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

To combat the growing demands for efficient processing of large scale graph-structured datasets, many distributed graph computing systems have been developed recently. As these systems require many messages to be exchanged among computing machines at each step of the computation, communication bandwidth has been observed to be 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 a coded scheme that enables an (asymptotically) inverse-linear trade-off between computation load and average communication load for Erdős–Rényi graphs. The proposed scheme is shown to be optimal asymptotically as the graph size increases. For finite-size graphs, 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) a multiplicative factor of $r$. Furthermore, we generalize our results to three other random graph models – random bi-partite model, stochastic block model and power law model. In particular, we prove that our schemes asymptotically enable an inverse-linear trade-off between computation and communication loads in distributed graph processing for these popular graph models as well. We finally carry out experiments over Amazon EC2 clusters to practically demonstrate the impact of our coded schemes. In particular, for the popular PageRank algorithm, we demonstrate that our proposed coding scheme results in speedups of up to 47% in comparison to the baseline approach.

I. INTRODUCTION

Graphs are widely used to identify and incorporate the relationship patterns and anomalies inherent in real-life datasets. Their adoption in a wide range of problems such as web search, intelligent recommendation systems, social behaviours and natural language processing have made graph processing indispensable. Their growing scale and importance have prompted the development of various large-scale distributed graph processing frameworks, such as Pregel [2], PowerGraph [3] and GraphLab [4]. The underlying theme in these systems is the “think like a vertex” approach [5] where the computation at each vertex requires only the data available in the neighborhood of the vertex (see Figure 1). This approach significantly improves performance in comparison to general-purpose distributed data processing systems (e.g., Dryad [6], MapReduce [7]), 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 [8], accounting for more than 50% of the overall execution time in representative cases [9]. We develop a new framework 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 graph structure 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) [10]. CDC achieves the communication bandwidth gain $r$, when each Map computation is carefully repeated at $r$ servers. In particular, CDC aims to compute $Q$ output functions of $n$ input files using $K$ distributed computing machines. In the Map phase, each input file is coded messages that simultaneously satisfy the data demand of multiple computing machines. In the Reduce phase, the coded messages are decoded to recover the output functions. When $Q > K$, the communication load is reduced by a factor of $r$.

Fig. 1: An overview of “think like a vertex” approach taken in common parallel graph computing frameworks, in which the intermediate computations only depend on the neighbors at each node. For two graph nodes, their scope of computations have been highlighted in this figure.

This work is supported by a startup grant for Ramtin Pedarsani and NSF grants CCF-1408639, CCF-1755808, 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.

A part of this work was presented at IEEE International Symposium on Information Theory, 2018 [1]
Mapped to $Q$ intermediate values each corresponding to one of the output functions. The injected redundancy provides coded multicast opportunities in the Shuffle phase where servers exchange coded messages that are simultaneously useful for multiple servers. Each server then decodes the received messages and reduces the output functions assigned to it. However, the general framework does not incorporate the heterogeneity in the file requirements by the Reducers, as each Reducer is assumed to need intermediate values corresponding to all input files.

Moving from the general MapReduce framework to graph analytics, the key challenge is that the computation associated with each vertex highly depends on the graph structure. In particular, computation at each vertex requires data only from the neighboring vertices, while in the general MapReduce framework, each computation needs all the input files (which in graph analytics corresponds to a complete graph). This asymmetry in the data requirements of the computations is the main challenge in developing efficient subgraph and computation allocations and Shuffling schemes.

As the main contribution of this paper, we consider the Erdös-Rényi (ER) model for random graphs, and develop an asymptotically optimal coded computing scheme for the distributed graph analytics problem. We define computation load $r$ as the average number of workers where the Mapping of a vertex takes place. For a given computation load $r$, we show through achievability-converse arguments that in the regime of interest, the minimum average normalized communication load is $L^*(r) \approx \frac{1}{2} L_{UC}(r)$, where $L_{UC}(r) = p(1 - \frac{r}{K})$ denotes the average communication load for a baseline uncoded scheme with computation load $r$, $p$ is the edge probability in the ER graph of size $n$, and $K$ denotes the number of servers.

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. Within each multicast group, each worker communicates a coded message which is generated using careful alignment of the intermediate values that the worker needs to communicate to all the remaining members of the multicast group. Using these coded transmissions, each machine retrieves the missing intermediate values required for its computation Reduction using the available intermediate values from Mapping and the received messages.

We also demonstrate that the proposed scheme is asymptotically optimal for the ER model. To this end, we derive an information-theoretic lower bound on the average communication load for any allocation with computation load $r$. To derive the lower bound, using induction we derive a lower bound on the expected load induced by any subset of servers. Therefore, the computation-communication trade-off for the ER model is asymptotically characterized as $L^*(r) \approx \frac{1}{2} p(1 - \frac{r}{K})$, for any integer computation load $1 \leq r \leq K$. Comparing with the expected uncoded load $L_{UC}(r) = p(1 - \frac{r}{K})$, this shows that the asymptotic communication gain $r$ obtained by the proposed scheme for the ER model is optimal and can not be improved.

To extend our results to more graph models, we leverage the coded scheme for the ER model and develop new coded schemes for random bi-partite model, stochastic block model and the power law model. The proposed schemes demonstrate that an inverse-linear trade-off between computation and communication loads in distributed graph processing exists for these graph models as well.

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 a baseline uncoded scheme described later. Furthermore, we implement the PageRank algorithm over Amazon EC2 machines, and demonstrate how our proposed scheme can be applied in practice. In these experiments, speedups of up to 47% are achieved in comparison to the baseline implementation, as demonstrated in Figure 2. The details of implementation are provided in Section VI.

**Related Work.** A number of coding theoretic strategies have been recently proposed to mitigate the bottlenecks in large scale distributed computing [10], [11]. Several generalizations to the Coded Distributed Computing (CDC) technique proposed in [10] have been developed. The authors in [12] extend CDC to wireless scenarios. The work in [13] extends CDC to multistage dataflows. An alternative trade-off between communication and distributed computation has been explored in [14] for MapReduce framework under predetermined storage constraints. Coding using resolvable designs has been proposed in [15]. [16] extends CDC to heterogeneous computing environments. The work in [17] proposes coding scheme for reducing communication load for computations associated with linear aggregation of intermediate results in the final Reduce stage. The key difference between our framework and each of these works is that general MapReduce computations over graphs have heterogeneity in the data requirements for the Reduce functions associated with the vertices. Other notable works that deal with communication bottleneck in distributed computation include [18]–[20], where the authors propose techniques to reduce communication load in data shuffling in distributed learning.

Apart from communication bottleneck, various coding theoretic works have been proposed to tackle the straggler bottleneck

![Fig. 2: Demonstrating the impact of Coded PageRank in speeding up computations in an Amazon EC2 cluster consisting of 16 machines when the computation load is 4 (details of implementation are provided in Section VI). One can observe that the Shuffle phase is the major component of the overall execution time, and the proposed coding scheme slashes the overall execution time by shortening the Shuffle phase (i.e., reducing the communication load) at the expense of increasing the Map phase (i.e., increasing the computation load).](image)
Stragglers are slow processors that have significantly larger delay for completing their computational task, thus slowing down the overall job execution in distributed computation. The first paper in this line of research proposed erasure correcting codes for straggler mitigation in linear computation [11]. The work in [21] explores the potential of the multicore nature of computing machines, while [23] extends the straggler mitigation for the matrix vector problem in wireless scenarios. Redundant short dot products for matrix multiplication with long vector has been proposed in [22]. The authors in [23] propose Heterogeneous Coded Matrix Multiplication (HCMM) scheme for matrix-vector multiplication in heterogeneous scenarios. In [24], the authors propose gradient coding schemes for straggler mitigation in distributed batch gradient descent. Works in [25] and [26] develop coding schemes for computing high-dimensional matrix-matrix multiplication. A Coupon Collector based straggler mitigation scheme for batch gradient descent has been proposed in [27]. Other notable schemes include Substitute decoding for coded iterative computing [28], coding for sparse matrix multiplications [29–31], approximate gradient coding [32], efficient gradient computation tackling both straggler and communication load [33], a unified coding scheme for distributed matrix multiplication [34], logistic regression with unreliable components [35], among others.

**Notation.** We denote by $|n|$ 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 $c_1, c_2$ and $n_0 \in \mathbb{N}$ such that $c_1 \leq f(n)/g(n) \leq c_2$ for every $n \geq n_0$, and $f = o(g)$ if $f(n)/g(n)$ converges to 0 as $n$ goes to infinity. We define $f = \omega(g)$, 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

We now describe the setting and formulate the distributed graph analytics problem. In particular, we specify our computation model, distributed implementation model and our problem formulation based on random graphs.

