TACC: Topology-Aware Coded Computing for Distributed Graph Processing

Başak Güler, *Member, IEEE*, A. Salman Avestimehr, *Fellow, IEEE*, and Antonio Ortega, *Fellow, IEEE*

***Abstract*—This article proposes a coded distributed graph processing framework to alleviate the communication bottleneck in large-scale distributed graph processing. In particular, we propose a topology-aware coded computing (TACC) algorithm that has two novel salient features: (i) a topology-aware graph allocation strategy, and (ii) a coded aggregation scheme that combines the intermediate computations for graph processing while constructing coded messages. The proposed setup results in a trade-off between computation and communication, in that increasing the computation load at the distributed parties can in turn reduce the communication load. We demonstrate the effectiveness of the TACC algorithm by comparing the communication load with existing setups on both Erdös-Rényi and Barabási-Albert type random graphs, as well as real-world Google web graph for PageRank computations. In particular, we show that the proposed coding strategy can lead to up to 82% reduction in communication load and up to 46% reduction in overall execution time, when compared to the state-of-the-art and implemented on the Amazon EC2 cloud compute platform.**

***Index Terms*—Distributed computing, large-scale graph processing, graph signal filtering.**

I. Introduction

GRAPHS are a fundamental building block for modeling large-scale data applications. Graph-based computations model scenarios for which the computations at each vertex can be expressed as a linear combination of the values associated with neighboring vertices. A large-number of real-world applications can be expressed in this form. Examples include PageRank evaluation for web search [2], and graph signal filtering [3]–[6], where the filter output at each node is a linear combination of signals from neighboring nodes scaled by filter coefficients. On the one hand, these applications require processing large amounts of data represented over graphs in a short amount of time. On the other hand, real-world graphs are often too large to fit in a single processor. Even in scenarios for which the graph can be stored at a single processor, the execution time might be too long for delay-sensitive applications. These limitations can be alleviated by leveraging distributed computing, where each processor only stores and processes a portion of the graph.

Distributed graph processing provides a convenient way to scale-up data-intensive graph applications, such as scaling graph signal processing algorithms to large graphs [3]–[6], by distributing the storage and computation load across multiple processors [7], [8]. As the topology of a graph is irregular, graph processing frameworks operate on a *think like a vertex* principle, where each vertex is assumed to send and receive messages along the graph edges [7]. Doing so can improve the performance of graph processing beyond that of conventional frameworks designed for processing regular data, such as MapReduce, which are oblivious to the underlying graph topology [9]. A major challenge in distributed graph processing is inter-processor communication, i.e., the amount of communication that needs to take place between the processors, as most graph algorithms require vertices stored at different processors to communicate with one another. As the computations are distributed across a larger number of machines, inter-server communication increases and becomes a major bottleneck. In fact, inter-processor communication can take up to 50% of the overall execution time, and may even overcome the benefits of parallelization [10], [11].

The conventional approach to address the inter-processor communication bottleneck has been to leverage effective graph partitioning schemes that minimize the total *cut size*, i.e., the number of edges between the nodes stored at different processors [12]. Various notable methods are proposed to obtain a sparse graph cut, such as spectral partitioning [13], geometric partitioning [14], and multi-level graph partitioning [15]. A detailed review of these methods is available in [16].

Recently, coding has proven to be an effective technique in tackling the communication bottleneck in distributed systems. It has been shown in [17] that coding can be utilized to achieve an inverse-linear trade-off between the communication and storage load in a MapReduce system [9]. This result demonstrates that by increasing the storage load of the distributed parties, i.e., the amount of computation performed by each processor, one can reduce by a linear factor the amount of communication that needs to take place between the workers. The coded MapReduce framework from [17] has been extended to graph-based computations in [18]. In fact, one can view the MapReduce framework

Manuscript received August 20, 2019; revised April 7, 2020; accepted May 11, 2020. Date of publication May 28, 2020; date of current version July 1, 2020. This work was presented in part at the 2019 International Conference on Acoustics, Speech, and Signal Processing (ICASSP'19) [1]. This work was supported in part by the NSF under grants CCF-1410009 and CCF-1527874, and in part by the Defense Advanced Research Projects Agency (DARPA) under Contract No. HR001117C0053, ARO award W911NF1810400, NSF grants CCF-1703575, ONR Award No. N00014-16-1-2189, and CCF-1763673. 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. The associate editor coordinating the review of this manuscript and approving it for publication was Dr. Gonzalo Mateos. *(Corresponding author: Başak Güler.)*

The authors are with the Department of Electrical and Computer Engineering, University of Southern California, Los Angeles, CA 90089 USA (e-mail: bguler@usc.edu; avestimehr@usc.edu; antonio.ortega@sipi.usc.edu).

Digital Object Identifier 10.1109/TSIPN.2020.2998223

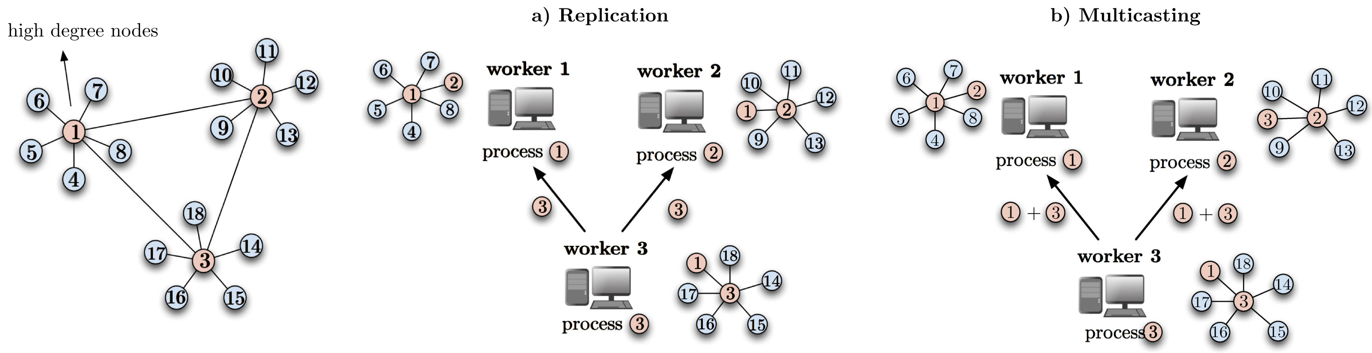


Fig. 1. Communication strategies for an irregular graph topology and 3 processors (workers). Each worker has a memory that can store up to 7 nodes.

from [17] as a special case of [18], where communication takes place between any pair of nodes. For Erdös-Rényi graphs, it has been shown in [18] that one can also achieve an inverse-linear trade-off between computation and communication. Recently, reference [19] has leveraged aggregation techniques based on linearly combining the intermediate results to reduce the communication load in the MapReduce framework [17].

This paper extends the topology-independent coded graph processing setup from [18] to networks with irregular graph topologies. Unlike classical random graph models such as the Erdös-Rényi graph, real-world graphs such as the web often have irregular degree topologies [20]. An important question for minimizing the inter-processor communication in such topologies is that of assigning nodes to processors. This assignment problem is particularly significant for high degree nodes. One can replicate a high degree node at a large number of processors (each containing a subset of its neighbors), and completely eliminate the cost of communication for that node. Alternatively, one can place that node at fewer processors and utilize multicasting to communicate its value with all its neighbors. By a careful placement of such nodes across the processors, this can also enable network coding opportunities, e.g., multicasting coded messages that will be useful to multiple processors simultaneously. This will cost less storage (than full replication) but at a higher communication cost. The irregularity in graph topology causes a trade-off between these two approaches, i.e., between less communication with higher storage (due to replication) and higher communication cost with lower storage (with multicasting). This requires the development of a topology-aware processing strategy.

The trade-off arising from these two alternative approaches is illustrated in Fig. 1. Consider Node 1 (high degree) and the communication load between Worker 3 and the other two workers. In Fig. 1(a), Node 1 is replicated at all 3 workers, thus eliminating the cost of communication completely for this node. Worker 3 then broadcasts the remaining node, Node 3, to the remaining workers. In Fig. 1(b), Node 1 is stored at only 2 workers, thus reducing the storage load for that node. On the other hand, the communication cost is now increased, as Node 1 needs to be sent to Worker 2. To achieve efficient communication, Worker 3 utilizes multi-casting opportunities, by sending a coded message, the sum of nodes 1 and 3. Worker 2 can then remove Node 3 from the coded message and learn Node 1, whereas Worker 1 can remove Node 1 to learn 3.

To address this trade-off we propose a topology-aware coded computing framework (TACC) that distributes the nodes across multiple processors to create multi-casting opportunities, but while doing so also ensures that high degree nodes are simultaneously stored at many processors. To do so, we first propose a topology-aware graph allocation strategy to create redundancy in the computations performed by each processor, where different fractions of a processor's memory are allocated to vertices with different degree structures. Second, we develop a coded aggregation technique that combines the intermediate computations prior to communication, and creates coded messages using the aggregated computations.

We show that TACC can greatly reduce the communication load as compared to existing coded and uncoded computing frameworks that are oblivious to the graph structures. We analyze theoretically the asymptotic performance of our algorithm in terms of the expected communication load on generalized random graphs, which extend the classical Erdös-Rényi random graph model to graphs with irregular degree topologies [21]–[23]. We develop upper and lower bounds on the expected communication load for TACC as well as the topology-independent coding setups, demonstrating the benefits of TACC compared to the topology-independent setup.

Next, we demonstrate the effectiveness of TACC on real-world graphs. Similar to random graph topologies, we show that topology-aware coding can also significantly outperform the topology-independent setup in real-world graphs. We first test it on a Google web graph, a real-world dataset that represents the connections between webpages [24]. For the Google web graph, we compare the communication load of TACC to the state-of-the-art distributed coding algorithm which is oblivious to the graph topology [18], demonstrate that topology aware coding with aggregation can lead up to 82% improvement in reducing the communication load when compared to existing distributed computing setups. Next, we implement the PageRank algorithm on the Amazon AWS EC2 distributed cloud computing framework, and show that for an Erdös-Rényi random graph, TACC can reduce the overall execution time by up to 46% over the state-of-the-art.

Our main contributions over the topology-independent computing framework of [18] can be summarized as follows:

- We introduce *topology-aware coded computing* (TACC) for irregular graph topologies. TACC builds a topology-aware graph allocation strategy, where overlapping parts of a graph is allocated to multiple workers in a way that enables multi-casting opportunities, while ensuring higher degree nodes are replicated at a larger fraction of workers.
- We propose *coded aggregation* for speeding up graph processing, in which coding is performed over linear combinations of the intermediate values, instead of coding directly over the intermediate values as in [18]. We show that coded aggregation can reduce the communication load by up to 95% on an Erdös-Rényi graph.
- We theoretically establish upper and lower bounds on the performance of TACC for random graph models with irregular topologies.
- We demonstrate that TACC can reduce the communication cost by more than 80% over the state-of-the-art, on both random graphs and real-world scale-free graphs.

This paper extends our preliminary work [1] by providing an extensive theoretical analysis of the algorithm as well as additional experimental evaluations.

Related Work: In addition to the graph processing specific applications, such as [7], [12], [18], this work is also related to distributed matrix multiplication and distributed learning. For these setups, coding has been utilized primarily to handle failed or straggling (delayed) workers [25]–[36], where some workers fail or are significantly slower than the other workers, causing a significant delay in the overall computation time. For these problems, a coded computation strategy can allow the master to wait only for the results of a subset of fast workers, instead of all the workers in the system. Notable techniques include encoding the input matrices to handle stragglers in distributed matrix multiplication [25]–[28], as well as encoding the dataset [29]–[31] or gradients [32] for iterative algorithms in distributed learning. In a related line of work, a coded computation technique has been developed for performing iterative algorithms (such as PageRank) under a deadline constraint [33]. More recently, coded computing schemes have been developed for sparse matrices using rateless codes [34], [35], to better utilize the work done by slow workers. Reference [36] has introduced novel coding schemes based on block-diagonal and LT codes for handling stragglers while achieving significantly lower computational delay. Finally, coded computing is heavily inspired by earlier research on coded caching for content delivery [37]–[40], where the goal is to store parts of the data at the local memory of users to handle network congestion during busy hours.

The focus of this paper is not on the straggler or node failure problem, but instead in reducing the communication load of the distributed system. Moreover, unlike the general matrix-vector multiplication problem, graphs often exhibit special topological behavior, for instance, matrices corresponding to graphs (adjacency matrices) are often sparse while node degrees can vary significantly.

Notation: In the remainder of the paper, x is a scalar variable, $\mathbf{x} = [x_1, \ldots, x_n]^T$ is a vector of size N, and $\mathcal{X}$ represents a set with cardinality $|\mathcal{X}|$. $\mathbf{A}$ is a matrix with A_{ij} denoting its $(i,j)^{th}$ element. Finally, $[K]$ represents the set $\{1, \ldots, K\}$.

