carefully repeated at r servers.

As we move from the general MapReduce framework to graph analytics, the key challenge is that the computation associated with each vertex is a function of graph structure. In particular, each computation needs data *only* from the neighboring vertices, while in the general MapReduce framework, each computation needs all the input files (corresponding to a complete graph). This asymmetry in the data requirements of the computations is the main challenge in developing efficient subgraph allocation and Map computation, Reduce computation allocation and Shuffling schemes.

As the main contribution of this paper, we solve the problem for random Erdös-Rényi (ER) graph. Specifically, for a given computation load r, i.e. when each vertex is stored on average at r distinct servers, we show that the minimum average normalized communication load is $L^*(r) = \frac{1}{r}p(1-\frac{r}{K})+o(p)$, where K denotes the number of servers and $p = \omega(\frac{1}{n^2})$ is the edge probability in the ER graph of size n. To prove the achievability, we propose a coded scheme that creates coding opportunities for communicating messages across machines by Mapping the same graph vertex at different machines, so that each coded transmission satisfies the data demand of multiple machines. For converse, we use cut-set bounds and show that the proposed scheme is asymptotically optimal as $n \to \infty$.

We demonstrate via numerical results that our coded scheme for ER graph achieves near optimal average communication load for finite n and provides a gain of (almost) r in comparison to an uncoded scheme described later.

Related Work: Recently, there has been significant interest in the use of coding theoretic ideas for mitigating several bottlenecks that arise in large scale distributed computation. These works can be divided into two categories: those that tackle the communication bandwidth bottleneck in distributed computation, e.g. [8]–[13], and those who tackle the straggler bottleneck in distributed computation, e.g. [10], [13]–[18].

Notation: We let $[n]$ represent the set $\{1, 2, \ldots, n\}$ for $n \in \mathbb{N}$. For non-negative functions f and g of n, we denote $f = \Theta(g)$ if there are positive constants c_1 and c_2 such that $c_1 \leq f/g \leq$

This material is based upon work supported 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. This work is also in part supported by ONR award N000141612189 and NeTS-1419632 and NSF Grants CCF-1703575 and CCF-1755808.

978-1-5386-4780-6/18/$31.00©2018 IEEE

c_2, and $f = o(g)$ if f/g converges to 0 as n goes to infinity. We denote $f = \omega(g)$, if for any positive constant c, there exists a constant $n_0 \in \mathbb{N}$ such that $f(n) > c \cdot g(n)$ for every $n \geq n_0$.

II. Problem Setting

In this section, we describe the setting and formulate our problem.

A. Computation Model

Consider an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ where $\mathcal{V} = [n]$ denotes the set of graph vertices and $\mathcal{E}$ is the set of edges. A binary file $w_i \in \mathbb{F}_{2^F}$ of size $F \in \mathbb{N}$ is associated with each graph vertex $i \in [n]$. The set of all files is denoted by $\mathcal{W} = \{w_i : i \in \mathcal{V}\}$. The neighborhood of vertex i is denoted by $\mathcal{N}(i) = \{j \in \mathcal{V} : (j,i) \in \mathcal{E}\}$ and the set of files in $\mathcal{N}(i)$ is represented by $\mathcal{W}_{\mathcal{N}(i)} = \{w_j : j \in \mathcal{N}(i)\}$. In general, vertex i can be contained in $\mathcal{N}(i)$, i.e. $\mathcal{G}$ can have self-loops. A computation $\phi_i : \mathbb{F}_{2^F}^{|\mathcal{N}(i)|} \to \mathbb{F}_{2^B}$ is associated with each vertex i, where $\phi_i(\cdot)$ outputs the input files in $\mathcal{W}_{\mathcal{N}(i)}$ to a length B binary stream $o_i = \phi_i(\mathcal{W}_{\mathcal{N}(i)})$. We assume the computation $\phi_i(\cdot)$ can be decomposed as a MapReduce computation: $\phi_i(\mathcal{W}_{\mathcal{N}(i)}) = h_i\big(\{g_{i,j}(w_j) : w_j \in \mathcal{W}_{\mathcal{N}(i)}\}\big)$, where the Map function $g_{i,j} : \mathbb{F}_{2^F} \to \mathbb{F}_{2^T}$ Maps file w_j to a length T binary intermediate value $v_{i,j} = g_{i,j}(w_j)$, $\forall i \in \mathcal{N}(j)$. The Reduce function $h_i : \mathbb{F}_{2^T}^{|\mathcal{N}(i)|} \to \mathbb{F}_{2^B}$ Reduces the intermediate values associated with the output function $\phi_i(\cdot)$ into the final output value $o_i = h_i\big(\{v_{i,j} : j \in \mathcal{N}(i)\}\big)$.