A. Computation Model

We consider an undirected graph $G = (V, E)$ where $V = |n|$ and $E = \{(i, j) : i, j \in V\}$ denote the set of graph vertices and the set of edges respectively. A binary file $w_i \in F_{2^F}$ of size $F \in \mathbb{N}$ is associated with each graph vertex $i \in V$. We denote by $W = \{w_i : i \in V\}$ the set of files associated with all vertices in the graph. The neighborhood of vertex $i$ is denoted by $N(i) = \{j \in V : (j, i) \in E\}$ and the set of files in the neighborhood of $i$ is represented by $W_{N(i)} = \{w_j : j \in N(i)\}$. In general, $G$ can have self-loops, i.e., vertex $i$ can be contained in $N(i)$. Furthermore, a computation is associated with each vertex $i \in V$ as follows:

$$\phi_i : F_{2^F} \rightarrow F_{2^F},$$

where $\phi_i(\cdot)$ outputs the input files in $W_{N(i)}$ to a length $B$ binary stream $o_i = \phi_i(W_{N(i)}).$

The computation $\phi_i(\cdot)$ can be represented as a MapReduce computation:

$$\phi_i(W_{N(i)}) = h_i((g_{i,j}(w_j) : w_j \in W_{N(i)})),$$

where the Map function $g_{i,j} : F_{2^F} \rightarrow F_{2^F}$ Maps file $w_j$ to a length $T$ binary intermediate value $v_{i,j} = g_{i,j}(w_j), \forall i \in N(j)$. The Reduce function $h_i : F_{2^F}^{2^{|N(i)|}} \rightarrow F_{2^F}$ Reduces the intermediate values associated with the output function $\phi_i(\cdot)$ into the final output value $o_i = h_i(\{v_{i,j} : j \in N(i)\})$.

We illustrate our computation model through an example. Fig. 3(a) illustrates a graph with $n = 6$ vertices, where each vertex is associated with a file, and Fig. 3(b) illustrates the corresponding MapReduce computations.

Iterative graph algorithms can be expressed in the MapReduce computation framework described above [43]. For brevity, we present two popular graph algorithms and describe how they can be expressed in the proposed computation framework.

**Example 1. PageRank** [44], [45] is a popular algorithm to measure the importance of the vertices in a webgraph based on the underlying hyperlink structure. In particular, the algorithm computes the likelihood that a random surfer would visit a page. Mathematically, the rank of a vertex $i$ satisfies the following relation:

$$\Pi(i) = (1 - d) \sum_{j \in N(i)} \Pi(j) \mathbb{P}(j \rightarrow i) + d \frac{1}{|V|},$$

where $(1 - d)$ is referred to as the damping factor, $\Pi(i)$ denotes the likelihood that the random surfer will arrive at vertex $i$, $|V|$ is the total number of vertices in the webgraph, and $\mathbb{P}(j \rightarrow i)$ is the transition probability from vertex $j$ to vertex $i$. The computation can be carried out iteratively as follows:

$$\Pi^k(i) = (1 - d) \sum_{j \in N(i)} \Pi^{k-1}(j) \mathbb{P}(j \rightarrow i) + d \frac{1}{|V|},$$

where $k$ and $k - 1$ are respectively the current and previous iterations and $\Pi^0(i) = \frac{1}{|V|}$ for all $i \in V$ and $k = 1, 2, \ldots$. The number of iterations depends on the stopping criterion for the algorithm. Usually, the algorithm is stopped when the change in the PageRank mass of each vertex is less than a pre-defined tolerance. The rank update at each vertex can be decomposed into Map and Reduce functions for each iteration $k$. For a given vertex $i$ and iteration $k$, let $\mathcal{W}^k_{N(i)} = \{\Pi^{k-1}(j), j \in N(i)\}$, and $\phi_i^k(\mathcal{W}^k_{N(i)}(i)) = (1 - d) \sum_{j \in N(i)} \Pi^{k-1}(j) \mathbb{P}(j \rightarrow i) + d \frac{1}{|V|}$. The Mapper $g_{i,j}(\cdot)$ Maps the file $w^k_j = \Pi^{k-1}(j)$ to the
Example 2. Single-source shortest path is one of the most studied problems in graph theory. The task here is to find the shortest path to each vertex \( v_i \) from the source \( s \). A sub-problem for this task is to compute the distance of each vertex \( v_i \) from the source vertex \( s \), where distance \( D(i) \) is the length of the shortest path from \( s \) to \( i \). This can be carried out iteratively in parallel. First, initialize \( D^0(s) = 0 \) and \( D^0(i) = +\infty, \forall i \in V \setminus \{s\} \). Subsequently, each vertex \( i \) is updated as follows at each iteration \( k \):

$$
D^k(i) = \min_{j \in \mathcal{N}(i)} (D^{k-1}(j) + t(j, i)),
$$

where \( t(j, i) \) is the weight of the edge \((j, i)\). The algorithm is stopped when the change in the distance value for each vertex is within a pre-defined tolerance. The distance computation for each vertex at iteration \( k \) can be decomposed into Map and Reduce computations. Particularly, for each vertex \( i \) and iteration \( k \), let \( W^k_{\mathcal{N}(i)} = \{D^{k-1}(j), j \in \mathcal{N}(i)\} \), and \( \phi^k(W^k_{\mathcal{N}(i)}) = \min_{j \in \mathcal{N}(i)} (D^{k-1}(j) + t(j, i)) \). The Mapper \( g_{i,j}(\cdot) \) Maps the file \( w^k_j = D^{k-1}(j) \) to the intermediate values \( v^k_{i,j} = g_{i,j}(w^k_j) = D^{k-1}(j) + t(j, i) \) for all neighboring vertices \( i \in \mathcal{N}(j) \). Using the intermediate values from the Map computations, the Reducer \( h_i(\cdot) \) computes \( i \)'s updated distance value as \( D^k(i) = h_i(\{v^k_{i,j} : j \in \mathcal{N}(i)\}) = \min_{j \in \mathcal{N}(i)} v^k_{i,j} \).

### B. Distributed Implementation

We now describe our model for distributed implementation of the MapReduce computations described above. Naturally, message passing would be needed in order to exchange intermediate values between machines after the Map stage, so that each machine has all the intermediate values needed for executing the Reduce computations allocated to it. We consider a network consisting of \( K \) machines that are connected to each other through a shared network, such that at any given time, only one machine is allowed to use the network. As the first step, a distributed implementation consists of allocating a subgraph to each machine.

**Subgraph Allocation and Map Computation:** We denote the subgraph that is allocated to each machine \( k \in [K] \) by \( \mathcal{M}_k \subseteq V \). Machine \( k \) will then store all the 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. Additionally, we allow redundant computations, i.e., each file can be Mapped by more than one machine. The goal is to trade the computational resources in order to reduce the communication load in the Shuffle phase. More formally, we define the computation load as follows.

**Definition 1** (Computation Load). For a subgraph allocation, \((\mathcal{M}_1, \cdots, \mathcal{M}_K)\), the computation load, \( r \in [K] \), is defined as

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

**Remark 1.** For a desired computation load \( r \), we assume that each server maps an equal number of \( r \frac{n}{K} \) vertices of the graph.
**Reduce Computation Allocation:** A Reducer is associated with each vertex of the graph $G$. We use $R_k \subseteq V$ to denote the set of vertices whose Reduce computations are assigned to machine $k$. The set of Reduce computations is partitioned into $K$ equal parts and each part is carried out exclusively with one machine, i.e., $\bigcup_{k=1}^{K} R_k = V$ and $R_m \cap R_n = \phi$ for $m, n \in [K], m \neq n$. Therefore, $|R_k| = \frac{m}{K}, \forall k \in [K]$.

Given a subgraph and computation allocation to servers, denoted by $A = (M, R)$ where $M = (M_1, \cdots, M_K)$ and $R = (R_1, \cdots, R_K)$, the computation proceeds distributedly in the following three phases.

**Map phase:** Each server first maps the vertices associated with all the Reduce functions that are allocated to it (i.e. for the vertices in $R_k$). The computation load 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.

$$L \triangleq \frac{\sum_{k=1}^{K} c_k}{n^2 T}. \quad (7)$$

**Reduce phase:** Server $k$ 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 (i.e. for the vertices in $R_k$), and then calculates $o_i = h_i(\{v_{i,j} : j \in \mathcal{N}(i)\})$ for all $i \in R_k$.

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

**C. Problem Formulation**

Our broader goal is to develop optimal schemes for allocation of subgraphs and computations to servers, and the optimal coding schemes for Shuffling in order to minimize the communication load for an allowed computation load $r$. 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 [40]. Hence, we restrict our attention to random graphs and focus on the average communication load.

We consider a random undirected graph $\mathcal{G} = (V, E)$, where edges independently exist with probability $\mathbb{P}[(i, j) \in E]$ for all $i, j \in V$. Let $A^{(r)}$ 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 $A \in A^{(r)}$, we denote by $L_A(r, G)$ the minimum (normalized) communication load (as defined in Definition 2) over all feasible Shuffling coding schemes that enable each machine to compute all the Reduce functions assigned to it.

We now formally define our problem as follows.

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

$$L^*(r) \triangleq \inf_{A \in A^{(r)}} \mathbb{E}_G[L_A(r, \mathcal{G})]. \quad (8)$$

**Remark 2.** For $r \geq K$, $L^*(r)$ is trivially 0 as each vertex can be mapped at each server, so all the intermediate values associated with the Reducers of any server is available at the server.

**Remark 3.** 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 an efficient Shuffle scheme.

**III. MAIN RESULTS**

In this section, we present the main results of our work. Our first result is the characterization of $L^*(r)$ (defined in (8)) for Erdős-Rényi model that is defined below.
Erdős-Rényi Model: 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. In Fig. 4(b), each cross-edge exists with a given probability $q$. In Fig. 4(c), each intra-cluster edge exists with a given probability $p$ and each cross-edge exists with a given probability $q$. In Fig. 4(d), expected degree of each node follows a power law distribution with exponent $\gamma$.