II. Problem Formulation

We consider a graph $G = (\mathcal{V}, \mathcal{E})$ with $|\mathcal{V}| = N$ nodes (vertices). If G is an undirected graph, $(i,j) = (j,i)$ represents an undirected edge between nodes i and j, whereas if G is directed, (i,j) indicates a directed edge from node i to node j. Vector $\mathbf{x} = (x_1, \ldots, x_N)$ represents vertex-labels, such that $x_i \in \mathbb{R}$ is the label of node i. $\mathbf{x}$ can also be interpreted as a graph signal [4]. An edge weight $e_{ji} \in \mathbb{R}$ is associated with edge $(i,j) \in \mathcal{E}$. We define the (open) neighborhood of node $i \in \mathcal{V}$ by $\mathcal{N}(i) = \{j : (i,j) \in \mathcal{E}\}$. We are interested in computing functions of the form

$$y_j = \sum_{i:j \in \mathcal{N}(i)} e_{ji} x_i \tag{1}$$

for $j \in [N]$. The output value y_j is a linear combination of the signals associated with the neighbors of node j. By defining the (weighted) adjacency matrix $\mathbf{A}$ such that,

$$A_{ji} = \begin{cases} e_{ji} & \text{if} \quad j \in \mathcal{N}(i) \\ 0 & \text{otherwise} \end{cases}, \tag{2}$$

one can represent (1) as,

$$\mathbf{y} = \mathbf{A}\mathbf{x}, \tag{3}$$

where $\mathbf{y} = (y_1, \ldots, y_N)$. In applications such as graph filtering, we replace $\mathcal{N}(i)$ in (1) with the closed neighborhood $\{i\} \cup \mathcal{N}(i)$. Equation (1) can also be represented as a MapReduce computation [9], by decomposing it into *map* and *reduce* functions. The map function represents the inner computation

$$g_{ji}(x_i) \triangleq e_{ji} x_i, \tag{4}$$

where $u_{ji} \triangleq g_{ji}(x_i)$ is called an *intermediate value*. The reduce function represents the summation

$$f_j(\{u_{ji} : j \in \mathcal{N}(i)\}) = \sum_{i:j \in \mathcal{N}(i)} u_{ji}, \tag{5}$$

to compute the output y_j in (1). A large number of graph-based algorithms can be represented in the form of (1), including PageRank computation, graph signal filtering, or semi-supervised learning. We next present two specific examples.

Example 1 (Distributed PageRank Computation): PageRank is a widely used graph algorithm for ranking webpages in web search engines [2]. Specifically, this algorithm operates on a directed graph in which nodes represent webpages, and an edge represents a hyperlink from one webpage to another. The PageRank algorithm can be viewed as a process in which computations in the form of (1) are performed iteratively. To do so, one can define the matrix $\mathbf{A}$ such that,

$$A_{ji} = \begin{cases} \frac{1}{|\mathcal{N}(i)|} & \text{if} \quad (i,j) \in \mathcal{E} \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

and compute

$$\mathbf{x}^{(t)} = \mathbf{A}\mathbf{x}^{(t-1)} \tag{7}$$

iteratively until a convergence criterion is satisfied, which is often stated as

$$\sum_{i\in\mathcal{V}} |x_i^{(t)} - x_i^{(t-1)}| \leq \varepsilon \tag{8}$$

for some small $\varepsilon > 0$, where $\mathbf{x}^{(t)} = (x_1^{(t)}, \ldots, x_N^{(t)})$ denotes the PageRank values at iteration t. The initial PageRank values are often set as $\mathbf{x}^{(0)} = (1/N, \ldots, 1/N)$.

Example 2 (Distributed Graph Signal Filtering): Graph filters are building blocks of various graph signal processing applications, such as graph neural networks [6], [41], and can be defined as follows. Let $\mathbf{A}$ denote the adjacency matrix from (2) for graph $G = (\mathcal{V}, \mathcal{E})$. Define a diagonal matrix $\mathbf{D} \triangleq \text{diag}(\deg(1), \ldots, \deg(N))$ whose i^{th} diagonal holds the degree of node $i \in \mathcal{V}$. Then, $\mathbf{L} \triangleq \mathbf{D} - \mathbf{A}$ is called the *graph Laplacian* of G. A graph filter of order T is defined as,

$$\mathbf{y} = \sum_{t=1}^{T} \alpha_t \mathbf{L}^t \mathbf{x}, \tag{9}$$

where $\alpha_t, \ldots, \alpha_T$ denote the filter coefficients and $\mathbf{L}^t$ corresponds to the tth power of $\mathbf{L}$. Output (9) can be computed in T iterations, by performing computations of the form (3) at each step, and treating the output of each step as the input of the next step. Specifically, by letting $\mathbf{x}^{(t)}$ denote the input at iteration t, one can compute

$$\mathbf{x}^{(t)} = \mathbf{L}\mathbf{x}^{(t-1)} \tag{10}$$

with $\mathbf{x}^{(0)} = \mathbf{x}$ and evaluate (9) from $\mathbf{y} = \sum_{t=1}^{T} \alpha_t \mathbf{x}^{(t)}$.

We consider a distributed scenario in which (1) is computed by K workers (processors), connected through a multicast network. Worker k is responsible from computing the outputs in (1) for $\frac{N}{K}$ nodes, given by a set $\mathcal{R}_k$ such that $\bigcup_{k=1}^{K} \mathcal{R}_k = \mathcal{V}$ and $\mathcal{R}_k \cap \mathcal{R}_j = \emptyset$ for $k \neq j$. Each worker stores a subgraph of G denoted by $\mathcal{M}_k$, where $\mathcal{R}_k \subseteq \mathcal{M}_k \subseteq \mathcal{V}$ and $|\mathcal{M}_k| = |\mathcal{M}_j|$ for all $k, j \in [K]$.[1] We then define the storage load of the system as follows.

Definition 1 (Storage Load): Given a subgraph allocation $\mathcal{M} = (\mathcal{M}_1, \ldots, \mathcal{M}_K)$, the storage load of the distributed graph processing system is characterized as,

$$r \triangleq \frac{\sum_{k=1}^{K} |\mathcal{M}_k|}{N}. \tag{11}$$

Accordingly, worker k stores a subgraph with $r\frac{N}{K}$ nodes.

Remark 1: The storage load r indicates what fraction of the graph nodes is stored at each worker. The computations carried out by each worker is also proportional to the storage load, therefore, we use the term storage load and computation load interchangeably throughout the paper.

We consider a scenario in which a large graph is stored (with redundancy r) at the beginning and then is processed multiple times. Storage takes place in an offline phase while graph processing computations are carried out in an online phase. Given a subgraph allocation $\mathcal{M} = (\mathcal{M}_1, \ldots, \mathcal{M}_K)$ and an output allocation $\mathcal{R} = (\mathcal{R}_1, \ldots, \mathcal{R}_K)$, the distributed processing setup consists of three stages: map phase, communication (shuffle) phase, and reduce phase.

Map Phase: In this phase, workers utilize their allocated subgraphs to compute the corresponding intermediate values. In particular, for every $i \in \mathcal{M}_k$, Worker k computes the intermediate values $\mathbf{g}_i = (\{u_{ji} : j \in \mathcal{N}(i)\})$.

Shuffle Phase: After computing the intermediate values, Worker k constructs a message $Z_k \in \mathbb{R}^{z_k}$ of length z_k, given by a function $Z_k = \phi_k(\{\mathbf{g}_i : i \in \mathcal{M}_k\})$, which consists of the computations required by other workers. In the shuffle phase, Worker k multicasts Z_k to other parties in the system. At the end of the shuffle phase, Worker k has all the information it needs to compute (1) for the nodes in R_k. We formally define the communication load of the shuffle phase as follows.

Definition 2 (Communication Load): The communication load is defined as the total number of messages communicated by K workers during the shuffle phase,

$$L_G(\mathcal{M}, \mathcal{R}, r) \triangleq \sum_{k=1}^{K} z_k \tag{12}$$

for a given subgraph allocation $\mathcal{M} = (\mathcal{M}_1, \ldots, \mathcal{M}_K)$ and output allocation $\mathcal{R} = (\mathcal{R}_1, \ldots, \mathcal{R}_K)$.

Reduce Phase: Worker k uses messages $Z_1, \ldots, Z_K$ communicated during the shuffle phase and local computations $(\{\mathbf{g}_i : i \in \mathcal{M}_k\})$ obtained during map phase to compute (1) for the nodes in $\mathcal{R}_k$, via a decoding function $y_j = \psi_k^j(Z_1, \ldots, Z_K, \{\mathbf{g}_i : i \in \mathcal{M}_k\})$ for $j \in \mathcal{R}_k$.

We wish to investigate the trade-off between the storage and communication. In particular, increasing the storage load at each worker increases the amount of computation that is carried out by each worker, but can reduce the amount of communication required between the workers. An extreme case is when the memory of each worker is large enough to store the entire graph, which corresponds to a storage load of $r = K$. In this case, each worker can process the entire graph locally, which eliminates the need for any communication. However, this setup does not benefit from parallelization (employing multiple workers to distribute the computation load and speed up the processing). Moreover, in practice, for large scale graphs, storing the entire graph is not possible and workers cannot store complete graphs. Our goal is to understand this trade-off, i.e., how much reduction in communication can be achieved with respect to the storage (and computation) load at each worker.

Main Problem: Given a graph G and storage load r, find the optimal subgraph allocation $\mathcal{M}$, output allocation $\mathcal{R}$, and coding scheme that minimizes the communication load in (12).

III. Topology-Aware Coded Computing (TACC)

We now introduce our topology-aware coded computing framework, by providing an illustrative example first.

A. Illustrative Example

Consider the graph $G = (\mathcal{V}, \mathcal{E})$ depicted in Fig. 2 along with a distributed computing setup with $K = 3$ workers. In the first step, the set $\mathcal{V}$ is partitioned into K equal-sized parts and each

[1]We assume that each worker can store a subgraph with an equal number of nodes.

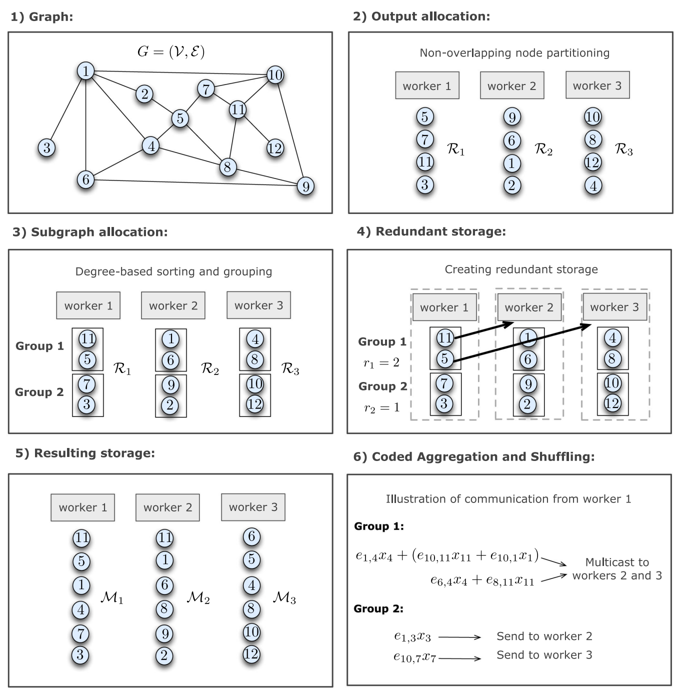

1) Graph:
$G = (\mathcal{V}, \mathcal{E})$
2) Output allocation:
Non-overlapping node partitioning
worker 1
worker 2
worker 3
$\mathcal{R}_1$
$\mathcal{R}_2$
$\mathcal{R}_3$
3) Subgraph allocation:
Degree-based sorting and grouping
Group 1
Group 2
4) Redundant storage:
Creating redundant storage
$r_1 = 2$
$r_2 = 1$
5) Resulting storage:
$\mathcal{M}_1$
$\mathcal{M}_2$
$\mathcal{M}_3$
6) Coded Aggregation and Shuffling:
Illustration of communication from worker 1
$e_{1,4}x_4 + (e_{10,11}x_{11} + e_{10,1}x_1)$
$e_{6,4}x_4 + e_{8,11}x_{11}$
Multicast to workers 2 and 3
$e_{1,3}x_3$
Send to worker 2
$e_{10,7}x_7$
Send to worker 3

Fig. 2. Illustrative example of TACC. Initially, the graph is partitioned into equal-sized parts and each part is assigned to a distinct worker. Next, nodes are sorted and grouped according to their degrees. Nodes are then duplicated at multiple workers (depending on their degree) to create redundancy in storage. In the resulting storage, high degree nodes will be replicated at a larger fraction of workers. Finally, the redundancy in storage and computations is utilized to create multicasting opportunities for efficient communication.

part is associated with a distinct worker. To do so, one can use any partitioning method ranging from random partitioning to more advanced min-cut based partitioning methods [42]. Each worker is responsible from computing the output values in (1) corresponding to the nodes within the part it is assigned to.

In the second step, the nodes in each part are sorted in descending order with respect to their degree, and divided into Q equal-sized chunks. In this example, we have $Q = 2$, therefore each part is divided into 2 groups with 2 nodes in each group. In the third step, we define a parameter r_q associated with part $q = 1, 2$. This parameter indicates that each node in group q is to be stored at r_q workers. For the example at hand, $r_1 = 2$, hence each node in the first group is replicated at 2 workers. On the other hand, for the second group we have $r_2 = 1$, hence the nodes in the second group is stored at a single worker. We call r_q the *storage load* for group $q = 1, 2$.

The node allocation is done according to the following procedure. For each worker, the nodes associated with part q are divided into $\binom{K-1}{r_q-1}$ equal-sized segments, and each segment is replicated at a distinct subset of $r_q - 1$ other workers. As a result, each node in group q is stored at r_q workers in total. By selecting $r_1 > r_2$, one can ensure that each worker allocates a larger fraction of its memory to higher degree nodes.

Finally, workers aggregate the intermediate computations that are targeted for the same destination node, and form coded messages to be communicated to the other workers. For instance, Worker 1 multicasts the coded message $e_{1,4}x_4 + (e_{10,11}x_{11} + e_{10,1}x_1)$ to Workers 2 and 3. The value $e_{10,11}x_{11} + e_{10,1}x_1$ is needed by Worker 3 to evaluate y_{10}, whereas $e_{1,4}x_4$ is needed by Worker 2 to compute y_1 from (1). To recover the missing values, Workers 2 and 3 evaluate $e_{10,11}x_{11} + e_{10,1}x_1$ and $e_{1,4}x_4$, respectively, and remove it from the coded message. Workers communicate the missing values separately for the two groups.