As an example, Fig. 1(a) illustrates a graph with $n = 6$ vertices, each associated with a file, and Fig. 1(b) illustrates the corresponding MapReduce computations.

Note that popular iterative graph algorithms can be expressed in the computation framework described above. For instance, in PageRank, each vertex's rank is Mapped to weighted ranks corresponding to neighboring vertices, and each Reducer updates the vertex's rank by adding up the Map contributions from the neighbors [19], [20].

B. Distributed Implementation

We now describe our model for distributed implementation of the computations described above. We consider a network consisting of K machines that are connected to each other. As the first step, a distributed implementation consists of allocating a subgraph to each machine.

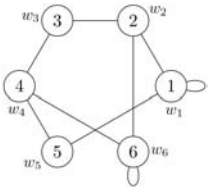
Subgraph Allocation: We denote the subgraph that is allocated to each machine $k \in [K]$ by $\mathcal{M}_k \subseteq \mathcal{V}$. Machine k will then store all files corresponding to subgraph $\mathcal{M}_k$, and will be responsible for computing the Map functions on those files. Note that each file (or vertex) should be Mapped by at least one machine. Furthermore, in order to help with reducing the overall communication load, we allow each file to be Mapped by more than one machine (i.e, increasing the computation load). More formally, the computation load is defined as follows.

Definition 1 (Computation Load): For a subgraph allocation, $\mathcal{M}_k, k \in [K]$, the computation load, r, is defined as

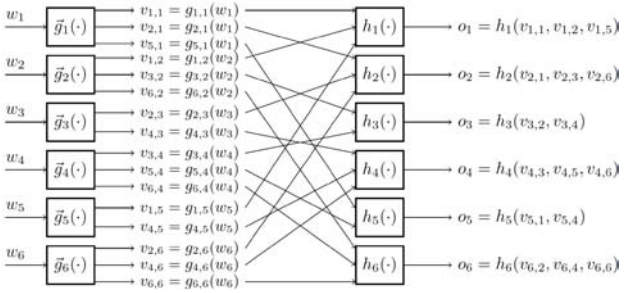
$$r \triangleq \frac{\sum_{k=1}^{K} |\mathcal{M}_k|}{n}. \tag{1}$$

Remark 1: For a desired computation load r, we assume that the subgraphs allocated to servers have all the same size of $r\frac{n}{K}$ vertices (i.e., equal memory is used at all servers).

Computation Allocation: Recall that a Reducer is associated with each vertex of the graph $\mathcal{G}$. The set of vertices whose Reduce computations are assigned to machine k is denoted by $\mathcal{R}_k \subseteq \mathcal{V}$. The Reduce computations are distributed disjointly and uniformly across the K servers, i.e. $|\mathcal{R}_k| = \frac{n}{K}$, $\forall k \in [K]$.



(a) An example of a graph with 6 vertices.



(b) MapReduce decomposition of computations for graph in (a).

Server	Subgraph allocation	Computation allocation	Intermediate values needed
1	$\mathcal{M}_1 = \{1,2,3,4\}$	$\mathcal{R}_1 = \{1,2\}$	$\{v_{1,5}, v_{2,6}\}$
2	$\mathcal{M}_2 = \{1,2,5,6\}$	$\mathcal{R}_2 = \{3,4\}$	$\{v_{3,4}, v_{4,3}\}$
3	$\mathcal{M}_3 = \{3,4,5,6\}$	$\mathcal{R}_3 = \{5,6\}$	$\{v_{5,1}, v_{6,2}\}$

(c) A subgraph and computation allocation for graph in (a).

Fig. 1: An illustrative example.

Given a subgraph and computation allocation to servers, denoted by $A = (\mathcal{M}, \mathcal{R})$ where $\mathcal{M} = (\mathcal{M}_1, \cdots, \mathcal{M}_K)$ and $\mathcal{R} = (\mathcal{R}_1, \cdots, \mathcal{R}_K)$, the computation proceeds distributedly in the following three phases.