Theorem 1. For Erdős-Rényi model ER($n, p$) with $p = \omega\left(\frac{1}{n}\right)$, we have

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

Remark 4. Theorem 1 reveals an interesting inverse linear trade-off between computation and communication. In particular, the scheme 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 $p/(1 - r/K)$. 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. 5).

Remark 5. Achievability of Theorem 1 is proved in Section V where we provide subgraph and computation allocations followed by the code design for Shuffling. The main idea is to leverage the coded multicast opportunities offered by the injected redundancy and create coded messages which simultaneously satisfy the data demand of multiple servers. Careful combination of available intermediate values during the Shuffle phase benefits from the missing graph connections by aligning the present intermediate values. Conversely, Theorem 1 demonstrates that the asymptotic bandwidth gain $r$ achieved by the proposed scheme is optimal and can not be improved. Proof of converse is provided in Section V where by induction we derive information-theoretic lower bounds on the average communication load required by any subset of servers.

Furthermore, we develop subgraph allocation and computation allocation schemes along with coded Shuffling schemes for three popular random graph models which are described below:

Random Bi-partite Model: Denoted by RB($n_1, n_2, q$), this model consists of graphs with two disjoint clusters of sizes $n_1$ and $n_2$ in which each inter-cluster edge exists with probability $q \in (0, 1]$, independently of other edges. In Fig. 4(b), each cross-edge exists with a given probability $q$. In Fig. 4(c), each intra-cluster edge exists with a given probability $p$ and each inter-cluster edge exists with probability $q$, $0 < q < p \leq 1$, all independent of each other (Fig. 4(c)).

Stochastic Block Model: Denoted by SBM($n_1, n_2, p, q$), this model consists of graphs with two disjoint clusters of sizes $n_1$ and $n_2$ such that each intra-cluster edge exists with probability $p$ and each inter-cluster edge exists with probability $q$, $0 < q < p \leq 1$, all independent of each other (Fig. 4(c)).

Power Law Model: Denoted by PL($n, \gamma, \rho$), this model consists of graphs of size $n$ in which degrees are i.i.d random variables drawn from a power law distribution with exponent $\gamma$ and edge probabilities are $\rho$-proportional to product of the degrees of the two end vertices (Fig. 4(d)).

The following theorems provide the achievability results for RB, SBM and PL models and are proved in the appendices.

Theorem 2. For random bi-partite model RB($n_1, n_2, q$) with $n = n_1 + n_2$, $n_1 = \Theta(n)$, $n_2 = \Theta(n)$ and $q = \omega(\frac{1}{n^2})$, we have

$$\limsup_{n \to \infty} \frac{L^*(r)}{2q(n_1 + n_2)/(n_1 + n_2)} \leq \frac{1}{r} \left(1 - \frac{r}{K}\right).$$

Theorem 3. For stochastic block model graph SBM($n_1, n_2, p, q$) with $n = n_1 + n_2$, $n_1 = \Theta(n)$, $n_2 = \Theta(n)$, and $p = \omega(1/n^2)$, we have

$$\limsup_{n \to \infty} \frac{L^*(r)}{p} = \frac{1}{r} \left(1 - \frac{r}{K}\right).$$
For each \( k \), each intermediate value \( v_{i,j} \) for \( i,j \in E, i \in R_k, j \in \cap_{k' \in S \setminus \{k\}} M_{k'} \), is evenly split into \( r \) segments \( v_{i,j}^{(1)}, \ldots, v_{i,j}^{(r)} \), each of size \( \frac{L}{r} \) bits. Each segment is associated with a distinct server in \( S \setminus \{k\} \). Therefore, \( Z_{S \setminus \{k\}}^k \) is evenly partitioned to \( r \) sets, which are denoted by \( Z_{S \setminus \{k\},s}^k \) for \( s \in S \setminus \{k\} \). Depending on the realization of the graph, the maximum possible size of \( Z_{S \setminus \{k\}}^k \) is \( \tilde{g} = g \frac{p}{n^2 K} = \frac{n^2}{K(\frac{r}{2})} = \Theta(n^2) \). Each server \( s \) 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 \( Z_{S \setminus \{k\},s}^k \), where \( k \in S \setminus \{s\} \) (Fig. 6). 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 $S \subseteq [K]$ with $|S| = r + 1$.

After the Shuffle phase, all but one intermediate values contributed in each coded message are locally available. Moreover, all possible subsets of multicast servers have sent their corresponding messages. Therefore, each server can recover all of the intermediate values associated with its assigned set of Reduce functions using the received coded messages and the locally computed intermediate values.

**Remark 7.** 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 theReducers 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. 3(a), Fig. 3(b) summarizes the subgraph and computation allocations followed from the proposed scheme. Consider the set $S = [K]$ of size $r + 1 = 3$. Every intermediate value in $Z^{3}_{\{1,2\}} = \{v_{5,1}, v_{6,2}\}$ 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 $X_1 = \{v_{5,1}^{(1)} + v_{4,3}^{(1)} + v_{3,4}^{(1)} + v_{2,6}^{(1)}\}$, $X_2 = \{v_{5,1}^{(2)} + v_{4,3}^{(1)} + v_{2,6}^{(1)}\}$, and $X_3 = \{v_{4,3}^{(2)} + v_{3,4}^{(2)} + v_{2,6}^{(2)}\}$, 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 $v_{3,4}$ are available locally. Server 3 can recover $v_{5,1}^{(1)}$ and $v_{5,1}^{(2)}$ from $v_{5,1}^{(1)} + v_{4,3}^{(1)}$ and $v_{5,1}^{(2)} + v_{3,4}^{(2)}$, respectively. Therefore, the overall uncoded communication load $\frac{6}{\gamma}$ is reduced to coded load $\frac{3}{\gamma}$.

### B. Proof of Achievability

We first define the average normalized communication loads for our scheme as follows. For graph $G$, the proposed allocation $A_C \in A(r)$, and the proposed coded and uncoded Shuffling scheme, we denote the normalized coded and uncoded communication loads by $L^C_{A_C}(r, G)$ and $L^U_{A_C}(r, G)$, respectively. The average normalized coded and uncoded communication loads will then be $L^C_{A_C} = E_G[L^C_{A_C}(r, G)]$ and $L^U_{A_C} = E_G[L^U_{A_C}(r, G)]$, respectively.

We now apply the proposed coded scheme to graph $G$ and compute the induced average coded load. Without loss of generality, we analyze our algorithm by a generic argument for servers $S = \{s_1, \cdots, s_{r+1}\}$ 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 $X^1, \cdots, X^g$ which are exclusively used for servers $s_2, \cdots, s_{r+1}$. For each non-empty column $j \in [\tilde{g}]$, $X^j$ 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 $Z^{1}_{\{s_1+1, s_1\}} = \{v^{(1)}_{\alpha(i,j)} : j \in [\tilde{g}]\}$ and $i \in [r]$ (Fig. 6).

**Fig. 6:** Creating coded messages by aligning present intermediate values.

Let $\text{Bern}(p)$ random variable $E_{\alpha(i,j)}$ indicate the existence of the edge $\alpha(i, j) \in V \times V$, i.e. $E_{\alpha(i,j)} = 1$, if $\alpha(i, j) \in E$, and $E_{\alpha(i,j)} = 0$, otherwise. Clearly, for all vertices $i, j, t, u \in V$, $E_{\alpha(i,j)}$ 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 \( \text{Bern}(p) \) random variables. Note that \( P_i \)'s are not independent in general. By careful alignment of present intermediate values (Fig. 6), \( s_1 \) broadcasts \( Q \) coded messages each of size \( \frac{r}{p} \) bits, where \( Q = \max_{i \in [r]} P_i \). Thus, the total coded communication load sent from server \( s_1 \) exclusively for servers \( s_2, \ldots, s_{r+1} \) is \( \frac{r}{p} Q \) bits. By similar arguments for other sets of servers, we can characterize the average coded communication load of the proposed scheme as
\[
\bar{L}_{\text{Ac}} = \frac{1}{r \eta^2} K \left( \frac{K-1}{r} \right) \mathbb{E}[Q].
\]
The following lemma asymptotically upper bounds \( \mathbb{E}[Q] \) and the proof is provided in Section IV-C.

**Lemma 1.** For \( \text{ER}(n, p) \) graphs with \( p = \omega \left( \frac{1}{n^2} \right) \), we have
\[
\mathbb{E}[Q] \leq \tilde{p} \tilde{g} + o(p \tilde{g}).
\]

Putting (15) and Lemma 1 together, we have
\[
L^*(r) \leq \bar{L}_{\text{Ac}} \leq \frac{1}{r \eta^2} \left(1 - \frac{r}{K} \right) + o(p),
\]
hence the achievability 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 \( \bar{L}_{\text{UC}} = p \left(1 - \frac{r}{K} \right) \), which implies that our scheme achieves an asymptotic gain \( r \).

### C. Proof of Lemma 1

Before proving Lemma 1, we first present the following lemma that will be used in our proof.