Algorithm 1: Topology-Aware Coded Computing (TACC).

```
1: Initialize graph $G = (\mathcal{V}, \mathcal{E})$, number of workers $K$, parameters $Q$,
   and $\mathbf{r} = (r_1, \ldots, r_Q)$.
   **Subgraph and Output Allocation Phase:**
2: Assign the outputs $\mathcal{R}_k$ and subgraph $\mathcal{M}_k$ determined from
   Algorithm 2 to workers $k \in [K]$.
   **Map Phase:**
3: **for** workers $k \in [K]$ **do**
4:     Map the nodes in $\mathcal{M}_k$ according to Algorithm 3.
   **Coded Aggregation and Shuffling Phase:**
5: **for** workers $k \in [K]$ **do**
6:     Create and multicast the coded messages according to Algorithm 4.
   **Reduce Phase:**
7: **for** workers $k \in [K]$ **do**
8:     Decode the coded messages by following the steps in Algorithm 5.
9:     Compute the outputs (1) for the nodes in $\mathcal{R}_k$.
```

B. Details of the TACC Framework

We now introduce the TACC framework. In essence, TACC aims to reduce the inter-processor communication load by leveraging the interplay between the two strategies illustrated in Fig. 1. These strategies correspond to two potential approaches for handling high degree nodes: one can either store a high degree node at many workers and eliminate the communication cost completely for that node, or alternatively, can distribute the nodes among different parties and take advantage of multicasting opportunities while communicating them. The overall procedure of TACC is presented in Algorithm 1. It consists of the following main phases.

Subgraph and Output Allocation Phase: Initially, the output assignments $\mathcal{R} = (\mathcal{R}_1, \ldots, \mathcal{R}_K)$ are determined by partitioning $\mathcal{V}$ into K equal-sized, non-overlapping parts and assigning each part to a different worker. This can be carried out by using any graph partitioning method. For theoretical analysis, we consider a random partitioning setup, i.e., graph nodes are partitioned into K equal-sized non-overlapping parts randomly. For practical applications, one can use a state-of-the-art min-cut based graph partitioning tool such as [42].

In the next step, the subgraph allocations $\mathcal{M} = (\mathcal{M}_1, \ldots, \mathcal{M}_K)$ are constructed as follows. For each $k \in [K]$, the nodes in $\mathcal{R}_k$ are sorted in descending degree order. The ordered nodes are divided into Q equal-sized groups, such that each group has $\frac{|\mathcal{R}_k|}{Q} = \frac{N}{QK}$ nodes. As a result, the first (last) group consists of the $\frac{N}{QK}$ highest (lowest) degree nodes in $\mathcal{R}_k$. We represent this sorted and grouped version of $\mathcal{R}_k$ by $\widetilde{\mathcal{R}}_k = (\widetilde{\mathcal{R}}_{k1}, \ldots, \widetilde{\mathcal{R}}_{kQ})$.

Each node in group $q \in [Q]$ is then duplicated at r_q workers for a given $r_q \in [K]$ as follows. First, $\widetilde{R}_{kq}$ is randomly divided into $\binom{K-1}{r_q-1}$ equal-sized, non-overlapping parts. Then, each part is stored at a distinct subset of workers of size r_q (including Worker

Algorithm 2: Subgraph and Output Allocation Phase.

```
Input: Graph $G = (\mathcal{V}, \mathcal{E})$, number of workers $K$, parameters $Q$ and
  $\mathbf{r} = (r_1, \ldots, r_Q)$.
Output: Output allocation $\mathcal{R} = (\mathcal{R}_1, \ldots, \mathcal{R}_K)$, subgraph allocation
  $\mathcal{M} = (\mathcal{M}_1, \ldots, \mathcal{M}_K)$.
1: Partition $\mathcal{V}$ into $K$ equal-sized parts $\mathcal{R} = (\mathcal{R}_1, \ldots, \mathcal{R}_K)$.
2: for $k = 1, \ldots, K$
3:   Let $\widetilde{\mathcal{R}}_k \leftarrow \{\mathcal{R}_k$ sorted using node degrees, in descending order$\}$
4:   Divide $\widetilde{\mathcal{R}}_k$ into $Q$ equal-sized groups $(\widetilde{\mathcal{R}}_{k1}, \ldots, \widetilde{\mathcal{R}}_{kQ})$. // $\widetilde{\mathcal{R}}_{k1}$
     holds the highest degree nodes from $\widetilde{\mathcal{R}}_k$.
5:   for $q = 1, \ldots, Q$
6:     Divide $\widetilde{\mathcal{R}}_{kq}$ into $\binom{K-1}{r_q-1}$ equal-sized, non-overlapping parts.
7:     Assign each part to a distinct subset of $r_q$ workers (including
       Worker $k$).
       // This is possible since there are exactly $\binom{K-1}{r_q-1}$ distinct subsets.
8: for $k = 1, \ldots, K$
9:   Return the subgraph $\mathcal{M}_k$ as the collection of all the nodes assigned
     to Worker $k$.
```

k). Note that this is possible since there are exactly $\binom{K-1}{r_q-1}$ such subsets of $[K]$. At the end of this step, each subset of workers $\mathcal{S} \subseteq [K]$ of size r_q will exclusively store $\frac{N}{Q\binom{K}{r_q}}$ nodes, for $q \in [Q]$. We denote the set of nodes stored exclusively by the workers in $\mathcal{S}$ with $\mathcal{M}_{\mathcal{S}}^q$, and define,

$$\mathcal{M}_k^q \triangleq \{\mathcal{M}_{\mathcal{S}}^q : k \in \mathcal{S} \subseteq [K], |\mathcal{S}| = r_q\} \tag{13}$$

as the subgraph stored at Worker k for group $q \in [Q]$. The subgraph allocated to Worker k is then given by $\mathcal{M}_k = \bigcup_{q=1}^{Q} \mathcal{M}_k^q$. The storage load of the system is given by,

$$r = \frac{\sum_{k=1}^{K} |\mathcal{M}_k|}{N} = \frac{1}{Q} \sum_{j=1}^{Q} r_j. \tag{14}$$

Remark 2: Our graph allocation strategy is based on two system parameters. The first parameter, Q, groups the vertices that have similar degrees. The second parameter, $\mathbf{r} = (r_1, \ldots, r_Q)$, indicates that each node in group q is duplicated at r_q workers. By selecting $r_1 \geq \ldots \geq r_Q$, we ensure that high degree nodes are stored at a larger fraction of workers.

We note that one can always find parameters Q and $\mathbf{r}$ so that the maximum memory size of the workers is not exceeded.[2] That is, if the workers have no additional memory for duplicate nodes, i.e., each worker can only store $\frac{N}{K}$ nodes, we select $Q = 1$ and $r_1 = 1$. On the other hand, if the memory of each worker is large enough to store the entire graph, we select $Q = 1$ and $r_1 = K$. Hence, memory constraints of the workers can always be satisfied. Finally, if $r_i = r_j$ for some $i \neq j$, one can combine the nodes within these two groups at each worker, and treat them as a single group during the duplication operation as well as the remaining steps of the algorithm. The individual steps of this phase are presented in Algorithm 2.

Remark 3: In order to keep our framework flexible, we allow TACC to be initialized with any graph partitioning strategy. Then, the communication load achieved by our coding framework will be no greater than a system that parallelizes the graph with the same initial partitioning, but without coding. This is due to the fact that as we use multicasting, there is no additional cost when some information is needed by more than one worker. The only additional cost that may be introduced with the added redundancy is if the algorithm is an iterative one. Then, the updated node values after the reduce phase need to be communicated back to the workers who need them for the next round. This cost, however, is typically smaller than the communication cost of the shuffle phase [18], [43], as long as r is relatively small compared to the total number of workers and the average node degree.

[2]We assume that workers have equal-sized memory and each worker can store at least $\frac{1}{K}$ fraction of the nodes

Algorithm 3: Map Phase.

```
Input: Subgraph allocation $\mathcal{M} = (\mathcal{M}_1, \ldots, \mathcal{M}_K)$.
Output: Intermediate values $\{u_{ji} : i \in \mathcal{M}_k, j \in \mathcal{N}(i)\}$ for $k \in [K]$.
1: for Workers $k \in [K]$ do
2:   for $i \in \mathcal{M}_k$
3:     for $j \in \mathcal{N}(i)$
4:       Compute the intermediate value $u_{ji}$ using (4).
```

Map Phase: In this phase, Worker k maps the nodes in $\mathcal{M}_k$ using the map function from (4), to obtain the intermediate values,

$$\{u_{ji} : i \in \mathcal{M}_k, j \in \mathcal{N}(i)\}. \tag{15}$$

Each worker maps a total number of $|\mathcal{M}_k| = \frac{N\sum_{j=1}^{Q} r_j}{QK}$ nodes. This phase follows the standard mapping stage of distributed algorithms, and unlike the previous stage, is not specific to the proposed algorithm. The individual steps of this phase are provided in Algorithm 3.

Coded Aggregation and Shuffling Phase: In this phase, we utilize the redundancy created during graph allocation for coded multicasting. For each group $q \in [Q]$, consider a set of workers $\mathcal{S}$ of size $|\mathcal{S}| = r_q + 1$. Define,

$$\mathcal{U}_{\mathcal{S}\backslash\{k\}}^k = \{u_{ji} : j \in \mathcal{R}_k, (i,j) \in \mathcal{E}, i \in \mathcal{M}_{\mathcal{S}\backslash\{k\}}^q\} \tag{16}$$

as the set of intermediate values needed by worker k and known exclusively by workers in $\mathcal{S}\backslash\{k\}$, where $\mathcal{M}_{\mathcal{S}\backslash\{k\}}^q = \bigcap_{k' \in \mathcal{S}\backslash\{k\}} \mathcal{M}_{k'}^q$ from (13). Next, workers in $\mathcal{S}\backslash\{k\}$ aggregate the intermediate values required for the computations in (1),

$$u_j = \sum_{i \in \mathcal{M}_{\mathcal{S}\backslash\{k\}}^q} u_{ji}, \quad j \in \mathcal{R}_k \tag{17}$$

and form a vector $\mathbf{u}_{\mathcal{S}\backslash\{k\}}^k = (\{u_j : j \in \mathcal{R}_k\})$. Then, $\mathbf{u}_{\mathcal{S}\backslash\{k\}}^k$ is evenly split into r_q segments. Segment s is denoted by $\mathbf{u}_{\mathcal{S}\backslash\{k\},s}^k$. Each segment is assigned to a distinct worker in $s \in \mathcal{S}\backslash\{k\}$. This process is then repeated for every $k \in \mathcal{S}$. At the end, each worker is assigned to a total number of $\binom{|\mathcal{S}|-1}{r_q-1} = r_q$ segments. Worker $j \in \mathcal{S}$ then computes a coded message by combining the assigned segments,

$$Z_j^{\mathcal{S}} = \sum_{k \in \mathcal{S}\backslash\{j\}} \mathbf{u}_{\mathcal{S}\backslash\{k\},j}^k \tag{18}$$

via zero-padding if needed to ensure that the segment sizes are equal, and multicasts the message to the workers in $\mathcal{S}\backslash\{j\}$. This

Algorithm 4: Coded Aggregation and Shuffling Phase.

```
1: for $q = 1, \ldots, Q$
2:   for each $\mathcal{S} \subseteq [K]$ of size $r_q + 1$
3:     for each $k \in \mathcal{S}$, workers in $\mathcal{S}\backslash\{k\}$ do
4:       Aggregate the intermediate values using (17) to find $u_j$ for
         $j \in \mathcal{R}_k$.
5:       Construct the vector $\mathbf{u}^k_{\mathcal{S}\backslash\{k\}} = (\{u_j : j \in \mathcal{R}_k\})$.
6:       Split $\mathbf{u}^k_{\mathcal{S}\backslash\{k\}}$ into $r_q$ equal-sized segments, given by $\mathbf{u}^k_{\mathcal{S}\backslash\{k\},s}$
         for $s \in \mathcal{S}\backslash\{k\}$.
7:       Assign each segment $s$ to a distinct worker $s \in \mathcal{S}\backslash\{k\}$.
8:     for each worker $j \in \mathcal{S}$ do
9:       Compute the coded message $Z^{\mathcal{S}}_j$ from (18).
10:      Multicast $Z^{\mathcal{S}}_j$ to the workers in $\mathcal{S}\backslash\{j\}$.
```

Algorithm 5: Reduce Phase.

```
1: for $q = 1, \ldots, Q$
2:   for each $\mathcal{S} \subseteq [K]$ of size $r_q + 1$
3:     for each worker $k \in \mathcal{S}$ do
4:       for $j \in \mathcal{S}\backslash\{k\}$
5:         Receive $Z^{\mathcal{S}}_j$ from worker $j$.
6:         Remove the known segments to compute the unknown
           segment from (19).
```

process is repeated for each subset $\mathcal{S} \subseteq [K]$ of size r_q and group $q \in [Q]$. This phase is presented in Algorithm 4.