Map phase: Each server first Maps the files associated with the subgraph that is allocated to it. More specifically, for each $i \in \mathcal{M}_k$, machine k computes a vector of intermediate values corresponding to the vertices in $\mathcal{N}(i)$: $\vec{g}_i = (v_{j,i} : j \in \mathcal{N}(i))$.

Shuffle phase: To be able to do the final Reduce computations, each server needs the intermediate values corresponding to the neighbors of each vertex that is responsible for its Reduction. Servers exchange messages so that at the end of the Shuffle phase, each server is able to recover its required set of intermediate values. More formally, for each $k \in [K]$, (i) machine k creates a message $X_k \in \mathbb{F}_{2^{c_k}}$ as a function of intermediate values computed locally at that server during the Map phase, i.e. $X_k = \psi_k\big(\{\vec{g}_i : i \in \mathcal{M}_k\}\big)$, where c_k is the length of the binary message X_k, (ii) machine k multicasts X_k to all the remaining servers.

We define the (normalized) communication load of the Shuffle phase as follows.

Definition 2 (Normalized Communication Load): The *normalized* communication load, denoted by L, is defined as

the number of bits communicated by K machines during the Shuffle phase, normalized by the maximum possible total number of bits in the intermediate values associated with all the Reduce functions, i.e.

$$L \triangleq \frac{\sum_{k=1}^{K} c_k}{n^2 T}. \tag{2}$$

Reduce phase: Each server uses its locally computed intermediate values and the messages received from other servers to first construct the required intermediate values for Reduce functions that are allocated to it and then calculates the final output. More formally, for each $k \in [K]$, (i) machine k recovers the needed intermediate values $\{v_{i,j} : i \in \mathcal{R}_k, j \in \mathcal{N}(i), j \notin \mathcal{M}_k\}$ using locally computed intermediate values $\{v_{i,j} : i \in \mathcal{N}(j), j \in \mathcal{M}_k\}$ and received messages $\{X_{k'} : k' \in [K] \setminus \{k\}\}$, (ii) machine k calculates $o_i = h_i(\{v_{i,j} : j \in \mathcal{N}(i)\})$ for all $i \in \mathcal{R}_k$.

To illustrate the above definitions, let us again consider the graph depicted in Fig. 1(a) where $K = 3$ machines are available to carry out the computation. For the subgraph and computation allocation described in Fig. 1(c), each server $k \in \{1, 2, 3\}$ Maps the vertices in subgraph $\mathcal{M}_k$ and computes the Reductions associated with vertices in $\mathcal{R}_k$. The computation load is $r = 2$ and the normalized (uncoded) communication load equals to $L = \frac{6}{36}$.

C. Problem Formulation

For an allowed computation load r, we aim to find the optimal allocation of subgraphs and computations to servers, and the optimal coding for Shuffling in order to minimize the communication load. However, we note that this problem even in the simplest case of $r = 1$ and uncoded transmission in the Shuffling phase is NP-hard for general graphs [21]. Hence, we restrict our attention to random graphs and focus on the average communication load.

We consider a random undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where edges independently exist with probability $\mathbb{P}[(i, j) \in \mathcal{E}]$ for all $i, j \in \mathcal{V}$. Let $\mathcal{A}(r)$ be the set of all possible subgraph and computation allocations for a given computation load r (as defined in the previous subsection). For a graph realization G and an allocation $A \in \mathcal{A}(r)$, we denote by $L_A(r, G)$ the minimum (normalized) communication load (as defined in Definition 2) over all possible Shuffling coding schemes that enable each machine to compute the Reduce functions assigned to it.

We now formally define our problem as follows.

Problem: For a given random undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and a computation load $r \in \mathbb{N}$, our goal is to characterize the minimum average normalized communication load, i.e.

$$L^*(r) \triangleq \inf_{A \in \mathcal{A}(r)} \mathbb{E}_{\mathcal{G}}[L_A(r, \mathcal{G})]. \tag{3}$$

Remark 2: As defined above, $L^*(r)$ essentially reveals a fundamental trade-off between computation and communication in distributed graph processing frameworks.

To solve the problem defined above, we need to establish the optimal subgraph and computation allocations for each server along with the efficient Shuffle scheme.

III. Main Results

In this section, we present the main results of the paper. The main contribution is the characterization of $L^*(r)$ (defined in (3)) for Erdös-Rényi random graph that is defined below.