**Lemma 2.** For random variables \( \{ P_i \}_{i=1}^r \) defined in (14), their moment generating functions for \( s' > 0 \) can be bounded by
\[
\mathbb{E} \left[ e^{s' P_i} \right] \leq (pe^{s' + 1} - 1)^{\tilde{g} \bar{g}/2}.
\]

**Proof.** Consider a generic random variable of the form (14)
\[
P = \sum_{j=1}^{\tilde{g}} E_j,
\]
where \( E_j \)'s are \( \text{Bern}(p) \) and possibly dependent. However, although \( E_j \)'s may not be all independent, but dependency is restricted to pairs of \( E_j \)'s. In other words, for all \( 1 \leq j \leq \tilde{g} \), \( E_j \) is either independent of all \( E_{[\tilde{g}] \setminus \{ j \}} \), or is equal to \( E_{\ell} \) for some \( \ell \in [\tilde{g}] \setminus \{ j \} \) and independent of all \( E_{[\tilde{g}] \setminus \{ j, \ell \}} \). By merging dependent pairs, we can write
\[
P = \sum_{j=1}^{\tilde{g} - J} F_j,
\]
where
(1) \( F_j \)'s are independent,
(2) \( \tilde{g} - 2J \) of \( F_j \)'s are \( \text{Bern}(p) \),
(3) \( J \) of \( F_j \)'s are \( 2 \times \text{Bern}(p) \),
for some integer \( 0 \leq J \leq \left\lfloor \frac{\tilde{g}}{2} \right\rfloor \). Now, we can bound the moment generating function of \( P \). For \( s' > 0 \),
\[
\mathbb{E} \left[ e^{s' P} \right] = \mathbb{E} \left[ e^{s' \sum_{j=1}^{\tilde{g} - J} F_j} \right]
= \prod_{j=1}^{\tilde{g} - J} \mathbb{E} \left[ e^{s' F_j} \right]
= (pe^{s' + 1} - 1)^{\tilde{g} - 2J} (pe^{s' + 1} - 1)^J
\]
\[
= \left( (pe^{s' + 1} - 1)^2 \right)^{\tilde{g} - 2J / 2} (pe^{s' + 1} - 1)^J
\]
\[
\leq (pe^{s' + 1} - 1)^{\tilde{g} - 2J / 2} (pe^{s' + 1} - 1)^J
\]
where inequality (a) is obtained using Lemma 4 (proof available in Appendix D).
Therefore, we employ the techniques employed in [10] to prove the optimality of CDC scheme. More precisely, we complete the proof of Theorem 1 and consider a subgraph and computation allocation $\tilde{\omega}$. Recall that

$$ \sum_{i=1}^{r} e^{s_i P_i} \leq \sum_{i=1}^{r} \mathbb{E}[e^{s_i P_i}] $$

where the last inequality follows from Lemma 2. Taking logarithm from both sides yields

$$ \mathbb{E}[Q] \leq \frac{1}{s_i} \log(r) + \frac{\tilde{g}}{s_i} \log(pe^{s_i} + 1 - p). $$

Let us substitute $s = 2s_i$ in (31). Then,

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

for any $s > 0$. Let $\tilde{p} = 1 - p$ and pick

$$ s_* = 2 \sqrt{\frac{\log(r)}{\tilde{g} \tilde{p} p}}. $$

We proceed with evaluation of the right hand side (RHS) of (32) at $s = s_*$. We first recall the following Taylor series

$$ \log(1 + x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \cdots, \quad \text{for } x \in (-1, 1], $$

$$ e^x = 1 + x + \frac{x^2}{2} + \frac{x^3}{3!} + \cdots, \quad \text{for } x \in \mathbb{R}. $$

Let $x = p(e^{s_*} - 1)$. It is easy to check that for $p = \omega(\frac{1}{n^2})$, we have $x \to 0$ and $s_* \to 0$ as $n \to \infty$. Therefore, for $n \to \infty$ we can write

$$ \log(pe^{s_*} + 1 - p) = \log(x + 1) = x - \frac{x^2}{2} + \frac{x^3}{3} - \cdots $$

$$ = p(e^{s_*} - 1) - \frac{p^2(e^{s_*} - 1)^2}{2} + \frac{p^3(e^{s_*} - 1)^3}{3} - \cdots $$

$$ = p(s_* + \frac{s_*^2}{2} + \frac{s_*^3}{3!} + \cdots) - \frac{p^2}{2}(s_* + \frac{s_*^2}{2} + \frac{s_*^3}{3!} + \cdots + s_*^3 + \cdots)^2 + \frac{p^3}{3}(s_* + s_*^2 + \frac{s_*^3}{3!} + \cdots)^3 - \cdots $$

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

Putting everything together, we have

$$ \mathbb{E}[Q] \leq \frac{1}{s_*} \log(r^2) + \frac{\tilde{g}}{s_*} \log(pe^{s_*} + 1 - p) $$

$$ = \frac{1}{s_*} \log(r^2) + \frac{\tilde{g}}{s_*} (ps_* + \frac{\tilde{p} p}{2} s_*^2 + o(ps_*^2)) $$

$$ = \frac{1}{s_*} \log(r^2) + \tilde{g} \tilde{p} + \frac{\tilde{p} \tilde{p}}{2} s_* + o(\tilde{g} ps_*), $$

$$ \tilde{g} \tilde{p} + 2 \sqrt{\tilde{g} \tilde{p} p \log(r)} + o \left( \sqrt{\tilde{g} \tilde{p}} \right). $$

Recall that $\tilde{g} = \frac{n^2}{K(\tilde{s})} = \Theta(n^2)$. Therefore, for $p = \omega(\frac{1}{n^2})$ we have $\tilde{g} \tilde{p} = \omega(1)$ and thus $\sqrt{\tilde{g} \tilde{p} p \log(r)} = \Theta \left( \sqrt{\tilde{g} \tilde{p}} \right) = o(\tilde{g} \tilde{p})$. Therefore, $\mathbb{E}[Q] \leq p\tilde{g} + o(p\tilde{g})$, as $n \to \infty$.

V. CONVERSE FOR ERDÖS-RÉNYI MODEL

In this section, we prove the asymptotic optimality of our proposed coded scheme for ER model by applying similar techniques employed in [10] to prove the optimality of CDC scheme. More precisely, we complete the proof of Theorem 1 by deriving the lower bound on the best average communication load for ER model. Let $G$ be an ER($n, p$) random graph and consider a subgraph and computation allocation $A = (M, R) \in A(r)$, where $\sum_{k=1}^{K} |M_k| = rn$ and $|R_k| = \frac{r}{K}$, for all $k \in [K]$. We denote the number of files that are Mapped at $j$ vertices under Map assignment $M$, as $\alpha_{M, j}$, for all $j \in [K]$. The following lemma holds.
Lemma 3. \( \mathbb{E}_G[L_A(r, G)] \geq p \sum_{j=1}^{K} a_{j,M}^i \frac{K-j}{K} \).

Proof. We let intermediate values \( v_{i,j} \) be realizations of random variables \( V_{i,j} \), uniformly distributed over \( \mathbb{F}_{2^n} \). For a random graph \( G = (V, \mathcal{E}) \) and subsets \( I, J \subseteq V = [n] \), define \( V^G_{i,j} = \{ v_{i,j} : (i, j) \in \mathcal{E}, i \in I, j \in J \} \) as the set of present intermediate values in graph \( G \) corresponding to Reducers in \( I \) and Mappers in \( J \). For a given allocation \( A = (M, R) \in A(r) \) and a subset of servers \( S \subseteq [K] \), we define \( X_S = \{ X_k : k \in S \} \) and \( Y^G_S = (V^G_{R_{S},:}, V^G_{S,,:}) \), where “:” denotes all possible indices (which depend on both allocation and graph realization). As described in Section II-B, each coded message is a function of the present intermediate values Mapped at the corresponding server. Moreover, all the intermediate values required by the Reducers are decodable from the locally available intermediate values and received messages at the corresponding server. That is, \( H(X_k|V^G_{R_{S,k}}, V^G_{S,,:}) = 0 \) and \( H(V^G_{R_{S,k}}, X_{k}, V^G_{S,,:}) = 0 \) for all servers \( k \in [K] \) and graphs \( G \). We denote the number of vertices that are exclusively Mapped by \( j \) servers in \( S \) as \( a_{j,M}^i \), that is

\[
a_{j,M}^i = \sum_{S \subseteq [K] : |S| = j, S \cap S_i = \emptyset} |\{ k \in S, M_k \} \setminus \{ \cup_{k' \notin S_i, M_{k'}} \}].
\] (41)

We prove the following claim by induction.