Reduce Phase: During this phase, each worker recovers the intermediate values needed to compute (1). For each group $q \in [Q]$, there are $\binom{K}{r_q+1}$ subsets $\mathcal{S}$ of size $r_q + 1$. Worker k is involved in $\binom{K-1}{r_q}$ of them, and within each subset, there are r_q distinct segments $\mathbf{u}^k_{\mathcal{S}\backslash\{k\},s}$ known by other workers $s \in \mathcal{S}\backslash\{k\}$ and needed by Worker k. Hence, Worker k needs to recover $\sum_{q=1}^{Q} \binom{K-1}{r_q} r_q$ distinct segments in total. Consider a group q and set $\mathcal{S} \subseteq [K]$ of size $r_q + 1$. For workers $j, k \in \mathcal{S}$ and $j \neq k$, there are $\binom{|\mathcal{S}|-2}{r_q-2} = r_q - 1$ subsets of $\mathcal{S}$ that has size r_q and contain both j and k. Among the r_q segments associated with Worker j, $r_q - 1$ of them are known also by Worker k, who needs the remaining segment. After receiving $Z^{\mathcal{S}}_j$ from workers $j \in \mathcal{S}\backslash\{k\}$, Worker k can remove the known segments $\{\mathbf{u}^{k'}_{\mathcal{S}\backslash\{k'\},j}\}_{k' \in \mathcal{S}\backslash\{j,k\}}$ to compute the unknown segments,

$$\mathbf{u}^k_{\mathcal{S}\backslash\{k\},j} = Z^{\mathcal{S}}_j - \sum_{k' \in \mathcal{S}\backslash\{j,k\}} \mathbf{u}^{k'}_{\mathcal{S}\backslash\{k'\},j} \quad \forall j \in \mathcal{S}\backslash\{k\}. \tag{19}$$

At the end, Worker k recovers r_q distinct segments for the subset $\mathcal{S}$. Since there are $\binom{K-1}{r_q}$ such subsets per group q, Worker k will recover all of the required $\sum_{q=1}^{Q} \binom{K-1}{r_q} r_q$ segments. The steps of this phase are presented in Algorithm 5.

Remark 4: Algorithm 1 defines our coding scheme for a general graph, where the coded messages are created as shown in (17) and (18). Our coding scheme is defined algorithmically, and presents flexibility in choosing the order in which the nodes are aggregated, as long as they are in the specified sets given in (17) and (18). Hence, any order can be used for aggregation, as long as the order is known at the decoder.

IV. Expected Communication Load Analysis for Random Graphs

In this section, we analyze the expected communication load of TACC for random graph structures.

A. General Theoretical Bounds

Introduced independently by Erdös-Rényi and Gilbert, random graphs have proved to be a powerful tool for understanding graph behavior and demonstrating the existence of various graph properties [44], [45]. The classical Erdös-Rényi random graph model considers a graph in which an edge between any two vertices occurs with some fixed probability $0 \leq p \leq 1$, independently from other edges. As a result, every vertex has an average degree equal to pN, which can be restrictive in modeling real-world graphs.

Generalized random graphs extend the classical model to describe a wider range of scenarios, by allowing the vertices to take arbitrary degree distributions [21]–[23]. Specifically, edge probabilities in these graphs are governed by vertex weights, which can be deterministic [21], [22] or random [23]. The average degree of each vertex is determined by its weight. This in turn facilitates modeling graphs with irregular degree distributions. Our focus is on the following class of generalized random graphs.

Definition 3 (Generalized Random Graph): Define W weights $N \geq \lambda_1 \geq \ldots \geq \lambda_W \geq 0$. Assign each vertex $i \in \mathcal{V}$ a random weight W_i such that,

$$P(W_i = \lambda_w) = \frac{1}{W} \text{ for } w \in [W]. \tag{20}$$

Then, draw an edge between each pair of vertices i and j with probability $\frac{W_i W_j}{\sum_{l=1}^{N} W_l}$.

Accordingly, node i has an average degree of $\sum_{j=1}^{N} \frac{W_i W_j}{\sum_{l=1}^{N} W_l} = W_i$, i.e., a fraction of $\frac{1}{W}$ nodes is expected to have an average degree of λ_w for each $w \in [W]$. A large variety of graph structures can be realized through this model, by creating different degree structures using W and $\{\lambda_j\}_{j \in [W]}$. A special case is the classical random graph model, which can be obtained by setting $W = 1$ and $\lambda_1 = pN$. In this case, every vertex has a fixed weight (an average degree of λ_1), and an edge exists between any two vertices with probability $\frac{\lambda_1}{N} = p$. We next characterize the minimum expected communication load for random graphs as a function of the storage load.

Definition 4: Given a storage load r, we define the minimum expected communication load as,

$$L^*(r) = \inf_{\mathcal{M},\mathcal{R}} E_G[L_G(\mathcal{M},\mathcal{R},r)] \tag{21}$$

where the expectation is taken over all graph realizations.

$L^*(r)$ identifies the trade-off between the storage load and the expected communication load of the system. The optimal storage-communication strategy that achieves $L^*(r)$ is an open problem in general. Hence, we elaborate on our design principles through asymptotic bounds on (21).

We first present a lower bound on $L^*(r)$ for a given storage load.

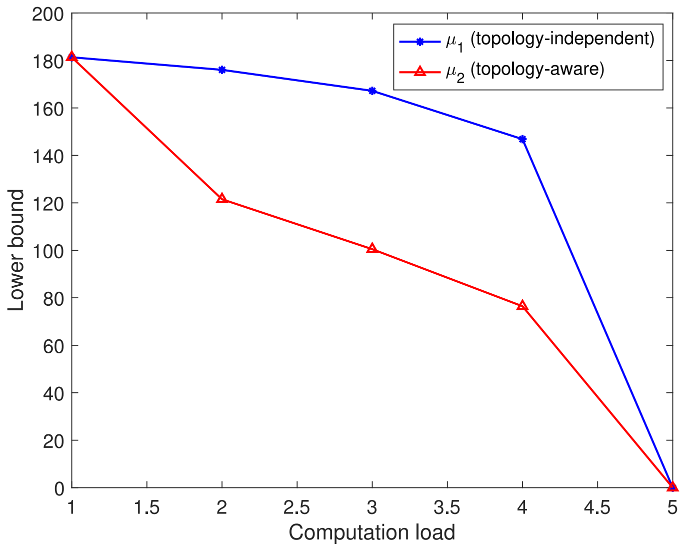


Fig. 3. Demonstration of the lower bounds μ_1 and μ_2 from Theorem 2 with respect to the storage (computation) load for a generalized random graph.

Theorem 1: (Lower Bound) For any computation scheme in which each worker stores $\frac{r_q N}{WK}$ nodes with average degree (weight) $\lambda_{1+\frac{(q-1)W}{Q}}, \ldots, \lambda_{\frac{qW}{Q}}$ for $q \in [Q]$,

$$L^*(r) \geq \frac{N}{KW}\sum_{i=1}^{W}\left(1-\prod_{q=1}^{Q}\prod_{j=1+\frac{(q-1)W}{Q}}^{\frac{qW}{Q}}\left(1-\frac{\lambda_i\lambda_j}{\frac{N}{W}\sum_{t=1}^{W}\lambda_t}\right)^{\frac{N}{W}(1-\frac{r_q}{K})}\right). \tag{22}$$

Proof: The proof is provided in Appendix A. ■

In the following, we show that the lower bound of (22) decreases when more storage is allocated to high degree nodes (compared to low degree nodes), while keeping the total storage of each worker the same. In other words, it is better to replicate more those nodes with higher degree, as proposed in our topology-aware graph allocation, since it reduces the minimum number of messages required by each worker.

Theorem 2: Consider any computation scheme in which each worker stores $\frac{rN}{WK}$ nodes with degree λ_i, where $i \in [W]$ (i.e., the same allocation of redundancy is used for all nodes, regardless of their degree). Let μ_1 be the lower bound from (22) for this setup. Next, consider another scheme where each worker stores $\frac{r_i N}{WK}$ nodes with degree λ_i, for some $r_1 \geq \ldots \geq r_W$ such that $r = \frac{1}{W}\sum_{i=1}^{W} r_i$ (i.e., high degree nodes are replicated more, while keeping the total number of nodes stored at each worker the same). Let μ_2 be the lower bound from (22) for this second case. Then, $\mu_2 \leq \mu_1$.

Proof: Please see Appendix B. ■

While Theorem 2 provides some support to our strategy of linking replication to node degree, it is based on a lower bound for $L^*(r)$, which suggests that the exact conditions for which irregular node allocation is strictly better are still an open question. Fig. 3 provides an illustration of the lower bounds μ_1 and μ_2 from Theorem 2 with respect to the storage (computation) load r for a generalized random graph with $N = 1000$ nodes, degree parameters $W = 5$ and $(\lambda_1, \ldots, \lambda_W) = (200, 20, 10, 5, 1)$, and $K = 5$ workers. For μ_2, the storage load at each worker is selected as follows,

$$\mathbf{r} = (r_1, \ldots, r_5) = \begin{cases} (1,1,1,1,1) \text{ for } r = 1 \\ (5,2,1,1,1) \text{ for } r = 2 \\ (5,3,3,3,1) \text{ for } r = 3 \\ (5,4,4,4,3) \text{ for } r = 4 \\ (5,5,5,5,5) \text{ for } r = 5. \end{cases} \tag{23}$$

We observe in Fig. 3 that the lower bound for the communication load of topology-aware coding is up to 48% lower than that of topology-independent coding.

Remark 5: $\mathbf{r}$ is a system parameter to be determined by the system designer, by taking into account the graph structure and memory size of each worker. We have found in our experiments that the following strategy, which is motivated by Theorem 2, provides a good methodology for selecting $\mathbf{r}$ for graphs with irregular degree structures. Initially, set $\mathbf{r} = (r_1, \ldots, r_Q) = (r, \ldots, r)$ where r is determined by the memory size of the workers, then gradually reduce the last term and increase the first term by equal amounts, i.e., set $r_Q - \kappa$ and $r_1 + \kappa$ for some $\kappa \in \mathbb{N}$ so that $1 \leq r_Q - \kappa$ and $r_1 + \kappa \leq K$.

Next, we provide an upper bound on $L^*(r)$ for the generalized random graph model by using TACC.

Theorem 3: (Upper Bound) For topology-aware coded graph processing with aggregation, the communication load for the generalized random graph model is bounded from above by,

$$L^*(r) \leq \sum_{q=1}^{Q}\frac{K}{r_q s_q}\binom{K-1}{r_q}\Bigg(\ln(r_q) + \sum_{m=1}^{W}\frac{N}{WK} \times \ln\Bigg(\prod_{l=1+\frac{(q-1)W}{Q}}^{\frac{qW}{Q}}\left(1-\frac{\lambda_m\lambda_l}{\frac{N}{W}\sum_{t=1}^{W}\lambda_t}\right)^{\frac{N}{W\binom{K}{r_q}}} + e^{s_q}\Bigg(1-\prod_{l=1+\frac{(q-1)W}{Q}}^{\frac{qW}{Q}}\left(1-\frac{\lambda_m\lambda_l}{\frac{N}{W}\sum_{t=1}^{W}\lambda_t}\right)^{\frac{N}{W\binom{K}{r_q}}}\Bigg)\Bigg)\Bigg), \tag{24}$$

for any $s_1, \ldots, s_Q > 0$ and Q such that $\frac{W}{Q} \in \mathbb{Z}$.

Proof: The proof is provided in Appendix C. ■

We note that the two key aspects of TACC, namely topology-aware subgraph allocation and coded aggregation can be applied independently from one another. As such, one can obtain the expected communication load specifically for the coded aggregation setup by setting $Q = 1$ in Theorems 3 and 1. In particular, by letting $Q = 1$ and $r_1 = r$, we obtain a topology-independent subgraph allocation model where each node is replicated at an equal number of workers, independent of its degree pattern. This special case is related to the coded computing framework of [18], the difference being the coded aggregation phase. Specifically, we construct the coded messages over the aggregated intermediate values, instead of the intermediate values themselves as in [18]. Formally, the upper and lower bounds on the expected communication load for coded aggregation is given as follows.

Corollary 1: For the generalized random graph model, the communication load for coded aggregation satisfies,

$$L^*(r) \leq \frac{K}{rs}\binom{K-1}{r}\left(\ln(r) + \sum_{m=1}^{W}\frac{N}{WK}\right.$$
$$\times \ln\left(\prod_{l=1}^{W}\left(1 - \frac{\lambda_m\lambda_l}{\frac{N}{W}\sum_{t=1}^{W}\lambda_t}\right)^{\frac{N}{W\binom{K}{r}}}\right.$$
$$\left.\left. + \, e^s\left(1 - \prod_{l=1}^{W}\left(1 - \frac{\lambda_m\lambda_l}{\frac{N}{W}\sum_{t=1}^{W}\lambda_t}\right)^{\frac{N}{W\binom{K}{r}}}\right)\right)\right)$$
$$\text{for any } s > 0. \tag{25}$$

Corollary 2: For any computation scheme in which each worker stores $\frac{rN}{WK}$ nodes with average degree (weight) $\lambda_1, \ldots, \lambda_W$, the communication load satisfies,

$$L^*(r) \geq \frac{N}{KW}\sum_{i=1}^{W}\left(1 - \prod_{j=1}^{W}\left(1 - \frac{\lambda_i\lambda_j}{\frac{N}{W}\sum_{t=1}^{W}\lambda_t}\right)^{\frac{N}{W}(1-\frac{r}{K})}\right). \tag{26}$$