Erdös-Rényi: Denoted by ER(n, p), this model consists of graphs of size n in which each edge exists with probability $p \in (0, 1]$, independently of other edges.

Theorem 1: For Erdös-Rényi graph ER(n, p) with $p = \omega(\frac{1}{n^2})$, we have

$$\lim_{n \to \infty} \frac{L^*(r)}{p} = \frac{1}{r}(1 - \frac{r}{K}). \tag{4}$$

Remark 3: Achievability of Theorem 1 is proved in Section IV, where we provide a subgraph and computation allocation followed by the code design for Shuffling. Proof of converse for Theorem 1 is provided in Section V.

Remark 4: Theorem 1 reveals an interesting inverse linear trade-off between computation and communication. In particular, the scheme that we propose for achievability of Theorem 1 asymptotically gives a communication load gain of r in comparison to the uncoded scheme that as we discuss later in Section IV only achieves an average normalized communication load of $p(1 - \frac{r}{K})$. This trade-off can be used to leverage additional computing resources and capabilities to alleviate the costly communication bottleneck. Moreover, we numerically demonstrate that even for finite graphs, not only the proposed scheme significantly reduces the communication load, but also has a small optimality gap (Fig. 2).

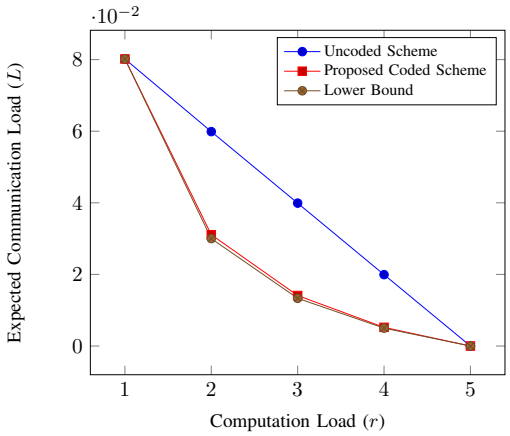


Fig. 2: Performance comparison of the coded scheme with uncoded scheme and the proposed lower bound. The averages for the communication load for the two schemes were obtained over 300 graph realizations with $p = 0.1$ and $K = 5$.

IV. Achievability for Erdös-Rényi Model

We now propose our coded and uncoded schemes for Erdös-Rényi model, and prove the achievability of Theorem 1.

A. Proposed Scheme

As explained in Section II, a scheme for distributed implementation of the computation consists of subgraph allocation, computation allocation, and Shuffling algorithm. We next precisely describe our proposed scheme.

Subgraph Allocation and Map Computation: The n vertices are partitioned into $\binom{K}{r}$ batches of size $g = n/\binom{K}{r}$ denoted

by $\mathcal{B}_\mathcal{T}$, where each of them corresponds to a set $\mathcal{T} \subseteq [K]$ of size r, i.e. $\{1, \cdots, n\} = \cup_{\mathcal{T}\subseteq[K],|\mathcal{T}|=r}\mathcal{B}_\mathcal{T}$. Server $k \in [K]$ Maps the vertices in $\mathcal{B}_\mathcal{T}$ if $k \in \mathcal{T}$. Equivalently, $\mathcal{B}_\mathcal{T} \subseteq \mathcal{M}_k$ if $k \in \mathcal{T}$, i.e. each server Maps $r\frac{n}{K}$ vertices.

Reduce Computation Allocation: The n Reduce functions are uniformly partitioned into K subsets and each subset is assigned to one machine. Thus, each server is responsible for computing $\frac{n}{K}$ Reducers. We denote the proposed subgraph and computation allocation by $A^\bullet$.

Uncoded Shuffle: Given the above subgraph and computation allocation, each vertex in $\mathcal{V}$ requires on average pn intermediate values for the Reduce phase, where a fraction $\frac{r}{K}$ of them are locally available. The rest of the intermediate values should be sent from the other servers. Therefore, the average normalized communication load for the uncoded scheme is $p(1 - \frac{r}{K})$.