Claim 1. For any subset \( S \subseteq [K], \)

\[
\mathbb{E}_G \left[ H(X_S|Y^G_S) \right] \geq pT \sum_{j=1}^{K} a_{j,M}^i \frac{|S| - j}{K}.
\] (42)

Proof. (i) If \( S = \{ k \} \), for any \( k \in [K] \) and graph \( G \) we have \( H(X_S|Y^G_S) \geq 0 \). Therefore,

\[
\mathbb{E}_G \left[ H(X_S|Y^G_S) \right] \geq 0 = pT \sum_{j=1}^{K} a_{j,M}^i \frac{n - 1}{K}.
\] (43)

(ii) Assume that the claim (42) holds for all subsets of size \( S_0 \). For any subset \( S \subseteq [K] \) of size \( S_0 + 1 \), we can write

\[
H(X_S|Y^G_S) \geq \frac{1}{S_0} \sum_{k \in S} H(X_S|V^G_{R_{S,k}}, V^G_{S,,:}).
\] (45)

We can lower bound expected value of the first RHS term in (45) as follows

\[
\mathbb{E}_G \left[ H(V^G_{R_{S,k}}, V^G_{S,,:}) \right] = \mathbb{E}_G \left[ \sum_{v \in R_k} H(V^G_{R_{v},:}|V^G_{S,,:}) \right]
\] (46)

\[
= \mathbb{E}_G \left[ \sum_{v \in R_k} |\mathcal{N}(v)| - |\mathcal{N}(v) \cap (M_k \cup M_{S_i})| \right]
\] (47)

\[
\geq \frac{n}{K} pT \sum_{j=0}^{S_0} a_{j,M}^i |\mathcal{N}(k)|
\] (48)

\[
\geq \frac{n}{K} pT \sum_{j=1}^{S_0} a_{j,M}^i |\mathcal{N}(k)|.
\] (49)

Expected value of the second term in RHS of (45) can be lower bounded from the induction assumption:

\[
\mathbb{E}_G \left[ H(X_S|V^G_{R_{S,k}}, V^G_{S,,:}) \right] \geq \mathbb{E}_G \left[ H(X_{S\setminus\{k\}}|Y^G_{S\setminus\{k\}}) \right]
\] (50)

\[
\geq pT \sum_{j=1}^{S_0} a_{j,M}^i |\mathcal{N}(k)| \frac{n - j}{K}.
\] (51)

Putting (44), (45), (49), and (51) together, we have

\[
\mathbb{E}_G \left[ H(X_S|Y^G_S) \right] \geq \frac{1}{S_0} \sum_{k \in S} \mathbb{E}_G \left[ H(X_S|V^G_{R_{S,k}}, V^G_{S,,:}) \right]
\] (52)

\[
= \frac{1}{S_0} \sum_{k \in S} \left( \frac{n}{K} pT \sum_{i=1}^{S_0} a_{j,M}^i |\mathcal{N}(k)| + pT \sum_{j=1}^{S_0} a_{j,M}^i |\mathcal{N}(k)| \frac{n - j}{K} \right)
\] (53)

\[
\geq \frac{1}{S_0} \sum_{k \in S} \left( \frac{n}{K} pT \sum_{i=1}^{S_0} a_{j,M}^i |\mathcal{N}(k)| + pT \sum_{j=1}^{S_0} a_{j,M}^i |\mathcal{N}(k)| \frac{n - j}{K} \right)
\] (54)
and workers are of type m4.large machine instances. The master is responsible for worker synchronization and time measurements. In all of our experiments, the master is of type r4.large.

Experiments employ a master-worker setup. The workers carry out Uncoded and Coded PageRank implementations, while the subgraphs are allocated to the workers before computation. For Uncoded PageRank, the machine partitions the graph sample with probability $p$ in the MapReduce framework. We use Erdős-Rényi model for random graphs, where each edge in the graph is present with probability $p$.

We first present our implementation choices and experimental scenarios. Then, we discuss the results and provide some remarks. Implementation codes are available at [47].

In this section, we demonstrate the practical impact of our proposed coded schemes via experiments over Amazon EC2 clusters. We use Python with mpi4py package. Using Amazon EC2 machines, the cluster consists of 10 workers and one master with communication bandwidth of 100 Mbps at each machine.

We implement one iteration of the popular PageRank algorithm, which as described in Example 1, can be easily expressed in the MapReduce framework. We use Erdős-Rényi model for random graphs, where each edge in the graph is present with probability $p$. For Simplicity, we do not consider self-loops. For comparison, we implement the following distributed implementations for PageRank:

- **Uncoded PageRank**: Each vertex $i \in V = [n]$ is stored at exactly one machine, where the Map computation for $i$ takes place. Furthermore, we let $\mathcal{M}_k = \mathcal{R}_k$ for each machine $k \in [K]$, i.e. the Map and Reduce tasks associated with any vertex $i$ take place in the same machine.

- **Coded PageRank**: The subgraph and Map allocation and Reduce computation allocation are done in accordance with Section IV-A. Coded Shuffling is used to exchange intermediate values between workers.

We now describe our implementation choices. We use Python with mpi4py package. Using Amazon EC2 machines, the experiments employ a master-worker setup. The workers carry out Uncoded and Coded PageRank implementations, while the master is responsible for worker synchronization and time measurements. In all of our experiments, master is of type r4.large and workers are of type m4.large machine instances.

We consider the following two scenarios:

1) **Scenario 1**: The graph has $n = 12600$ vertices with edge probability $p = 0.3$. The cluster consists of 10 workers and one master with communication bandwidth of 100 Mbps at each machine.

2) **Scenario 2**: The graph has $n = 87360$ vertices with edge probability $p = 0.06$. We use a cluster of 16 workers and one master with communication bandwidth of 100 Mbps at each machine.

For each scenario under consideration, we use a sample from the Erdős-Rényi model. This process is carried out using a c4.8xlarge machine instance. The graphs are processed and subgraph allocation is done as a pre-processing step. The generated subgraphs are allocated to the workers before computation. For Uncoded PageRank, the machine partitions the graph sample into $K$ subgraphs and assigns each subgraph to a machine.

For the Coded PageRank, we use the following symbols:

- $a^j_M$: The size of the subgraph allocated to machine $j$.
- $S$: The total number of subgraphs.
- $\mathcal{M}$: The set of machines.
- $\mathcal{R}$: The set of machines.
- $n$: The total number of vertices in the graph.
- $p$: The edge probability.
- $K$: The total number of machines.
- $A$: The subgraph allocation matrix.
- $G$: The graph.
- $Y$: The adjacency matrix of the graph.
- $X$: The vertex degrees of the graph.
- $T$: The time to compute the PageRank.
- $L$: The communication load.
- $E$: The expected value.
- $P$: The probability.

We define the following equations:

$$L^*(r) \geq \inf_A E_G[|L_A(r, G)|] \geq \inf_A E_G[H(X_S|Y_{S^c})] \geq \inf_A E_G[H(X_S|Y_{S^c})] \geq p \sum_j a^j_M K - j.$$ (55)

Proof of Converse for Theorem 1

First, we can bound the best average communication load as

$$L^*(r) \geq \inf_A E_G[L_A(r, G)] \geq \inf_A p \sum_j a^j_M K - j,$$ (56)

Now, pick $S = [K]$. Then,

$$E_G[L_A(r, G)] \geq \frac{E_G[H(X_S|Y_{S^c})]}{n^2 T} \geq \frac{p \sum_j a^j_M K - j}{n^2 T}. $$ (57)

Proof of Converse for Theorem 1

First, we can bound the best average communication load as

$$L^*(r) \geq \inf_A E_G[L_A(r, G)] \geq \inf_A p \sum_j a^j_M K - j,$$ (58)

where the infimum is over all subgraph and computation allocations $A = (\mathcal{M}, \mathcal{R}) \in A(r)$ for which $\sum_{k=1}^{K} |\mathcal{M}_k| = rn$ and $|\mathcal{R}_k| = \frac{rn}{K}$, $\forall k \in [K]$. Additionally, for any Map allocation with computation load $r$, we have the following equations:

$$\sum_{j=1}^{K} a^j_M = n, \quad \sum_{j=1}^{K} j a^j_M = rn.$$ (59)

Using convexity of $\frac{K - j}{Kj}$ in $j$ and (60), the converse is proved:

$$L^*(r) \geq \frac{1}{r} p \left( 1 - \frac{r}{K} \right). $$ (61)

VI. Experiments over Amazon EC2 Clusters

In this section, we demonstrate the practical impact of our proposed coded schemes via experiments over Amazon EC2 clusters. We first present our implementation choices and experimental scenarios. Then, we discuss the results and provide some remarks. Implementation codes are available at [47].

A. Implementation Details

We implement one iteration of the popular PageRank algorithm, which as described in Example 1, can be easily expressed in the MapReduce framework. We use Erdős-Rényi model for random graphs, where each edge in the graph is present with probability $p$. For simplicity, we do not consider self-loops. For comparison, we implement the following distributed implementations for PageRank:

- **Uncoded PageRank**: Each vertex $i \in V = [n]$ is stored at exactly one machine, where the Map computation for $i$ takes place. Furthermore, we let $\mathcal{M}_k = \mathcal{R}_k$ for each machine $k \in [K]$, i.e. the Map and Reduce tasks associated with any vertex $i$ take place in the same machine.

- **Coded PageRank**: The subgraph and Map allocation and Reduce computation allocation are done in accordance with Section IV-A. Coded Shuffling is used to exchange intermediate values between workers.

We now describe our implementation choices. We use Python with mpi4py package. Using Amazon EC2 machines, the experiments employ a master-worker setup. The workers carry out Uncoded and Coded PageRank implementations, while the master is responsible for worker synchronization and time measurements. In all of our experiments, master is of type r4.large and workers are of type m4.large machine instances.

We consider the following two scenarios:

1) **Scenario 1**: The graph has $n = 12600$ vertices with edge probability $p = 0.3$. The cluster consists of 10 workers and one master with communication bandwidth of 100 Mbps at each machine.

2) **Scenario 2**: The graph has $n = 87360$ vertices with edge probability $p = 0.06$. We use a cluster of 16 workers and one master with communication bandwidth of 100 Mbps at each machine.