B. Erdös-Rényi Graph Model

We now specialize our bounds from Corollary 1 and 2 to the Erdös-Rényi random graph with edge probability p, such that each edge occurs with a fixed probability $0 < p < 1$. By setting $W = 1$ and $\lambda_1 = pN$ in (25), we obtain the following upper bound for the Erdös-Rényi random graph with coded aggregation,

$$L^*(r) \leq \min_{s>0}\frac{K}{rs}\binom{K-1}{r}\left(\ln(r) + \frac{N}{K}\right.$$
$$\left.\times \ln\left((1-p)^{\frac{N}{\binom{K}{r}}} + e^s\left(1 - (1-p)^{\frac{N}{\binom{K}{r}}}\right)\right)\right) \tag{27}$$
$$\leq \frac{K}{r}\binom{K-1}{r} + \frac{N}{r}\binom{K-1}{r}$$
$$\times\left(\frac{\ln\left(r - (r-1)(1-p)^{\frac{N}{\binom{K}{r}}}\right)}{\ln r}\right) \tag{28}$$

where (28) follows from setting $s = \ln(r)$ in (27). As $N \to \infty$, we observe that $(1-p)^{\frac{N}{\binom{K}{r}}} \to 0$ for any $p > 0$ and as a result, (28) reduces to,

$$L^*(r) \leq \frac{N}{r}\binom{K-1}{r}. \tag{29}$$

We next consider the lower bound from Corollary 2 for the Erdös-Rényi random graph with edge probability p. By setting $W = 1$ and $\lambda_1 = pN$ in (22), for this case we obtain,

$$L^*(r) \geq \frac{N}{K}\left(1 - (1-p)^{N(1-\frac{r}{K})}\right). \tag{30}$$

In the following, we compare the expected communication load for an Erdös-Rényi random graph with edge probability p for the three setups, namely, our coded aggregation strategy from (29), the uncoded transmission strategy from [7], and coding without aggregation from [18]. For the Erdös-Rényi graph, the expected communication load for uncoded transmission from [7] is,

$$L^*_{\text{uncoded}}(r) = N^2 p\left(1 - \frac{r}{K}\right) \tag{31}$$

which follows from the fact that each worker stores $\frac{Nr}{K}$ nodes, and needs to process the output values for $\frac{N}{K}$ nodes. For each node to be processed, an average number of $p(N - \frac{Nr}{K})$ intermediate values are needed from other workers, each intermediate value corresponding to an edge between the node to be processed by the worker and nodes mapped by the remaining workers. As a result, a total number of $K\frac{N}{K}p(N - \frac{Nr}{K}) = N^2 p(1 - \frac{r}{K})$ messages are needed by all K workers. On the other hand, the communication load for coded computing without aggregation is, from [18],[3]

$$L^*_{\text{no-agg}}(r) = \frac{N^2 p}{r}\left(1 - \frac{r}{K}\right). \tag{32}$$

When compared with (29), it can be observed that the communication load for both uncoded transmission and coded computing without aggregation scales with respect to N^2, whereas our aggregation strategy reduces this to a factor of N. Also, the communication load for uncoded transmission and coded computing without aggregation scales linearly with respect to p, hence for dense graphs the communication load grows linearly with respect to the edge probability. On the other hand, as $N \to \infty$, the communication load for coded aggregation does not scale with respect to the edge probability p. Therefore, the benefits of aggregation becomes more significant as the graph size becomes larger and the graph connectivity becomes denser, making coded aggregation a viable setup for large-scale graph processing.

C. Stochastic Block Model

We next specialize our bound to the K-block stochastic block model. To do so, we consider a graph of N nodes that consists of K equal-sized clusters where the probability of an edge between any two nodes is $0 < p < 1$ if the nodes belong to the same cluster and $0 < q < 1$ if the nodes belong to different clusters, where $q < p$. We initially partition the graph across K workers such that each worker is assigned to a distinct block, then apply our coded aggregation technique.

For this graph model, the expected degree is the same for all nodes, since a node is connected to each node within the same cluster with probability p and each node from different clusters with probability q, and the cluster sizes are equal. Due to this homogeneity in the degree structures of the nodes, we use a single group in our algorithm by selecting $Q = 1$. Then, the computation load for the K-block stochastic block model can be bounded by,

$$L^*(r)$$
$$\leq \frac{K}{r}\binom{K-1}{r} + \frac{N}{r}\binom{K-1}{r}\left(\frac{\ln\left(r - (r-1)(1-q)^{\frac{N}{\binom{K}{r}}}\right)}{\ln r}\right) \tag{33}$$

[3]We note that [18] reports the normalized communication load where (32) is divided by N^2.

where the proof follows along the lines of Appendix C by setting $Q = 1$ and calculating the expectation by using the given initial partitioning strategy and edge probabilities. Note that the difference between (33) and the Erdös-Rényi bound from (28) is that the communication load for the K-block stochastic block model depends on the inter-cluster edge probability q.

On the other hand, the communication load for the K-block stochastic block model with uncoded transmission is,

$$L^*_{\text{uncoded}}(r) = N^2 q \left(1 - \frac{r}{K}\right). \tag{34}$$

Lastly, we consider the communication load for coding without aggregation from [18]. We first note that the stochastic block model considered in [18] is a 2-block stochastic model, as opposed to the K-block stochastic model considered in this work. For the 2-block stochastic model with equal sized blocks, the communication load from [18] is given by,

$$L^*_{\text{no-agg}}(r) = \frac{N^2(p+q)}{2r}\left(1 - \frac{r}{K}\right), \tag{35}$$

where the dependency of the bound on p is due to the fact that in the node allocation strategy employed by [18], part of the nodes allocated to each worker comes from the first block and the remaining part comes from the second block. Hence, a worker may require information about a given node both from the neighbors within the same block as well as neighbors from different blocks.

By comparing (33) with (34) and (35), we observe that for the stochastic block model, the communication load for coding with aggregation from (33) scales linearly with respect to the number of nodes N whereas for the two baselines from (34) and (35) this scaling is quadratic, i.e., with respect to N^2. Moreover, as observed from (33), for our coding with aggregation strategy, the dependency of the communication load on the edge probabilities decreases as $N \to \infty$.

D. Random Bipartite Graph Model

Finally, we specialize our bound to the random bipartite graph model. To do so, we consider a bipartite graph with N nodes consisting of two sets of nodes of size $\frac{N}{2}$. The probability that an edge exists between two nodes belonging to different sets is $0 < p < 1$, whereas no edge exists between the nodes belonging to the same set.

In this setup, for the initial partitioning of the nodes to the workers, we adopt the following strategy. We divide each set into K parts, and assign each worker a distinct part from each set. As a result, each worker will receive a total number of $\frac{N}{K}$ nodes, $\frac{N}{2K}$ nodes from the first set and $\frac{N}{2K}$ nodes from the second set. The intuition behind this strategy is that in a random bipartite graph, edges only occur across the nodes belonging to different sets. Therefore, storing nodes that belong to different sets at the local memory of the workers will reduce the number of edges across the workers, compared to storing nodes from a single set. Then, we apply our coded aggregation strategy.

Note that in this setup also, the expected degree is the same for every node, therefore, the degree structure is homogeneous across the nodes. Accordingly, we again use a single group in our algorithm by selecting $Q = 1$. Then, the expected communication load for the random bipartite graph model with coded aggregation can be bounded as,

$$L^*(r) \le \frac{K}{r}\binom{K-1}{r} + \frac{N}{r}\binom{K-1}{r}\left(\frac{\ln\left(r - (r-1)(1-p)^{\frac{N}{2\binom{K}{r}}}\right)}{\ln r}\right) \tag{36}$$

where the proof again follows along the lines of Appendix C by setting $Q = 1$ and using the given graph allocation strategy and edge probability.

In contrast, the expected communication load for the random bipartite graph model by using uncoded transmission from [7] is,

$$L^*_{\text{uncoded}}(r) = \frac{N^2}{2} p \left(1 - \frac{r}{K}\right), \tag{37}$$

whereas the expected communication load by using coding without aggregation from [18] is,

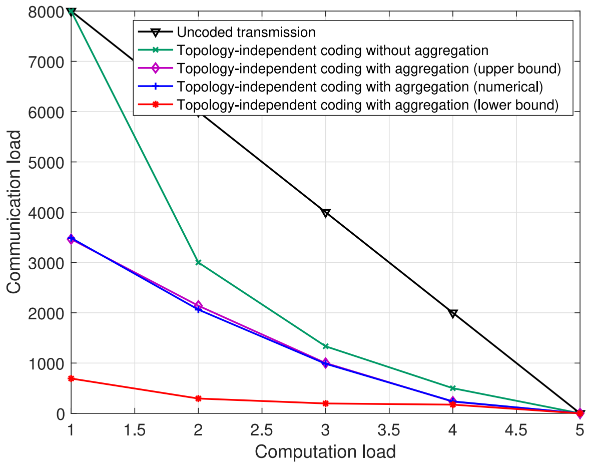
$$L^*_{\text{no-agg}}(r) = \frac{N^2 p}{2r}\left(1 - \frac{2r}{K}\right). \tag{38}$$

By comparing (36) with (37) and (38), we observe that for the random bipartite graph also, the communication load for coding with aggregation scales linearly with respect to the number of nodes N, whereas for the two benchmarks from (37) and (38), this scaling is quadratic. Moreover, as $N \to \infty$, the dependency of the communication load to the edge probability decreases, but with a slower rate than the Erdös-Rényi graph. The intuition behind this is that, as N grows, the probability that a node is connected to at least one other node grows at a faster rate in an Erdös-Rényi graph than a random bipartite graph with the same parameter p. In the former, each node can be connected to any of the $N - 1$ other nodes with the same probability, whereas in the latter, each node can be connected to $\frac{N}{2}$ other nodes. Due to aggregation, our communication load depends on whether or not a node has neighbors belonging to a different worker, rather than the number of neighbors. In the Erdös-Rényi graph, this information converges more quickly as N grows, since overall each node will have a higher probability of being connected, than the random bipartite graph.

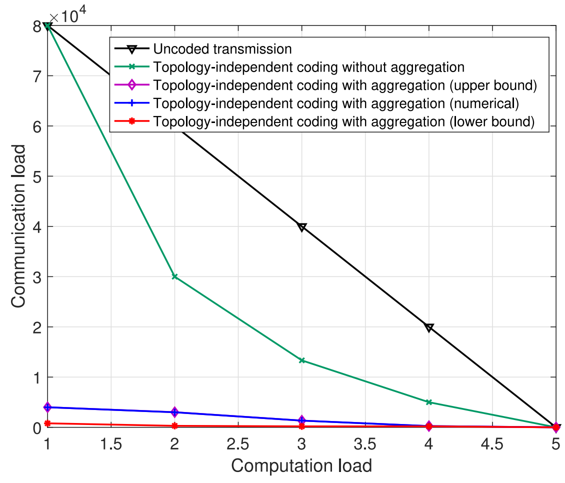
E. Numerical Evaluations

In the remainder of this section, we illustrate the communication-computation trade-off of our coded graph processing framework for various random graph structures.

1) Erdös-Rényi Graphs: Initially, we investigate the benefits of aggregation in reducing the communication load, by setting $Q = 1$ and considering an Erdös-Rényi graph with edge probability p. Then, we compare the communication load for coded aggregation from (29) with uncoded transmission from (31) and coded computing without aggregation from (32). Fig. 4a and 4b



(a)



(b)

Fig. 4. Communication-computation trade-off for Erdös-Rényi graph with (a) $p = 0.01$, (b) $p = 0.1$, respectively.

show the communication load for an Erdös-Rényi graph with $N = 1000$ nodes and $K = 5$ workers. Fig. 4a compares the communication load when $p = 0.01$, with an average degree of 10 for each node. Fig. 4b compares the communication load when $p = 0.1$, with an average degree of 100 for each node.

Fig. 4a and 4b show that aggregation can have a great impact in reducing the communication load, moreover, the benefits of aggregation increases as the graph becomes denser. In particular, we observe from Fig. 4b that even when the storage load is 1, which represents the case when there is no overlap between the subgraphs stored at different workers, aggregation itself reduces the communication load by 95% compared to the other schemes. When the storage load is increased to 4, coded aggregation reduces the communication load by 96% with respect to coding without aggregation, and 99% with respect to uncoded transmission.

2) Barabási-Albert Graphs: We next demonstrate the benefits of topology-aware coded computing, by considering a Barabási-Albert random graph. Unlike the Erdös-Rényi random graph where the average degree of each node is equal, real-world networks often exhibit highly asymmetric degree structures.

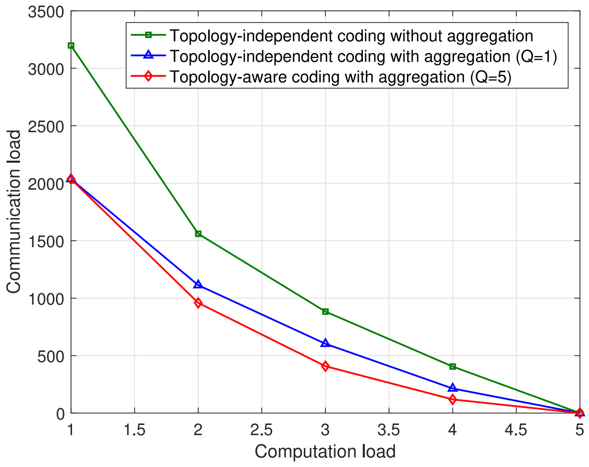


Fig. 5. Communication-computation trade-off for the Barabási-Albert graph.

The degree distribution of many real-world graphs, such as the World Wide Web, follow a power-law, in which the fraction of nodes in the graph with degree k scales with respect to $k^{-\gamma}$ for some parameter γ. These graphs are also known as scale-free networks.

Barabási-Albert graph is a random graph proposed to model the behavior of such scale-free networks [46]. To do so, it starts with m seed nodes, and iteratively adds a new node and k links from the new node to the existing nodes, where the probability that the new node will connect to an existing node is proportional to the existing node's degree. Therefore, higher degree nodes are more likely to link to new nodes, which is also known as the preferential attachment model. In our setup, we consider a Barabási-Albert graph with $N = 1000$ nodes and $m = k = 2$.

We consider a distributed system with $K = 5$ workers, and simulate the following three scenarios. The first one is coded computing without aggregation from [18]. The second one is topology-independent coded aggregation, obtained by setting $Q = 1$. The third scenario is topology-aware coded computing with aggregation, where we let $Q = 5$. We then compare the communication load for the three setups, by keeping the memory size of each worker the same, that is, each worker can store the same number of nodes in all three scenarios. To do so, we denote storage load for the first two scenarios as $r \in [K]$, and select the storage load $\mathbf{r} = (r_1, \ldots, r_Q)$ for the third scenario as in (23) so that (14) is satisfied. As a result, the average storage load is the same for all three scenarios.