Coded Shuffle: Consider a set of servers $\mathcal{S} \subseteq [K]$, $|\mathcal{S}| = r+1$. For each server $k \in \mathcal{S}$, let $\mathcal{Z}^k_{\mathcal{S}\setminus\{k\}}$ be the set of all intermediate values needed by Reduce functions in k, which are available exclusively at each server $k' \in \mathcal{S} \setminus \{k\}$, i.e.

$$\mathcal{Z}^k_{\mathcal{S}\setminus\{k\}} = \{v_{i,j} : (i,j) \in \mathcal{E}, i \in \mathcal{R}_k, j \in \cap_{k'\in\mathcal{S}\setminus\{k\}}\mathcal{M}_{k'}\}.$$

For each $k \in \mathcal{S}$, each intermediate value $v_{i,j} \in \mathcal{Z}^k_{\mathcal{S}\setminus\{k\}}$ is evenly split into r segments $v^{(1)}_{i,j}, \cdots, v^{(r)}_{i,j}$, each of size $\frac{T}{r}$ bits. Each segment is associated with a distinct server in $\mathcal{S} \setminus \{k\}$. Therefore, $\mathcal{Z}^k_{\mathcal{S}\setminus\{k\}}$ is evenly partitioned to r sets, which are denoted by $\mathcal{Z}^k_{\mathcal{S}\setminus\{k\},s}$ for $s \in \mathcal{S} \setminus \{k\}$. Depending on the realization of the graph, the maximum possible size of $\mathcal{Z}^k_{\mathcal{S}\setminus\{k\}}$ is $\tilde{g} = g\frac{n}{K} = \frac{n^2}{K\binom{K}{r}} = \Theta(n^2)$. Each server $s \in \mathcal{S}$ creates an $r \times \tilde{g}$ table and fills that out with segments which are associated with it. Each row of the table is filled from left by the segments in one of the sets $\mathcal{Z}^k_{\mathcal{S}\setminus\{k\},s}$, where $k \in \mathcal{S} \setminus \{s\}$ (Fig. 3). Then, server s broadcasts the XOR of all the segments in each (non-empty) column of the table (for each non-empty column, the empty entries are zero padded). Clearly, there exist at most $\tilde{g}$ of such coded messages. The process is carried out similarly for all remaining subsets $\mathcal{S} \subseteq [K]$ with $|\mathcal{S}| = r + 1$.

After the Shuffle phase, each server can recover the intermediate values associated with its assigned set of Reduce functions using the received coded messages and the locally computed intermediate values.

Remark 5: The proposed scheme carefully aligns and combines the existing intermediate values to benefit from the coding opportunities. This resolves the issue posed by the asymmetry in the data requirements of the Reducers which is one of the main challenges in moving from the general MapReduce framework to graph analytics.

As an example, consider a system of $K = 3$ servers and computation load $r = 2$. For the graph in Fig. 1(a), Fig. 1(c) summarizes the subgraph and computation allocation followed from the proposed scheme. Consider the set $\mathcal{S} = [K]$ of size $r + 1 = 3$. Every intermediate value in $\mathcal{Z}^3_{\{1,2\}} = \{v_{5,1}, v_{6,2}\}$ is split into $r = 2$ segments, each associated with a distinct server in $\{1, 2\}$. This is done similarly for servers 1 and 2. Then, servers 1, 2, and 3 broadcast their coded messages $X_1 = \{v^{(1)}_{5,1} \oplus v^{(1)}_{4,3}, v^{(1)}_{3,4} \oplus v^{(1)}_{6,2}\}$, $X_2 = \{v^{(2)}_{5,1} \oplus v^{(1)}_{1,5}, v^{(2)}_{6,2} \oplus v^{(1)}_{2,6}\}$, and $X_3 = \{v^{(2)}_{4,3} \oplus v^{(2)}_{1,5}, v^{(2)}_{3,4} \oplus v^{(2)}_{2,6}\}$, respectively.

All three servers can recover their needed intermediate values. For instance, server 3 needs $v_{5,1}$ to carry out the Reduce function associated with vertex 5. Since it has already Mapped vertices 3 and 5, intermediate values $v_{4,3}$ and $v_{1,5}$ are available locally. Thus, server 3 can recover $v^{(1)}_{5,1}$ and $v^{(2)}_{5,1}$ from $v^{(1)}_{5,1} \oplus v^{(1)}_{4,3}$ and $v^{(2)}_{5,1} \oplus v^{(1)}_{1,5}$, respectively. Therefore, the overall uncoded communication load $\frac{6}{36}$ is reduced to coded load $\frac{3}{36}$.