For each scenario under consideration, we use a sample from the Erdős-Rényi model. This process is carried out using a c4.8xlarge machine instance. The graphs are processed and subgraph allocation is done as a pre-processing step. The generated subgraphs are allocated to the workers before computation. For Uncoded PageRank, the machine partitions the graph sample
into smaller instances which have equal numbers of vertices. Each such partition consists of two Python list, one that consists of the vertices that will be Mapped by the corresponding machine, and the other one that consists of the neighborhood information of each vertex to be Mapped. The position of the neighborhood tuple in the neighborhood list is same as the position of the corresponding vertex in the vertex list, so that one can iterate over the two together during the Map stage. For Coded PageRank, the graph sample is divided into \( \binom{n}{r} \) subfiles, where each subfile consists of equal numbers of vertices (and their neighborhood data). Then each subfile is included in the subgraph of the corresponding set of \( r \) machines. This way, we get a computation load of \( r \).

The overall execution consists of the following phases:

1) **Map**: Without loss of generality, the rank for each vertex is initialized to \( \frac{1}{n} \). Each worker goes over its subgraph and Maps the rank associated with a vertex to intermediate values that are required by the neighboring vertices during the Reduce stage. Each intermediate value consists of key-value pair, where the key is an integer storing the vertex id, while the value is a real number storing the associated value. Based on the vertex id, the intermediate value is associated with the partition where the vertex is Reduced, which is obtained by hashing the vertex id. For each partition, a separate list is created for storing keys and values.

2) **Encode/Pack**: In Uncoded PageRank, no encoding is done as the transfer of intermediate values is done directly. For Coded PageRank, coded multicast packets are created using the proposed encoding scheme. Transmission data is serialized before Shuffling.

3) **Shuffle**: At any time, only one worker is allowed to use the network for transmission. In Uncoded PageRank, each worker unicasts its message to different servers, while in Coded PageRank, the communication takes place in multicast groups. For any multicast group, each worker takes its turn to broadcast its message to all the remaining workers in the group.

4) **Unpack/Decode**: The messages received during the Shuffle phase are de-serialized. In Coded PageRank, each worker decodes the coded packets received from other workers in accordance with the proposed coded scheme to recover the intermediate values. After the decoding phase, all intermediate values that are needed for Reduce phase are available at the workers.

5) **Reduce**: Each worker goes over its set of vertices that it needs to Reduce and updates the corresponding PageRank values. In Uncoded PageRank, for any vertex \( i \in V \), the Map and Reduce operations associated with it are done at the same worker. Therefore, no further data transmission is needed to communicate the updated ranks for the Map phase in next iteration. In the Coded scheme, message passing is done in order to transmit the updated PageRanks to the Mappers.

Next, we discuss the results of our experiments.

**B. Experimental Results**

For each scenario, we consider two cases of computation load for Coded PageRank, \( r = 3 \) and \( r = 4 \). The execution times for different phases are presented in Figures 7 and 8.

![Fig. 7: Comparison of Coded PageRank and Uncoded PageRank for Scenario 1.](a) Coded PageRank with \( r = 3 \)  
(b) Coded PageRank with \( r = 4 \)

We make the following conclusions from the results:

- Our proposed coded computing scheme achieves a speedup of 32\% to 47\% in comparison to the baseline un-coded implementation of PageRank.
- Shuffle phase dominates the overall execution time. By increasing the computation load, Coded PageRank leverages extra computing in the Map phase to slash the Shuffle phase, thus speeding up the overall execution time.
- Theoretically, we demonstrated that by increasing the computation load by \( r \), we slash the expected communication load in Shuffle phase by \( r \). Here, we empirically observe that due to large size of the graph model, we have a similar trade-off between computation load and communication load for each sample of the graph model as well.

\(^{1}\)The Map time includes the time spent in Encode/Pack stage, while the Unpack stage is combined with Reduce phase.
We observe that while the Map phase increases almost linearly with \( r \), the overall gain begins to saturate, since the Shuffle phase does not decrease linearly with \( r \). This is because as we increase \( r \), the overheads in multicast data transmissions increase and start to dominate the overall Shuffling time. Furthermore, unicasting one packet is smaller than the time for broadcasting the same packet to multiple machines [11].

VII. CONCLUSION

We described a mathematical model for graph based MapReduce computations and demonstrated how coding theoretic strategies can be employed to substantially reduce the communication load in distributed graph analytics. Our results reveal an inverse linear trade-off exists between computation load and communication load in distributed graph analytics. This trade-off can be used to leverage additional computing resources and capabilities to alleviate the costly communication bottleneck in distributed graph processing systems. In particular, we developed a coded scheme that asymptotically characterizes the minimum average normalized communication load for random Erdős-Rényi model. Moreover, we numerically demonstrated that even for finite graphs, not only the proposed scheme significantly reduces the communication load, but also has a small optimality gap. Furthermore, we carried out experiments over Amazon EC2 clusters to corroborate our claims, demonstrating speedups of up to 47% in the overall execution time of PageRank. Furthermore, we extended the achievability of the minimum average normalized communication load for three more random graph models – random bi-partite model, stochastic block model and power law model, thus revealing an interesting inverse linear trade-off between computation and communication in large scale distributed graph analytics.

REFERENCES

[1] S. Prakash, A. Reisizadeh, R. Pedarsani, and S. Avestimehr, “Coded computing for distributed graph analytics,” ISIT, 2018.
[2] 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.
[3] J. E. Gonzalez, Y. Low, H. Gu, D. Bickson, and C. Guestrin, “Powergraph: distributed graph-parallel computation on natural graphs,” in OSDI, 2012.
[4] 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.
[5] 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.
[6] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly, “Dryad: distributed data-parallel programs from sequential building blocks,” EuroSys, 2007.
[7] J. Dean and S. Ghemawat, “Mapreduce: simplified data processing on large clusters,” Communications of the ACM, 2008.
[8] A. Lumsdaine, D. Gregor, B. Hendrickson, and J. Berry, “Challenges in parallel graph processing,” Parallel Processing Letters, 2007.
[9] 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.
[10] 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.
[11] K. Lee, M. Lam, R. Pedarsani, D. Papailiopoulos, and K. Ramchandran, “Speeding up distributed machine learning using codes,” IEEE Transactions on Information Theory, 2017.
[12] S. Li, Q. Yu, M. A. Maddah-Ali, and A. S. Avestimehr, “A scalable framework for wireless distributed computing,” IEEE/ACM Transactions on Networking, vol. 25, no. 5, pp. 2643–2654, 2017.
[13] S. Li, M. A. Maddah-Ali, and A. S. Avestimehr, “Coded distributed computing: Straggling servers and multistage dataflows,” in Communication, Control, and Computing (Allerton), 2016 54th Annual Allerton Conference on, pp. 164–171, IEEE, 2016.
[14] Y. H. Ezzeldin, M. Karmoose, and C. Fragouli, “Communication vs distributed computation: an alternative trade-off curve,” arXiv preprint arXiv:1705.08966, 2017.
[15] K. Konstantinidis and A. Ramamoorthy, “Leveraging coding techniques for speeding up distributed computing,” arXiv preprint arXiv:1802.03049, 2018.
[16] M. Kiamari, C. Wang, and A. S. Avestimehr, “On heterogeneous coded distributed computing,” arXiv preprint arXiv:1709.00196, 2017.
[17] S. Li, M. A. Maddah-Ali, and A. S. Avestimehr, “Compressed coded distributed computing,” arXiv preprint arXiv:1805.01993, 2018.
[18] M. A. Attia and R. Tandon, “Information theoretic limits of data shuffling for distributed learning,” Globecom, 2016.
[19] M. A. Attia and R. Tandon, “Near optimal coded data shuffling for distributed learning,” arXiv preprint arXiv:1801.01875, 2018.
[20] J. Chung, K. Lee, R. Pedarsani, D. Papailiopoulos, and K. Ramchandran, “Ubershuffle: Communication-efficient data shuffling for sgd via coding theory,” NIPS Workshop on ML Systems, 2017.
Consider RB\((n_1, n_2, q)\) graph \(G = (V_1 \cup V_2, E)\) with \(n = n_1 + n_2, |V_1| = n_1 = \Theta(n), \) and \(|V_2| = n_2 = \Theta(n)\). Subgraph and computation allocations are described as follows. Mappers in \(V_1\) and Reducers in \(V_2\) are distributedly allocated to \(K\) servers according to the allocation scheme proposed in Section IV-A. Similarly, Mappers in \(V_2\) and Reducers in \(V_1\) are distributedly allocated to \(K\) servers according to the allocation scheme proposed in Section IV-A. Therefore, each server Maps \(n_1 r / K\) vertices in \(V_1\) and \(n_2 r / K\) vertices in \(V_2\), inducing the computation load \(r\). Moreover, each server Reduces \(n_1 / K\) functions in \(V_1\) and \(n_2 / K\) functions in \(V_2\). We denote this allocation by \(\tilde{A} \in \mathcal{A}(r)\).

Given the allocation \(\tilde{A}\) defined above, both uncoded and coded Shuffling schemes proceed in two steps. In uncoded Shuffling, each server requires \(q n_2\) intermediate values on average to Reduce each function in \(V_1\) while \(r / K\) of them are locally available. Therefore, expected uncoded load for Reducing functions in \(V_1\) is \(\tilde{F}^{\text{UC1}}_A = q n_1 n_2 / (n_1 + n_2)^2 \left(1 - \frac{r}{K}\right)\). Similarly, expected uncoded load for Reducing \(V_2\) is \(\tilde{F}^{\text{UC2}}_A = q n_1 n_2 / (n_1 + n_2)^2 \left(1 - \frac{r}{K}\right)\).