Fig. 5 shows the communication-computation trade-off for the three setups. As expected, for $r = 1$, i.e., when there is no overlap between the subgraphs stored at different workers, performance of topology-independent and topology-aware coded aggregation schemes are the same, and both are better than coding without aggregation by 36%. In this case, the communication load for topology-independent coding without aggregation is very large compared to the remaining two setups. As we increase the storage load, topology-aware coding consistently outperforms the topology-independent setup. For $r = 4$, topology-aware coding provides up to 44% improvement

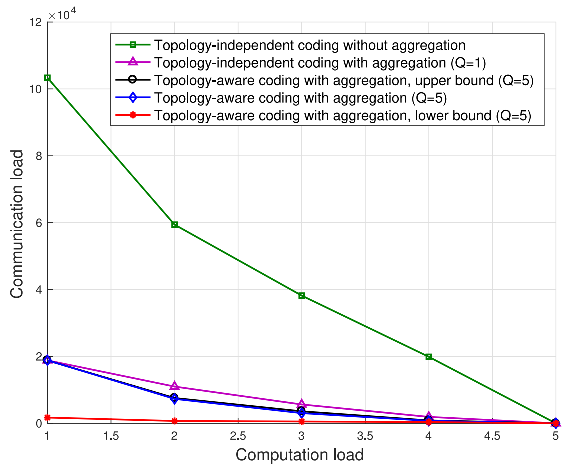


Fig. 6. Communication-computation trade-off for generalized random graph.

over topology-independent coding with aggregation, whereas the gain of topology-independent coding with aggregation over topology-independent coding without aggregation is 47%. As such, topology-aware coding with aggregation leads up to 70% improvement over topology-independent coding without aggregation. Finally, coding with aggregation consistently outperforms coding without aggregation.

3) Generalized Random Graphs: Lastly, we investigate the performance of topology-aware coded computing on a generalized random graph. To do so, we consider a graph with $N = 10000$ nodes. For the node weights, we let $W = 10$,

$$(\lambda_1, \ldots, \lambda_W) = (100, 4, 4, 4, 4, 2, 2, 2, 2, 1), \qquad (39)$$

and assign a weight to each node uniformly at random as indicated in (20). Edges are constructed according to Definition 3. Accordingly, we expect 10% of the nodes to have a degree of 100, 40% of the nodes to have a degree of 4, 40% to have a degree of 2, and 10% to have a degree of 1. As such, the generated graph has a highly irregular degree structure. We consider a distributed system with $K = 5$ workers and simulate the three scenarios described in Section IV-2. For the topology-aware coded aggregation scenario, we choose the same parameters from (23).

Fig. 6 demonstrates the communication load for the three setups versus storage load. When $r = 1$, performance of topology-independent and topology-aware coded aggregation schemes are equal, and better than the coding scheme without aggregation by 81%. As storage load is increased, topology-aware coding consistently outperforms the topology-independent scenario, leading to up to 61% improvement when $r = 4$. Compared to coding without aggregation, both setups consistently perform better, where topology-aware coded aggregation leads up to a 96% improvement when $r = 4$. Lastly, we show that there exists scenarios for which topology-aware subgraph allocation is strictly better than one that does not take into account the graph topology.

Corollary 3: Coded graph processing with topology-aware graph allocation ($Q > 1$) can strictly outperform coding with topology-independent graph allocation ($Q = 1$).

Proof: We prove this result by demonstrating a scenario for which the lower bound for the topology-independent subgraph allocation ($Q = 1$) is strictly greater than the upper bound for a degree-aware subgraph allocation ($Q > 1$). To do so, we again consider the generalized random graph with $N = 10000$ nodes with weights assigned randomly from (39). We first consider a scenario with $Q = 1$ and $r = 4$. For this scenario, the lower bound from (26) on the communication load indicates that $L^*(r) \geq 913$. Next, we consider a scenario with $Q = 5$ and $\mathbf{r} = (5, 4, 4, 4, 3)$. For this scenario, we have from (24) the following upper bound on the communication load $\tilde{L}^*(r) \leq 865$. Lastly, from (14), we have that the average storage load is equal ($r = \frac{1}{5}\sum_{j=1}^{5} r_j = 4$) for both scenarios. Therefore, the topology-aware coding strategy ($Q = 5$) is strictly better than the topology-independent coding strategy ($Q = 1$). ■

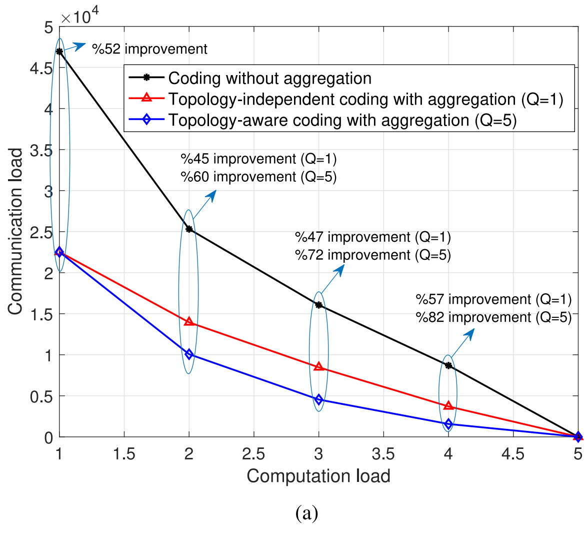
V. Performance on Real-World Networks

This section demonstrates the performance of our coded graph processing framework on various real-world systems.

Google Web Graph Implementation: We first investigate the performance of the topology-aware coded computing framework (TACC) on a real-world scale-free network, and demonstrate the effectiveness of topology-aware coded computing over the topology-independent frameworks. An important real-world example of a scale-free network is a *web graph*, which represents the connectivity structure between webpages in the World Wide Web. In this graph, nodes represent webpages and edges denote the hyperlinks between webpages. Web graphs are utilized for various graph computations such as the PageRank algorithm for determining the ranking of webpages in web search. In our evaluations, we consider the web graph released by Google as part of a programming contest in 2002, which is available in [24]. The graph has 875,713 nodes and 5,105,039 directed edges. Average node degree is 5.57, hence the graph is relatively sparse.

For the initial partitioning of the graph in order to create the sets $\mathcal{R} = (\mathcal{R}_1, \ldots, \mathcal{R}_K)$, we utilize the METIS graph partitioning tool from [15], [42]. The motivation behind using a structured graph partitioning tool instead of random partitioning is to reduce the initial cut size, i.e., the number of edges whose endpoints are stored at different workers, and obtain a smaller boundary than random partitioning. Next, we consider the three scenarios from Section IV-3 corresponding to: coding without aggregation, topology-independent coding with aggregation ($Q = 1$), and topology-aware coding with aggregation ($Q = 5$). For the last scenario, the storage load is defined as in (23).

We provide the comparison of the communication load for the three schemes in Fig. 7. As observed from Fig. 7a, aggregation can lead to an improvement of 57% over the coding strategy without aggregation, whereas topology-aware coding with aggregation can lead up to an 82% improvement. Hence, aggregation can be very useful in reducing the communication cost for sparse graphs with irregular degree structures, which complements our results from Section IV-1. In Fig. 7b, we



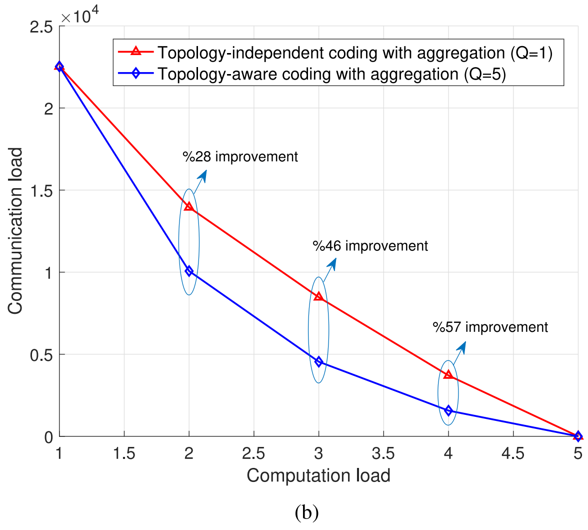


Fig. 7. Demonstrating the gain of the topology-aware coded computing (TACC) algorithm over (i) topology-independent coding with aggregation and (ii) coding without aggregation, in reducing the communication load for running PageRank on the Google web graph dataset consisting of 875,713 nodes and 5,105,039 edges.

compare the communication load for topology-independent and topology-aware coding schemes, both with aggregation, and find that topology-aware subgraph allocation can lead up to 57% improvement over the topology-independent setup.

Amazon EC2 Cloud Implementation: We next consider a distributed implementation of the PageRank algorithm from Section 1 on the Amazon EC2 cloud computing platform. A distributed system with 12 workers is constructed using the StarCluster open source cluster-computing toolkit [47]. Communication between the workers is implemented by using the `MPI4Py` message passing interface on `Python` [48]. All workers are initialized using the instance type m4.large and with a maximum bandwidth of 100 Mbps.

An important feature of cloud environments is that network conditions can become highly variable, on which the user has no control. In fact, cloud network architectures allow jobs from many users to share the same resources, such as compute machines and network traffic. One implication of this on our coding framework is that we have no control over the scheduling of the coded multicast instances, as well as the overhead that is brought by creating them. This could have a significant impact on the shuffle time of our coded computing scheme, since our coded aggregation scheme requires $\binom{K}{r+1}$ multicast groups to be created for a storage load of r. As a result, even though the total number of messages that is transmitted between the workers is reduced when r is increased, the cost associated with creating more multicast instances may in turn negate the gain obtained from reduced communication load. To address this, we adopt in our cloud experiments the task allocation scheme proposed in [49], which aims at reducing the number of communication instances that needs to be created in the coded multicasting setup. In particular, this setup requires $q^{r-1}(q-1)$ multicast groups instead of $\binom{K}{r+1}$, where $q = \frac{K}{r}$.

TABLE I
BREAKDOWN OF THE TOTAL EXECUTION TIME FOR PAGERANK IMPLEMENTATION ON THE AMAZON EC2 CLOUD

	Uncoded	[Prakash'18]	TACC
Map time (s)	5	14	17
Shuffle time (s)	144	106	46
Reduce time (s)	3	20	19
Total time (s)	152	140	82

We then implement our coded aggregation framework on an Erdös-Rényi random graph with $N =$ 207,360 nodes and edge probability $p = 0.0001$, with an average node degree of 20.7, and computation load $r = 4$. Since the degree structure of an Erdös-Rényi graph is uniform across all nodes, it has a regular graph topology, hence we consider a single group by setting $Q = 1$ in our algorithm. Table I demonstrates the overall execution time of our algorithm (TACC) as well as the time spent in individual map, shuffle, reduce phases, compared to uncoded transmission [7] and coded transmission without aggregation ([Prakash'18]) from [18]. It can be observed from Table I that TACC decreases the overall execution time by 46% compared to uncoded transmission and 41% compared to coded transmission without aggregation.

Finally, we note that iterative algorithms have an additional cost of communicating back the updated values of processed nodes, i.e., $\mathcal{R}$, back to the workers who are responsible for mapping them. In our framework, this is done via multicasting the nodes that are reduced by each worker to the corresponding r workers who need the updated values for the next round. In Table I, this additional time cost is included in the reduce time. As also observed in our experiments, this cost is typically smaller than the communication cost during the shuffle phase, as long as r is relatively small compared to the total number of workers K and the average node degree.

VI. CONCLUSION

We have proposed a topology-aware coding framework (TACC) to address the communication bottleneck in distributed graph processing. TACC leverages the graph topology for storage allocation, in that higher degree nodes are stored at a larger fraction of workers. It then creates redundancy between the parts of the graph allocated to different workers and aggregates the intermediate computations to enable coded multicasting opportunities. We analyze the performance gain of TACC both theoretically and experimentally. Our results demonstrate that TACC can achieve a significant speedup over the existing graph processing frameworks.

This work is an initial step towards taking into account the graph topology in coded computing, but the optimal solution for the general problem is still open as the solution space is highly complex. For instance, while creating redundant storage for nodes assigned to one worker, say Worker 1, we partition its assigned nodes into batches and make each batch redundant by assigning it also to other workers. Then, the assignment of a specific batch can be made by taking into account how well connected nodes in the batch are to the nodes already assigned to workers to which this batch will be assigned. For instance, if for a given batch we have a choice of assigning it to Worker 2 or Worker 3, then, we would assign the batch to Worker 2 if the nodes already stored at Worker 2 are more tightly connected to the nodes in the batch, than the nodes stored at Worker 3, and vice versa. This could in turn reduce the communication load further. We plan to investigate this direction in our future work.

Appendix A
Proof of Theorem 3

Consider the expected number of messages that needs to be sent to worker k from the other workers. Since Worker k knows the nodes in subgraph $\mathcal{M}_k$ and processes the outputs for the nodes in $\mathcal{R}_k$, from (20), as $N \to \infty$, we expect $\frac{|\mathcal{R}_k|}{W} = \frac{N}{KW}$ nodes in $\mathcal{R}_k$ to admit a weight λ_i for $i \in [K]$. Next, we define the following random variable,

$$A_v = \begin{cases} 1 & \text{if } \exists (u,v) \in \mathcal{E}, u \in \mathcal{V}\backslash\mathcal{M}_k \\ 0 & \text{otherwise} \end{cases} \tag{40}$$

for each node $v \in \mathcal{R}_k$. That is, $A_v = 1$ if there exists an edge from a node not mapped by Worker k to node $v \in \mathcal{R}_k$. Hence, Worker k needs to receive at least $\sum_{v\in R_k} A_v$ distinct messages from the remaining workers. Otherwise, Worker k will not be able to distinguish the messages aimed for some of the nodes in $v \in \mathcal{R}_k$ and will not be able to process them.