B. Proof of Achievability

We first define the average normalized communication loads for our scheme as follows. For a graph realization G, the proposed allocation $A^\bullet \in \mathcal{A}(r)$, and the proposed coded and uncoded Shuffling scheme, we denote the normalized coded and uncoded communication loads by $L^{\mathsf{C}}_{A^\bullet}(r, G)$ and $L^{\mathsf{UC}}_{A^\bullet}(r, G)$, respectively. The average normalized coded and uncoded communication loads will then be $\bar{L}^{\mathsf{C}}_{A^\bullet} \triangleq \mathbb{E}_\mathcal{G}[L^{\mathsf{C}}_{A^\bullet}(r, \mathcal{G})]$ and $\bar{L}^{\mathsf{UC}}_{A^\bullet} \triangleq \mathbb{E}_\mathcal{G}[L^{\mathsf{UC}}_{A^\bullet}(r, \mathcal{G})]$, respectively.

We now apply the proposed coded scheme to graph $\mathcal{G}$ and compute the induced average coded load. WLOG, we analyze our algorithm by a generic argument for servers $\mathcal{S} = \{s_1, \cdots, s_{r+1}\}$ which can be similarly applied for other sets of servers due to the symmetric structure induced by the graph model and allocations. Following the Shuffle phase of the proposed scheme, consider r servers $s_2, \cdots, s_{r+1}$ and the $(r + 1)$'th server s_1. Server s_1 broadcasts at most $\tilde{g}$ coded messages $X^1, \cdots, X^{\tilde{g}}$ which are exclusively useful for servers $s_2, \cdots, s_{r+1}$. Each X^j, $j \in [\tilde{g}]$, is XOR of at most r segments of size $\frac{T}{r}$ bits, associated with server s_1. More formally, for all $j \in [\tilde{g}]$, $X^j = \bigoplus_{i=1}^r v^{(1)}_{\alpha(i,j)}$, where $\mathcal{Z}^{s_{i+1}}_{\mathcal{S}\setminus\{s_{i+1}\},s_1} = \{v^{(1)}_{\alpha(i,j)} : j \in [\tilde{g}]\}$ and $i \in [r]$ (Fig. 3).

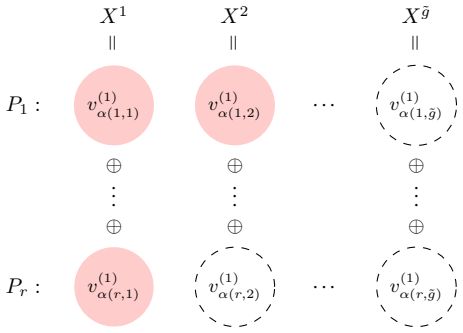


Fig. 3: Creating coded messages by aligning intermediate values.

Let $\mathrm{Bern}(p)$ random variable $E_{\alpha(i,j)}$ indicate the existence of the edge $\alpha(i, j) \in \mathcal{V}\times\mathcal{V}$, i.e. $E_{\alpha(i,j)} = 1$, if $\alpha(i, j) \in \mathcal{E}$, and $E_{\alpha(i,j)} = 0$, otherwise. Clearly, for all vertices $i, j, t, u \in \mathcal{V}$, $E_{\alpha(i,j)}$ is independent of $E_{\alpha(t,u)}$ if $\alpha(i, j)$ and $\alpha(t, u)$ do not represent the same edge, and $E_{\alpha(i,j)} = E_{\alpha(t,u)}$, otherwise. For $i \in [r]$, the random variable P_i is defined as $P_i = \sum_{j=1}^{\tilde{g}} E_{\alpha(i,j)}$, i.e. each P_i is sum of $\tilde{g}$ possibly dependent $\mathrm{Bern}(p)$ random variables. Note that P_i's are not independent in general. By careful alignment of present intermediate values (Fig. 3), s_1 broadcasts Q coded messages each of size $\frac{T}{r}$ bits,

where $Q = \max_{i\in[r]} P_i$. Thus, the total coded communication load sent from server s_1 exclusively for servers $s_2, \cdots, s_{r+1}$ is $\frac{T}{r}Q$ bits. By similar arguments for other sets of servers, we can characterize the average coded communication load of the proposed scheme as

$$\bar{L}_{A^\bullet}^{\mathsf{C}} = \frac{1}{rn^2} K \binom{K-1}{r} \mathbb{E}[Q]. \tag{5}$$

The following lemma asymptotically upper bounds $\mathbb{E}[Q]$ and the proof is provided in Section IV-C .