However, coded Shuffling applies the coded scheme proposed in Section IV-A for Reducing functions in \(V_1\) and \(V_2\) separately. Let \(\tilde{L}^{\text{C1}}_A\) and \(\tilde{L}^{\text{C2}}_A\) denote the average normalized communication load induced by coded Shuffling according to allocation \(\tilde{A}\). From the achievability result in Theorem 1 for \(q = \omega_1\left(\frac{r}{n_2}\right)\), we have

\[
\tilde{L}^{\text{C1}}_A \leq \frac{1}{r} \tilde{F}^{\text{UC1}}_A + o(q) = \frac{1}{r} q \frac{n_1 n_2}{(n_1 + n_2)^2} \left(1 - \frac{r}{K}\right) + o(q),
\]

and

\[
\tilde{L}^{\text{C2}}_A \leq \frac{1}{r} \tilde{F}^{\text{UC2}}_A + o(q) = \frac{1}{r} q \frac{n_1 n_2}{(n_1 + n_2)^2} \left(1 - \frac{r}{K}\right) + o(q).
\]
Finally, we employ the achievable scheme described in Appendix A to RB graph $G$, and uncoded communication loads averaged over the random connections and random degrees induced by the subgraph and connected with probability $p$, given the realization of the expected degrees drawn from a power law distribution with exponent $\gamma$. Which concludes Theorem 3.

Let us denote by $V_in$ and $ER_A = \tilde{L}_A^U = \frac{1}{p} \frac{1}{\frac{1}{2} (n_1 + n_2)^2} \left(1 - \frac{r}{K}\right)$ the total average normalized communication loads of the coded and uncoded schemes, respectively.

In the coded scheme, we propose to employ achievable schemes used for ER and RB models in the regime of interest, that is $p = \omega(\frac{1}{n^2}), q = \omega(\frac{1}{n^2})$ and $p \geq q$. More specifically, we first apply the coded scheme described in Section IV-A to ER graph $G$ which induces the average coded load

$$L^c_A \leq \frac{1}{r} \frac{1}{\tilde{L}^U} + o(p) = \frac{1}{r} \frac{1}{\frac{1}{2} (n_1 + n_2)^2} \left(1 - \frac{r}{K}\right) + o(p).$$

Similarly, the same scheme applied to ER graph $G$ results in the average coded load

$$L^c_A \leq \frac{1}{r} \frac{1}{\tilde{L}^U} + o(p) = \frac{1}{r} \frac{1}{\frac{1}{2} (n_1 + n_2)^2} \left(1 - \frac{r}{K}\right) + o(p).$$

Finally, we employ the achievable scheme described in Appendix A to RB graph $G$ which induces the average coded load

$$L^c_A \leq \frac{1}{r} \frac{1}{\tilde{L}^U} + o(q) = \frac{1}{r} \frac{2n_1 n_2}{(n_1 + n_2)^2} \left(1 - \frac{r}{K}\right) + o(q).$$

Let us denote by $L^c_A$ and $L^U_A$ the total average normalized communication loads of the coded and uncoded schemes, respectively. Therefore,

$$L^c_A \leq \frac{1}{r} \frac{1}{\tilde{L}^U} + o(p) = \frac{1}{r} \frac{1}{\frac{1}{2} (n_1 + n_2)^2} \left(1 - \frac{r}{K}\right) + o(p).$$

which concludes Theorem 3.

### Appendix B

**Achievability for Stochastic Block Model**

Consider an SBM($n_1, n_2, p, q$) graph $G = (V_1 \cup V_2, E_1 \cup E_2 \cup E_3)$ with $n = n_1 + n_2$, $|V_1| = n_1 = \Theta(n)$, and $|V_2| = n_2 = \Theta(n)$. Edge subsets $E_1$, $E_2$ and $E_3$ respectively represent intra-cluster edges among vertices in $V_1$, intra-cluster edges among vertices in $V_2$, and inter-cluster edges between vertices in $V_1$ and $V_2$. Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be graphs induced by $V_1$ and $V_2$, respectively, and denote the graph of inter-cluster connections by $G_3 = (V_1 \cup V_2, E_3)$. Clearly, $G_1$ and $G_2$ are ER($n_1, p$) and ER($n_2, p$) graphs, while $G_3$ is RB($n_1, n_2, q$) graph.

We consider the allocation $A$, proposed in Appendix A for both uncoded and coded Shufflings. In uncoded schemes, Reducing each function in $V_1$ requires on average $p n_1$ intermediate values Mapped by vertices in $V_1$ due to intra-cluster connections which introduces the average uncoded load $L^U_A = p \frac{n_1^2}{(n_1 + n_2)^2} \left(1 - \frac{r}{K}\right)$. Similarly, the average uncoded load for Reducing $V_2$ due to intra-cluster connections is $L^U_A = p \frac{n_2^2}{(n_1 + n_2)^2} \left(1 - \frac{r}{K}\right)$. Moreover, inter-cluster connections induce an average load $L^U_A = q \frac{2n_1 n_2}{(n_1 + n_2)^2} \left(1 - \frac{r}{K}\right)$.

In the coded scheme, we propose to employ achievable schemes used for ER and RB models in the regime of interest, that is $p = \omega(\frac{1}{n^2}), q = \omega(\frac{1}{n^2})$ and $p \geq q$. More specifically, we first apply the coded scheme described in Section IV-A to ER graph $G$ which induces the average coded load

$$L^c_A \leq \frac{1}{r} \frac{1}{\tilde{L}^U} + o(p) = \frac{1}{r} \frac{1}{\frac{1}{2} (n_1 + n_2)^2} \left(1 - \frac{r}{K}\right) + o(p).$$

Similarly, the same scheme applied to ER graph $G$ results in the average coded load

$$L^c_A \leq \frac{1}{r} \frac{1}{\tilde{L}^U} + o(p) = \frac{1}{r} \frac{1}{\frac{1}{2} (n_1 + n_2)^2} \left(1 - \frac{r}{K}\right) + o(p).$$

Finally, we employ the achievable scheme described in Appendix A to RB graph $G$ which induces the average coded load

$$L^c_A \leq \frac{1}{r} \frac{1}{\tilde{L}^U} + o(q) = \frac{1}{r} \frac{2n_1 n_2}{(n_1 + n_2)^2} \left(1 - \frac{r}{K}\right) + o(q).$$

Let us denote by $L^c_A$ and $L^U_A$ the total average normalized communication loads of the coded and uncoded schemes, respectively. Therefore,

$$L^c_A \leq \frac{L^c_A}{r} + \frac{L^U_A}{r} + o(q)$$

which concludes Theorem 3.

### Appendix C

**Achievability for Power Law Model**

We consider a general model for random graphs where the expected degree sequence $d_i = c d^{1-\alpha}$ is independently drawn from a power law distribution with exponent $\alpha$, i.e. $P[d_i = d] = cd^{1-\alpha}$ for $i \in [n]$ and $d \geq 1$ and proper constant $c$. Given the realization of the expected degrees $d_i, d_j$ for a sufficiently small constant $p$ and all $i, j \in [n]$, vertices $i$ and $j$ are connected with probability $p_{i,j} = P[(i, j) \in E] = pd_i d_j$, independently of other edges. We now proceed to analyze the coded and uncoded communication loads averaged over the random connections and random degrees induced by the subgraph and computation allocation $A_c$ proposed in Section IV-A.
Consider the allocation $A_C = (\mathcal{M}, \mathcal{R})$ and a subset of servers $S \subseteq [K]$ of size $|S| = r+1$. According to the proposed scheme in Section IV-A for every server $s \in S$, servers in $S \setminus \{s\}$ form a table and construct coded messages using the intermediate values in the sets $Z_{S \setminus \{k\}}^k$ (defined in (13)) where $k \in S \setminus \{s\}$. Therefore, $r+1$ tables are formed each constructing coded messages of size $\max_{k \in S \setminus \{s\}} |Z_{S \setminus \{k\}}^k| \leq \frac{r}{k}$ bits. The total coded load induced by the subset $S$ (and exclusively for the use of servers in $S$) denoted by $L_{AC}^c(S)$ is

$$L_{AC}^c(S) = \frac{1}{n^x} \sum_{s \in S} \max_{k \in S \setminus \{s\}} |Z_{S \setminus \{k\}}^k|. \quad (76)$$

However, in uncoded scenarios, denoted by $L_{AC}^{uc}(S)$ the total uncoded load induced by subset $S$ (and exclusively for the use of servers in $S$) is

$$L_{AC}^{uc}(S) = \frac{1}{n^x} \sum_{s \in S} |Z_{S \setminus \{s\}}^s|. \quad (77)$$