The total number of messages sent from all workers is at least as large as the number of messages that needs to be sent from $[K]\backslash\{k\}$ to Worker k. Therefore,

$$L^*(r)$$
$$\geq E[\sum_{v\in\mathcal{R}_k} A_v] \tag{41}$$
$$= \sum_{v\in\mathcal{R}_k}\sum_{i=1}^{W} \frac{1}{W} E[A_v | w_v = \lambda_i] \tag{42}$$
$$= \sum_{v\in\mathcal{R}_k}\sum_{i=1}^{W} \frac{1}{W} P(A_v = 1 | w_v = \lambda_i) \tag{43}$$
$$= \sum_{v\in\mathcal{R}_k}\sum_{i=1}^{W} \frac{1}{W} (1 - P(A_v = 0 | w_v = \lambda_i)) \tag{44}$$
$$= \sum_{v\in\mathcal{R}_k}\sum_{i=1}^{W} \frac{1}{W} \left(1 - \prod_{u\in\mathcal{V}\backslash\mathcal{M}_k} P((u,v)\notin\mathcal{E} | w_v = \lambda_i)\right) \tag{45}$$
$$\geq \frac{N}{KW}\sum_{i=1}^{W}\left(1 - \prod_{q=1}^{Q}\prod_{j=1+\frac{(q-1)W}{Q}}^{\frac{qW}{Q}} \times \left(1 - \frac{\lambda_i\lambda_j}{\frac{N}{W}\sum_{t=1}^{W}\lambda_t}\right)^{\frac{N}{W}(1-\frac{r_q}{K})}\right) \tag{46}$$

where (46) is from the fact that, as $N \to \infty$, each worker stores $\frac{Nr_q}{WK}$ nodes with weight $\lambda_{1+\frac{(q-1)W}{Q}}, \ldots, \lambda_{\frac{qW}{Q}}$, whereas the total number of nodes with weight λ_j, $j \in [W]$, is $\frac{N}{W}$.

Appendix B

For the first scenario, we obtain from (22) that,

$$\mu_1 = \frac{N}{KW}\sum_{i=1}^{W}\left(1 - \prod_{j=1}^{W}\left(1 - \frac{\lambda_i\lambda_j}{\frac{N}{W}\sum_{t=1}^{W}\lambda_t}\right)^{\frac{N}{W}(1-\frac{r}{K})}\right)$$
$$= \frac{N}{K} - \frac{N}{KW}\sum_{i=1}^{W}\frac{\prod_{j=1}^{W}\left(1 - \frac{\lambda_i\lambda_j}{\frac{N}{W}\sum_{t=1}^{W}\lambda_t}\right)^{\frac{N}{W}}}{\prod_{j=1}^{W}\left(1 - \frac{\lambda_i\lambda_j}{\frac{N}{W}\sum_{t=1}^{W}\lambda_t}\right)^{\frac{Nr}{WK}}} \tag{47}$$

by setting $Q = W$ in (22). Similarly, for the second scenario, we use (22) to find that,

$$\mu_2 = \frac{N}{K} - \frac{N}{KW}\sum_{i=1}^{W}\frac{\prod_{j=1}^{W}\left(1 - \frac{\lambda_i\lambda_j}{\frac{N}{W}\sum_{t=1}^{W}\lambda_t}\right)^{\frac{N}{W}}}{\prod_{j=1}^{W}\left(1 - \frac{\lambda_i\lambda_j}{\frac{N}{W}\sum_{t=1}^{W}\lambda_t}\right)^{\frac{Nr_j}{WK}}}. \tag{48}$$

From the denominator of (47), we then find that,

$$\prod_{j=1}^{W}\left(1 - \frac{\lambda_i\lambda_j}{\frac{N}{W}\sum_{t=1}^{W}\lambda_t}\right)^{\frac{Nr}{WK}}$$
$$= \prod_{j=1}^{W}\left(1 - \frac{\lambda_i\lambda_j}{\frac{N}{W}\sum_{t=1}^{W}\lambda_t}\right)^{\frac{N}{WK}\left(\frac{1}{W}\sum_{k=1}^{W}r_k\right)}$$
$$= \left(1 - \frac{\lambda_i\lambda_1}{\frac{N}{W}\sum_{t=1}^{W}\lambda_t}\right)^{\frac{N}{WK}\cdot\frac{r_1}{W}} \cdots \left(1 - \frac{\lambda_i\lambda_1}{\frac{N}{W}\sum_{t=1}^{W}\lambda_t}\right)^{\frac{N}{WK}\cdot\frac{r_W}{W}}$$
$$\cdots\left(1 - \frac{\lambda_i\lambda_W}{\frac{N}{W}\sum_{t=1}^{W}\lambda_t}\right)^{\frac{N}{WK}\cdot\frac{r_1}{W}} \cdots \left(1 - \frac{\lambda_i\lambda_W}{\frac{N}{W}\sum_{t=1}^{W}\lambda_t}\right)^{\frac{N}{WK}\cdot\frac{r_W}{W}}$$
$$\geq \left(1 - \frac{\lambda_i\lambda_1}{\frac{N}{W}\sum_{t=1}^{W}\lambda_t}\right)^{\frac{N}{WK}\cdot\frac{r_1}{W}} \cdots \left(1 - \frac{\lambda_i\lambda_W}{\frac{N}{W}\sum_{t=1}^{W}\lambda_t}\right)^{\frac{N}{WK}\cdot\frac{r_W}{W}}$$

$$\ldots\left(1-\frac{\lambda_i\lambda_1}{\frac{N}{W}\sum_{t=1}^{W}\lambda_t}\right)^{\frac{N}{WK}\cdot\frac{r_1}{W}}\ldots\left(1-\frac{\lambda_i\lambda_W}{\frac{N}{W}\sum_{t=1}^{W}\lambda_t}\right)^{\frac{N}{WK}\cdot\frac{r_W}{W}} \tag{49}$$

$$=\prod_{j=1}^{W}\left(1-\frac{\lambda_i\lambda_j}{\frac{N}{W}\sum_{t=1}^{W}\lambda_t}\right)^{\frac{Nr_j}{WK}} \tag{50}$$

Equation (49) follows from the fact that,

$$\left(1-\frac{\lambda_i\lambda_j}{\frac{N}{W}\sum_{t=1}^{W}\lambda_t}\right)^{\frac{N}{WK}\cdot\frac{r_k}{W}}\left(1-\frac{\lambda_i\lambda_k}{\frac{N}{W}\sum_{t=1}^{W}\lambda_t}\right)^{\frac{N}{WK}\cdot\frac{r_j}{W}}$$
$$\geq\left(1-\frac{\lambda_i\lambda_j}{\frac{N}{W}\sum_{t=1}^{W}\lambda_t}\right)^{\frac{N}{WK}\cdot\frac{r_j}{W}}\left(1-\frac{\lambda_i\lambda_k}{\frac{N}{W}\sum_{t=1}^{W}\lambda_t}\right)^{\frac{N}{WK}\cdot\frac{r_k}{W}} \tag{51}$$

for any $k,j\in[W]$, $k\neq j$. To observe this, without loss of generality, assume that $k\leq j$ and let $\beta_j\triangleq\frac{\lambda_i\lambda_j}{\frac{N}{W}\sum_{t=1}^{W}\lambda_t}$ and $\beta_k\triangleq\frac{\lambda_i\lambda_k}{\frac{N}{W}\sum_{t=1}^{W}\lambda_t}$. By definition, $\beta_k\geq\beta_j$ and $r_k\geq r_j$, hence,

$$(1-\beta_j)^{\frac{N}{WK}\cdot\frac{r_k}{W}}(1-\beta_k)^{\frac{N}{WK}\cdot\frac{r_j}{W}}$$
$$=(1-\beta_j)^{\frac{N}{WK}\cdot\frac{r_j}{W}}(1-\beta_k)^{\frac{N}{WK}\cdot\frac{r_k}{W}}\left(\frac{1-\beta_j}{1-\beta_k}\right)^{\frac{N}{WK}\cdot\frac{(r_k-r_j)}{W}} \tag{52}$$
$$\geq(1-\beta_j)^{\frac{N}{WK}\cdot\frac{r_j}{W}}(1-\beta_k)^{\frac{N}{WK}\cdot\frac{r_k}{W}}. \tag{53}$$

Equation (50) is equal to the denominator of (48). Since the remaining terms are equal in both (47) and (48), it then follows from (50) that $\mu_1\geq\mu_2$.

Appendix C
Proof of Theorem 3

We prove this result by analyzing the expected communication load of the achievable scheme from Section III for generalized random graphs. We first note that coded multicasting is performed within each group $q\in[Q]$ independently. Within group q, Worker $k\in[K]$ is a member of $\binom{K-1}{r_q}$ subsets of size r_q+1, and multicasts a coded message for each one of them. Due to the symmetry in subgraph allocation, coding scheme, and the weight assignment for the random graph, the expected number of messages sent from each worker is equal. Then, one can find the expected communication load of the distributed system to be,

$$\sum_{q=1}^{Q}K\binom{K-1}{r_q}E[L_q], \tag{54}$$

where $E[L_q]$ is the expected communication load of each worker for group q. In the following, we bound $E[L_q]$ as $N\to\infty$. Consider a group $q\in[Q]$. Let $\mathcal{S}\subseteq[K]$ be a subset of workers of size r_q+1. Without loss of generality, let these r_q+1 workers be $\mathcal{S}=\{1,2,\ldots,r_q,r_q+1\}$. Consider the coded message formed by Worker r_q+1. Define a binary random variable Y_{kj},

$$Y_{kj}=\begin{cases}1 & \text{if } \exists(i,j)\in\mathcal{E}\text{ where } i\in\mathcal{M}^q_{\mathcal{S}\backslash\{k\}}\\ 0 & \text{otherwise}\end{cases} \tag{55}$$

for $j\in\mathcal{R}_k$ and $k\in[r_q]$, where $\mathcal{M}^q_{\mathcal{S}\backslash\{k\}}=\bigcap_{k'\in\mathcal{S}\backslash\{k\}}\mathcal{M}^q_{k'}$. Then, the number of aggregated messages to be sent by Worker r_q+1 and needed by Worker $k\in[r_q]$ is,

$$Y_k=\frac{1}{r_q}\sum_{j\in\mathcal{R}_k}Y_{kj}. \tag{56}$$

From the subgraph allocation phase in Algorithm 1, we observe that $\mathcal{R}_k\cap\mathcal{M}^q_{\mathcal{S}\backslash\{k\}}=\emptyset$, hence the nodes reduced by Worker k (nodes for which Worker k computes the output values for) are disjoint from the nodes mapped exclusively by the subset of workers $\mathcal{S}\backslash\{k\}$. Then, given the node weights, the edges between the map nodes in $\mathcal{M}^q_{\mathcal{S}\backslash\{k\}}$ and reduce nodes in $\mathcal{R}_k$ are independent. Thus, random variables Y_{kj} are independent for all $j\in\mathcal{R}_k$ when conditioned on node weights $W_1,\ldots,W_N$. From (56), the total number of messages multicasted from Worker r_q+1 to workers in $[r_q]$ is,

$$L_q=\max_{k=1,\ldots,r_q}Y_k. \tag{57}$$

Our asymptotic analysis utilizes the notion of strong typicality from information theory.

Definition 5 (Strong Typicality, [50]): Consider the sequence of random variables $(W_1,\ldots,W_N)$ drawn i.i.d from the distribution in (20). Furthermore, let $\mathbf{w}=(w_1,\ldots,w_N)$ denote a realization of $(W_1,\ldots,W_N)$. Define $\mathcal{T}^N_\epsilon$ as the set of sequences $\mathbf{w}$ that satisfy,

$$\sum_{t=1}^{W}\left|\frac{|\{i:w_i=\lambda_t\}|}{N}-p(\lambda_t)\right|<\epsilon \tag{58}$$

where $|\{i:w_i=\lambda_t\}|$ is the number of occurrences of λ_t in $\mathbf{w}$. Then, $\mathcal{T}^N_\epsilon$ is called a strongly typical set, and sequences $\mathbf{w}$ that satisfy (58) are called strongly ϵ-typical sequences. From the Law of Large Numbers and strong AEP (Asymptotic Equipartition Property), it follows that,

$$P\{(W_1,\ldots,W_N)\in\mathcal{T}^N_\epsilon\}>1-\epsilon \tag{59}$$

as $N\to\infty$ and $\epsilon\to0$.