Lemma 1: For ER(n,p) graphs with $p = \omega(\frac{1}{n^2})$, we have

$$\mathbb{E}[Q] \le p\tilde{g} + o(p\tilde{g}). \tag{6}$$

Putting (5) and Lemma 1 together, we have

$$L^*(r) \le \bar{L}_{A^\bullet}^{\mathsf{C}} \le \frac{1}{r} p(1 - \frac{r}{K}) + o(p),$$

hence the acievability claimed in Theorem 1 is proved. Finally, we note that as explained in the uncoded Shuffle algorithm, the average normalized uncoded communication load of the proposed scheme is $\bar{L}_{A^\bullet}^{\mathsf{UC}} = p(1 - \frac{r}{K})$, which implies that our scheme achieves an asymptotic gain r.

C. Proof of Lemma 1

For any $s' > 0$, we have

$$e^{s'\mathbb{E}[Q]} \le \mathbb{E}[e^{s'Q}] = \mathbb{E}\big[\max_{i=1,\cdots,r} e^{s'P_i}\big] \le \mathbb{E}\Big[\sum_{i=1}^{r} e^{s'P_i}\Big]$$

$$= \sum_{i=1}^{r} \mathbb{E}[e^{s'P_i}] \le r(1 + pe^{2s'} - p)^{\tilde{g}/2},$$

where the last inequality follows from Lemma 2 (proved in [22]). Taking logarithm from both sides yields

$$\mathbb{E}[Q] \le \frac{1}{s'}\log(r) + \frac{\tilde{g}}{2s'}\log(1 + pe^{2s'} - p).$$

Let us substitute $s = 2s'$ in the above equation. Then,

$$\mathbb{E}[Q] \le \frac{1}{s}\log(r^2) + \frac{\tilde{g}}{s}\log(1 + pe^{s} - p), \forall s > 0.$$

Let $\bar{p} \triangleq 1 - p$ and pick $s_* = 2\sqrt{\frac{\log(r)}{\tilde{g}p\bar{p}}}$. As $\tilde{g} = \Theta(n^2)$, for $p = \omega(\frac{1}{n^2})$, $s^* = o(1)$. By Taylor series expansion, we have

$$\log(1 + p(e^{s_*} - 1)) = \sum_{j=1}^{\infty} (-1)^{j+1} \frac{(p(e^{s_*} - 1))^j}{j}$$

$$= ps_* + \frac{p\bar{p}}{2}s_*^2 + o(ps_*^2).$$

Putting everything together, we have

$$\mathbb{E}[Q] \le \frac{1}{s_*}\log(r^2) + \frac{\tilde{g}}{s_*}\big(ps_* + \frac{p\bar{p}}{2}s_*^2 + o(ps_*^2)\big)$$

$$= \tilde{g}p + 2\sqrt{\tilde{g}p\bar{p}\log(r)} + o\big(\sqrt{\tilde{g}p}\big).$$

Recall that $\tilde{g} = \Theta(n^2)$. Therefore, for $p = \omega(\frac{1}{n^2})$ we have $\tilde{g}p = \omega(1)$ and thus $\sqrt{\tilde{g}p\bar{p}\log(r)} = \Theta\big(\sqrt{\tilde{g}p}\big) = o(\tilde{g}p)$ and hence the claim (6) is proved.

V. Converse for Erdös-Rényi model

We complete the proof of Theorem 1 by deriving the lower bound on the best average communication load for Erdös-Rényi graph. Let $\mathcal{G}$ be an ER(n,p) random graph and consider a subgraph and computation allocation $A = (\mathcal{M}, \mathcal{R})$, where $\sum_{k=1}^{K} |\mathcal{M}_k| = rn$ and $|\mathcal{R}_k| = \frac{n}{K}$, for all $k \in [K]$. We denote the number of files that are Mapped exclusively at j servers under Map assignment $\mathcal{M}$, as $a^j_{\mathcal{M}}$, for all $j \in [K]$. The following lemma holds (the proof is available in [22]).

Lemma 2: $\mathbb{E}_{\mathcal{G}}[L_A(r,\mathcal{G})] \ge p\sum_{j=1}^{K} \frac{a^j_{\mathcal{M}}}{n}\frac{K-j}{Kj}$.