We have

$$|Z_{S \setminus \{s\}}^s| = \sum_{i \in \mathcal{R}_s} |\mathcal{N}(i) \cap (\cap_{k' \in S \setminus \{s\}} \mathcal{M}_{k'})| \quad (78)$$

$$= \sum_{m \in (\cap_{k' \in S \setminus \{s\}} \mathcal{M}_{k'})} \mathbb{I}\{(i, m) \in \mathcal{E}\}, \quad (79)$$

where the random Bernoulli $\mathbb{I}\{(i, m) \in \mathcal{E}\}$ indicates the realization of the edge connecting vertices $i$ and $m$, i.e. $\mathbb{E}[\mathbb{I}\{(i, m) \in \mathcal{E}\}|\mathbf{d}] = pd_i d_m$. We note that $|\mathcal{R}_s| = n/K$ and $|\cap_{k' \in S \setminus \{s\}} \mathcal{M}_{k'}| = n/\binom{K}{r}$. Therefore, there are $\tilde{g} = \frac{n^x}{\mathcal{R}(S)}$ Bernoulli summands in (79) in which every two summands are either independent or equal and independent of other summands. More precisely, (79) can be decomposed to sum of all independent Bernoulli random variables and sum of dependent ones as follows:

$$|Z_{S \setminus \{s\}}^s| = \sum_{i \in \mathcal{R}_s} \mathbb{I}\{(i, m) \in \mathcal{E}\} \quad (80)$$

$$= \sum_{i \in \mathcal{R}_s, m \in (\cap_{k' \in S \setminus \{s\}} \mathcal{M}_{k'}) \cap \mathcal{R}_s} \mathbb{I}\{(i, m) \in \mathcal{E}\} + 2 \sum_{i < m, i \in \mathcal{R}_s, m \in (\cap_{k' \in S \setminus \{s\}} \mathcal{M}_{k'}) \cap \mathcal{R}_s} \mathbb{I}\{(i, m) \in \mathcal{E}\}. \quad (81)$$

Note that with this decomposition, all the Bernoulli summands in both terms in (81) are independent. Assume that the first and second terms in (81) contain $\tilde{g} - 2J$ and $J$ summands respectively.

According to Kolmogorov’s strong law of large numbers (Proposition 1) and given that the second condition in the proposition is satisfied for Bernoullis, we have

$$\frac{1}{\tilde{g} - 2J} \sum_{i \in \mathcal{R}_s, m \in (\cap_{k' \in S \setminus \{s\}} \mathcal{M}_{k'}) \cap \mathcal{R}_s} \mathbb{I}\{(i, m) \in \mathcal{E}\} \rightarrow \mathbb{E}[\mathbb{I}\{(i, m) \in \mathcal{E}\}] - \mathbb{E}[pd_i d_m] \quad \text{a.s.} \quad (82)$$

$$\frac{1}{J} \sum_{i < m, i \in \mathcal{R}_s, m \in (\cap_{k' \in S \setminus \{s\}} \mathcal{M}_{k'}) \cap \mathcal{R}_s} \mathbb{I}\{(i, m) \in \mathcal{E}\} \rightarrow \mathbb{E}[\mathbb{I}\{(i, m) \in \mathcal{E}\}] - \mathbb{E}[pd_i d_m] \quad \text{a.s.} \quad (83)$$

Therefore, size of the set $Z_{S \setminus \{s\}}^s$ converges almost surely, that is

$$\frac{1}{\tilde{g}} \left(|Z_{S \setminus \{s\}}^s| - \mathbb{E}[|Z_{S \setminus \{s\}}^s|]\right) = \frac{\tilde{g} - 2J}{\tilde{g} - 2J} \sum_{i \in \mathcal{R}_s, m \in (\cap_{k' \in S \setminus \{s\}} \mathcal{M}_{k'}) \cap \mathcal{R}_s} \mathbb{I}\{(i, m) \in \mathcal{E}\} - \mathbb{E}[pd_i d_m] \quad (84)$$

$$+ \frac{J}{\tilde{g}} \sum_{i < m, i \in \mathcal{R}_s, m \in (\cap_{k' \in S \setminus \{s\}} \mathcal{M}_{k'}) \cap \mathcal{R}_s} \mathbb{I}\{(i, m) \in \mathcal{E}\} - \mathbb{E}[pd_i d_m] \quad (85)$$

$$\rightarrow 0, \quad (86)$$
where
\[
\mathbb{E}[|Z^*_S\setminus\{s\}|] = \sum_{i\in\mathcal{K}_S} \mathbb{E}[d_id_m]
\]  \hspace{1cm} (87)
and \(\text{vol}(V) = \sum_{v \in V} d_v\) for any subset of vertices \(V \subseteq [n]\). Moreover,
\[
\lim_{n \to \infty} \frac{1}{\rho K} \mathbb{E}[|Z^*_S\setminus\{s\}|] = \mathbb{E}[d_1]
\]  \hspace{1cm} (88)

Each of the terms \(\text{vol}(\mathcal{K}_S)\) and \(\text{vol}(\cap_{k \in \mathcal{K}_S \setminus \{s\}} \mathcal{M}_k')\) are summation of i.i.d power law random variables for which the expected value exists for \(\gamma > 2\) and \(\mathbb{E}[d_1] = \frac{\gamma - 1}{\gamma - 2}\). Therefore, by strong law of large numbers (Proposition 1, condition 1) each term approaches its average almost surely, that is for \(\gamma > 2\)
\[
\frac{1}{n/K} \text{vol}(\mathcal{K}_S) \xrightarrow{a.s.} \mathbb{E}[d_1] = \gamma - 1
\]  \hspace{1cm} (90)
\[
\frac{1}{n/(r \gamma)} \text{vol}(\cap_{k \in \mathcal{K}_S \setminus \{s\}} \mathcal{M}_k') \xrightarrow{a.s.} \mathbb{E}[d_1] = \gamma - 1
\]  \hspace{1cm} (91)

Plugging into (88), we have
\[
\lim_{n \to \infty} \frac{1}{\rho K} \mathbb{E}[|Z^*_S\setminus\{s\}|] = \rho \left(\frac{\gamma - 1}{\gamma - 2}\right)^2
\]  \hspace{1cm} (92)

Therefore, denoted by \(L^*_\mathcal{U}_c\) the total uncoded communication load, we have
\[
\lim_{n \to \infty} \mathbb{E}[L^*_\mathcal{U}_c(S)] = \lim_{n \to \infty} \frac{1}{n^2} \sum_{s \in S} \mathbb{E}[|Z^*_S\setminus\{s\}|]
\]  \hspace{1cm} (93)
\[
= \frac{1}{K(r \gamma)} \lim_{n \to \infty} \frac{1}{n} \sum_{s \in S} \mathbb{E}[|Z^*_S\setminus\{s\}|]
\]  \hspace{1cm} (94)

For the coded scheme, we have
\[
\lim_{n \to \infty} \mathbb{E}[L^*_\mathcal{C}_c(S)] = \lim_{n \to \infty} \frac{1}{n^2} \sum_{s \in S} \mathbb{E}[\max_{k \in S \setminus \{s\}} |Z^*_S\setminus\{k\}|]
\]  \hspace{1cm} (95)
\[
\leq \lim_{n \to \infty} \frac{r + 1}{n^2} \mathbb{E}[\max_{s \in S} |Z^*_S\setminus\{s\}|]
\]  \hspace{1cm} (96)
\[
= \frac{r + 1}{r K'(r \gamma)} \rho \left(\frac{\gamma - 1}{\gamma - 2}\right)^2
\]  \hspace{1cm} (97)

The last equality follows the fact that \(\frac{1}{\rho} \max_{s \in S} |Z^*_S\setminus\{s\}| \xrightarrow{a.s.} \rho \left(\frac{\gamma - 1}{\gamma - 2}\right)^2\), since \(\frac{1}{\rho} |Z^*_S\setminus\{s\}| \xrightarrow{a.s.}\) converges almost surely for any \(s \in S\). Plugging into (97), the expected coded load is
\[
\lim_{n \to \infty} \mathbb{E}[L^*_\mathcal{C}_c] = \lim_{n \to \infty} \sum_{S \subseteq [K]} \mathbb{E}[L^*_\mathcal{C}_c(S)] \leq \left(\frac{K}{r + 1} + \frac{r + 1}{r K'(r \gamma)} \rho \left(\frac{\gamma - 1}{\gamma - 2}\right)^2\right)^2 = \frac{1}{r} \left(1 - \frac{r}{K}\right) \rho \left(\frac{\gamma - 1}{\gamma - 2}\right)^2
\]  \hspace{1cm} (98)

Comparing the coded load with uncoded load proves the achievability of gain \(r\) for power law model.

**Proposition 1** (Kolmogorov’s Strong Law of Large Numbers [49], [50]). Let \(X_1, X_2, \cdots, X_n, \cdots\) be a sequence of independent random variables with \(|\mathbb{E}[X_n]| < \infty\) for \(n \geq 1\). Then
\[
\frac{1}{n} \sum_{i=1}^{n} (X_i - \mathbb{E}[X_i]) \xrightarrow{a.s.} 0
\]  \hspace{1cm} (100)

if one of the following conditions are satisfied:
1) \( X_i \)'s are identically distributed,
2) \( \forall n, \text{var}(X_n) < \infty \) and \( \sum_{n=1}^{\infty} \frac{\text{var}(X_n)}{n^2} < \infty \).

**Appendix D**

**Lemma 4.** For all \( p \in [0, 1] \) and \( s' > 0 \), we have \( (pe^{s'} + 1 - p)^2 \leq pe^{2s'} + 1 - p \).

**Proof.** For given \( p \in [0, 1] \), define \( f(s') = (pe^{s'} + 1 - p)^2 - (pe^{2s'} + 1 - p) \). Clearly \( f(0) = 0 \). Moreover,

\[
f'(s') = 2p\bar{p}(e^{s'} - e^{2s'}) < 0,
\]

for \( s' > 0 \). Therefore, \( f(s') \leq 0 \) for all \( s' > 0 \), concluding the claim of the lemma. \( \square \)