Proof of Converse for Theorem 1. First, we can bound the best average communication load as

$$L^*(r) \ge \inf_{A:|\mathcal{M}_1|+\cdots+|\mathcal{M}_K|=rn} \mathbb{E}_{\mathcal{G}}[L_A(r,\mathcal{G})]$$

$$\ge \inf_{A:|\mathcal{M}_1|+\cdots+|\mathcal{M}_K|=rn} p\sum_{j=1}^{K} \frac{a^j_{\mathcal{M}}}{n}\frac{K-j}{Kj}.$$

Additionally, for any subgraph allocation with computation load r, we have the following equations:

$$\sum_{j=1}^{K} a^j_{\mathcal{M}} = n, \ \sum_{j=1}^{K} j a^j_{\mathcal{M}} = rn. \tag{7}$$

By convexity of $\frac{K-j}{Kj}$ in j and (7), the converse is proved.

References

[1] G. Malewicz, M. H. Austern, A. J. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski, "Pregel: a system for large-scale graph processing," *SIGMOD*, 2010.

[2] Y. Low, D. Bickson, J. Gonzalez, C. Guestrin, A. Kyrola, and J. M. Hellerstein, "Distributed GraphLab: a framework for machine learning and data mining in the cloud," *VLDB*, 2012.

[3] R. R. McCune, T. Weninger, and G. Madey, "Thinking like a vertex: a survey of vertex-centric frameworks for large-scale distributed graph processing," *ACM Computing Surveys*, 2015.

[4] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly, "Dryad: distributed data-parallel programs from sequential building blocks," *EuroSys*, 2007.

[5] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," *Communications of the ACM*, 2008.

[6] A. Lumsdaine, D. Gregor, B. Hendrickson, and J. Berry, "Challenges in parallel graph processing," *Parallel Processing Letters*, 2007.

[7] R. Chen, X. Ding, P. Wang, H. Chen, B. Zang, and H. Guan, "Computation and communication efficient graph processing with distributed immutable view," *HPDC*, 2014.

[8] 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*, 2017.

[9] Y. H. Ezzeldin, M. Karmoose, and C. Fragouli, "Communication vs distributed computation: an alternative trade-off curve," *arXiv preprint arXiv:1705.08966*, 2017.

[10] K. Lee, M. Lam, R. Pedarsani, D. Papailiopoulos, and K. Ramchandran, "Speeding up distributed machine learning using codes," *IEEE Transactions on Information Theory*, 2017.

[11] M. A. Attia and R. Tandon, "Information theoretic limits of data shuffling for distributed learning," *GLOBECOM*, 2016.

[12] S. Li, S. Supittayapornpong, M. A. Maddah-Ali, and S. Avestimehr, "Coded terasort," *IPDPSW*, 2017.

[13] S. Li, M. A. Maddah-Ali, and A. S. Avestimehr, "Coding for distributed fog computing," *IEEE Communications Magazine*, 2017.

[14] S. Dutta, V. Cadambe, and P. Grover, "Short-dot: Computing large linear transforms distributedly using coded short dot products," *NIPS*, 2016.

[15] R. Tandon, Q. Lei, A. G. Dimakis, and N. Karampatziakis, "Gradient coding: Avoiding stragglers in distributed learning," *ICML*, 2017.

[16] Q. Yu, M. Maddah-Ali, and S. Avestimehr, "Polynomial codes: an optimal design for high-dimensional coded matrix multiplication," *NIPS*, 2017.

[17] K. Lee, C. Suh, and K. Ramchandran, "High-dimensional coded matrix multiplication," *ISIT*, 2017.

[18] A. Reisizadeh, S. Prakash, R. Pedarsani, and S. Avestimehr, "Coded computation over heterogeneous clusters," *ISIT*, 2017.

[19] J. Lin and M. Schatz, "Design patterns for efficient graph algorithms in mapreduce," *MLG Workshop*, 2010.

[20] W. Xing and A. Ghorbani, "Weighted pagerank algorithm," *CNSR*, 2004.

[21] E. Dahlhaus, D. S. Johnson, C. H. Papadimitriou, P. D. Seymour, and M. Yannakakis, "The complexity of multiway cuts," *STOC*, 1992.

[22] S. Prakash, A. Reisizadeh, R. Pedarsani, and S. Avestimehr, "Coded computing for distributed graph analytics," *arXiv preprint arXiv:1801.05522*, 2018.