Then, for a given $s_q>0$, we can bound the expected communication load of Worker r_q+1 as:

$$e^{s_q r_q E[L_q]} \tag{60}$$
$$\leq E\left[e^{s_q r_q L_q}\right] \tag{61}$$
$$=E\left[e^{s_q r_q \max_{k=1,\ldots,r_q}Y_k}\right] \tag{62}$$
$$\leq E\left[\sum_{k=1}^{r_q}e^{s_q r_q Y_k}\right] \tag{63}$$
$$=\sum_{k=1}^{r_q}E\left[e^{s_q\sum_{j\in\mathcal{R}_k}Y_{kj}}\right] \tag{64}$$
$$=\sum_{k=1}^{r_q}E\left[\prod_{j\in\mathcal{R}_k}e^{s_q Y_{kj}}\right] \tag{65}$$

$$= \sum_{k=1}^{r_q} E\left[E\left[\prod_{j\in\mathcal{R}_k} e^{s_q Y_{kj}} | W_1, \ldots, W_N\right]\right] \tag{66}$$

$$= \sum_{k=1}^{r_q} E\left[\prod_{j\in\mathcal{R}_k} E[e^{s_q Y_{kj}} | W_1, \ldots, W_N]\right] \tag{67}$$

$$= \sum_{k=1}^{r_q} E\Bigg[\prod_{j\in\mathcal{R}_k} (P(Y_{kj}=0|W_1,\ldots,W_N) + e^{s_q} P(Y_{kj}=1|W_1,\ldots,W_N))\Bigg] \tag{68}$$

$$= \sum_{k=1}^{r_q} E\left[\prod_{j\in\mathcal{R}_k}\left(\prod_{i\in\mathcal{M}^q_{\mathcal{S}\backslash\{k\}}}\left(1-\frac{W_iW_j}{\sum_{t=1}^N W_t}\right) + e^{s_q}\left(1-\prod_{i\in\mathcal{M}^q_{\mathcal{S}\backslash\{k\}}}\left(1-\frac{W_iW_j}{\sum_{t=1}^N W_t}\right)\right)\right)\right] \tag{69}$$

$$= \sum_{k=1}^{r_q}\sum_{\mathbf{w}\in\mathcal{T}^N_\epsilon} p(\mathbf{w}) \times E\left[\prod_{j\in\mathcal{R}_k}\left(\prod_{i\in\mathcal{M}^q_{\mathcal{S}\backslash\{k\}}}\left(1-\frac{w_iw_j}{\sum_{t=1}^N w_t}\right) + e^{s_q}\left(1-\prod_{i\in\mathcal{M}^q_{\mathcal{S}\backslash\{k\}}}\left(1-\frac{w_iw_j}{\sum_{t=1}^N w_t}\right)\right)\right)\right]$$
$$+ \sum_{k=1}^{r_q}\sum_{\mathbf{w}\notin\mathcal{T}^N_\epsilon} p(\mathbf{w}) \times E\left[\prod_{j\in\mathcal{R}_k}\left(\prod_{i\in\mathcal{M}^q_{\mathcal{S}\backslash\{k\}}}\left(1-\frac{w_iw_j}{\sum_{t=1}^N w_t}\right) + e^{s_q}\left(1-\prod_{i\in\mathcal{M}^q_{\mathcal{S}\backslash\{k\}}}\left(1-\frac{w_iw_j}{\sum_{t=1}^N w_t}\right)\right)\right)\right] \tag{70}$$

$$< \sum_{k=1}^{r_q}\sum_{\mathbf{w}\in\mathcal{T}^N_\epsilon} p(\mathbf{w}) \times E\left[\prod_{j\in\mathcal{R}_k}\left(\prod_{i\in\mathcal{M}^q_{\mathcal{S}\backslash\{k\}}}\left(1-\frac{w_iw_j}{\sum_{t=1}^N w_t}\right) + e^{s_q}\left(1-\prod_{i\in\mathcal{M}^q_{\mathcal{S}\backslash\{k\}}}\left(1-\frac{w_iw_j}{\sum_{t=1}^N w_t}\right)\right)\right)\right]$$
$$+ \epsilon\, r_q \max_{\mathbf{w}\notin\mathcal{T}^N_\epsilon} E\left[\prod_{j\in\mathcal{R}_k}\left(\prod_{i\in\mathcal{M}^q_{\mathcal{S}\backslash\{k\}}}\left(1-\frac{w_iw_j}{\sum_{t=1}^N w_t}\right) + e^{s_q}\left(1-\prod_{i\in\mathcal{M}^q_{\mathcal{S}\backslash\{k\}}}\left(1-\frac{w_iw_j}{\sum_{t=1}^N w_t}\right)\right)\right)\right] \tag{71}$$

$$= \sum_{k=1}^{r_q}\sum_{\mathbf{w}\in\mathcal{T}^N_\epsilon} p(\mathbf{w}) \times E\left[\prod_{j\in\mathcal{R}_k}\left(\prod_{i\in\mathcal{M}^q_{\mathcal{S}\backslash\{k\}}}\left(1-\frac{w_iw_j}{\sum_{t=1}^N w_t}\right) + e^{s_q}\left(1-\prod_{i\in\mathcal{M}^q_{\mathcal{S}\backslash\{k\}}}\left(1-\frac{w_iw_j}{\sum_{t=1}^N w_t}\right)\right)\right)\right] \tag{72}$$

as $N \to \infty$ and $\epsilon \to 0$. Equation (61) follows from Jensen's inequality, (66) is from the law of iterated expectation; the outer expectation defined over the node weights and random subgraph partitioning whereas the inner expectation is defined over the edge probabilities. Equation (67) holds since random variables Y_{kj} are independent when conditioned on the node weights, equation (69) follows from (55) and Definition 3. Equation (70) follows from the law of iterated expectation, in which the inner expectation is defined over the random subgraph partitions, where each worker is assigned to $\frac{N}{K}$ nodes uniformly at random. Next, we define,

$$n^{\mathcal{R}_k}_{\mathbf{w}}(m) = |\{j\in\mathcal{R}_k : w_j = \lambda_m\}|,\ m\in[W], \tag{73}$$

$$n^{\mathcal{M}^q_{\mathcal{S}\backslash\{k\}}}_{\mathbf{w}}(m) = |\{j\in\mathcal{M}^q_{\mathcal{S}\backslash\{k\}} : w_j = \lambda_m\}|,\ m\in[W], \tag{74}$$

as the number of occurrences of weight λ_m in $\mathcal{R}_k$, and in $\mathcal{M}^q_{\mathcal{S}\backslash\{k\}}$, respectively. It then follows from (72)–(74) that,

$$\sum_{k=1}^{r_q}\sum_{\mathbf{w}\in\mathcal{T}^N_\epsilon} p(\mathbf{w}) \times E\left[\prod_{j\in\mathcal{R}_k}\left(\prod_{i\in\mathcal{M}^q_{\mathcal{S}\backslash\{k\}}}\left(1-\frac{w_iw_j}{\sum_{t=1}^N w_t}\right) + e^{s_q}\left(1-\prod_{i\in\mathcal{M}^q_{\mathcal{S}\backslash\{k\}}}\left(1-\frac{w_iw_j}{\sum_{t=1}^N w_t}\right)\right)\right)\right]$$
$$= \sum_{k=1}^{r_q}\sum_{\mathbf{w}\in\mathcal{T}^N_\epsilon} p(\mathbf{w}) \times E\left[\prod_{m=1}^{W}\left(\prod_{l=1}^{W}\left(1-\frac{\lambda_m\lambda_l}{\frac{N}{W}\sum_{t=1}^W\lambda_t}\right)^{n^{\mathcal{M}^q_{\mathcal{S}\backslash\{k\}}}_{\mathbf{w}}(l)} + e^{s_q}\left(1-\prod_{l=1}^{W}\left(1-\frac{\lambda_m\lambda_l}{\frac{N}{W}\sum_{t=1}^W\lambda_t}\right)^{n^{\mathcal{M}^q_{\mathcal{S}\backslash\{k\}}}_{\mathbf{w}}(l)}\right)\right)^{n^{\mathcal{R}_k}_{\mathbf{w}}(m)}\right] \tag{75}$$

$$= \sum_{k=1}^{r_q}\sum_{\mathbf{w}\in\mathcal{T}^N_\epsilon} p(\mathbf{w}) \prod_{m=1}^{W}\left(\prod_{l=1+\frac{(q-1)W}{Q}}^{\frac{qW}{Q}}\left(1-\frac{\lambda_m\lambda_l}{\frac{N}{W}\sum_{t=1}^W\lambda_t}\right)^{\frac{N}{W\binom{K}{r_q}}} + e^{s_q}\left(1-\prod_{l=1+\frac{(q-1)W}{Q}}^{\frac{qW}{Q}}\left(1-\frac{\lambda_m\lambda_l}{\frac{N}{W}\sum_{t=1}^W\lambda_t}\right)^{\frac{N}{W\binom{K}{r_q}}}\right)\right)^{\frac{N}{WK}} \tag{76}$$

$$= \sum_{k=1}^{r_q}\prod_{m=1}^{W}\left(\prod_{l=1+\frac{(q-1)W}{Q}}^{\frac{qW}{Q}}\left(1-\frac{\lambda_m\lambda_l}{\frac{N}{W}\sum_{t=1}^W\lambda_t}\right)^{\frac{N}{W\binom{K}{r_q}}} + e^{s_q}\left(1-\prod_{l=1+\frac{(q-1)W}{Q}}^{\frac{qW}{Q}}\left(1-\frac{\lambda_m\lambda_l}{\frac{N}{W}\sum_{t=1}^W\lambda_t}\right)^{\frac{N}{W\binom{K}{r_q}}}\right)\right)^{\frac{N}{WK}} \tag{77}$$

$$= r_q\prod_{m=1}^{W}\left(\prod_{l=1+\frac{(q-1)W}{Q}}^{\frac{qW}{Q}}\left(1-\frac{\lambda_m\lambda_l}{\frac{N}{W}\sum_{t=1}^W\lambda_t}\right)^{\frac{N}{W\binom{K}{r_q}}} + e^{s_q}\left(1-\prod_{l=1+\frac{(q-1)W}{Q}}^{\frac{qW}{Q}}\left(1-\frac{\lambda_m\lambda_l}{\frac{N}{W}\sum_{t=1}^W\lambda_t}\right)^{\frac{N}{W\binom{K}{r_q}}}\right)\right)^{\frac{N}{WK}} \tag{78}$$

where (76) is due to the random weight assignment for the generalized random graph described in (20) and the typicality argument from (59), as $N \to \infty$, out of N nodes in the graph, a fraction of $\frac{1}{W}$ nodes have weight λ_m, $m \in [W]$. During the subgraph allocation phase, Worker k is assigned to $|\mathcal{R}_k| = \frac{N}{K}$ nodes. From the random partitioning setup described in Section III, $\frac{N}{WK}$ nodes within $\mathcal{R}_k$ will have weight (and average degree) λ_m. The nodes in $\mathcal{R}_k$ are sorted according to their degrees and divided into Q groups, in which group q has $\frac{N}{QK}$ nodes. As $\frac{W}{Q} \in \mathbb{Z}$, group q consists of the nodes with degree λ_l for $l = 1 + \frac{(q-1)W}{Q}, \ldots, \frac{qW}{Q}$.

For each worker, the $\frac{N}{QK}$ nodes in group q are divided randomly into $\binom{K-1}{r_q-1}$ parts where each part is stored at a distinct subset of $r_q - 1$ other workers. Accordingly, each part has $\frac{1}{\binom{K-1}{r_q-1}} \frac{N}{\frac{W}{Q}QK} = \frac{1}{\binom{K-1}{r_q-1}} \frac{N}{WK}$ nodes with degree λ_l for $l = 1 + \frac{(q-1)W}{Q}, \ldots, \frac{qW}{Q}$. As a result, each subset of r_q workers will exclusively store $\frac{r_q}{\binom{K-1}{r_q-1}} \frac{N}{WK} = \frac{N}{W\binom{K}{r_q}}$ nodes with degree λ_l, $l = 1 + \frac{(q-1)W}{Q}, \ldots, \frac{qW}{Q}$. Lastly, (76) holds since $\sum_{\mathbf{w} \in \mathcal{T}_\epsilon^N} p(\mathbf{w}) \to 1$ as $N, \epsilon \to \infty$ from (59). By combining (60) with (78) and taking the logarithm of both sides,

$$E[L_q] \leq \frac{1}{s_q r_q} \left(\ln(r_q) + \sum_{m=1}^{W} \frac{N}{WK} \right.$$
$$\ln \left(\prod_{l=1+\frac{(q-1)W}{Q}}^{\frac{qW}{Q}} \left(1 - \frac{\lambda_m \lambda_l}{\frac{N}{W} \sum_{t=1}^{W} \lambda_t} \right)^{\frac{N}{W\binom{K}{r_q}}} \right.$$
$$\left. \left. + e^{s_q} \left(1 - \prod_{l=1+\frac{(q-1)W}{Q}}^{\frac{qW}{Q}} \left(1 - \frac{\lambda_m \lambda_l}{\frac{N}{W} \sum_{t=1}^{W} \lambda_t} \right)^{\frac{N}{W\binom{K}{r_q}}} \right) \right) \right),$$

which, along with (54), leads to (24).

References

[1] B. Guler, A. S. Avestimehr, and A. Ortega, "A topology-aware coding framework for distributed graph processing," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process.*, 2019, pp. 8182–8186.

[2] S. Brin and L. Page, "The anatomy of a large-scale hypertextual web search engine," *Comp. Net. Integr. Services Digit. Netw. Syst.*, vol. 30, pp. 107–117, 1998.

[3] A. Ortega, P. Frossard, J. Kovačević, J. M. Moura, and P. Vandergheynst, "Graph signal processing: Overview, challenges, and applications," *Proc. IEEE*, vol. 106, no. 5, pp. 808–828, May 2018.

[4] D. I. Shuman, S. K. Narang, P. Frossard, A. Ortega, and P. Vandergheynst, "The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains," *IEEE Signal Proc. Mag.*, vol. 30, no. 3, pp. 83–98, May 2013.

[5] A. Sandryhaila and J. M. Moura, "Discrete signal processing on graphs," *IEEE Trans. Signal Process.*, vol. 61, no. 7, pp. 1644–1656, Apr. 2013.

[6] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," in *Proc. Adv. Neural Inf. Process. Syst.*, 2016, pp. 3844–3852.

[7] G. Malewicz *et al.*, "Pregel: A system for large-scale graph processing," in *Proc. ACM Int. Conf. Manage. Data*, 2010, pp. 135–146.

[8] 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 Endownment*, vol. 5, no. 8, pp. 716–727, 2012.

[9] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[10] A. Lumsdaine, D. Gregor, B. Hendrickson, and J. Berry, "Challenges in parallel graph processing," *Parallel Proc. Lett.*, vol. 17, pp. 5–20, 2007.

[11] R. Chen, X. Ding, P. Wang, H. Chen, B. Zang, and H. Guan, "Computation and communication efficient graph processing with distributed immutable view," in *Proc. Int. Symp. High-Perform. Parallel Distrib. Comput.*, 2014, pp. 215–226.

[12] B. Hendrickson and T. G. Kolda, "Graph partitioning models for parallel computing," *Parallel Comput.*, vol. 26, no. 12, pp. 1519–1534, 2000.

[13] M. Fiedler, "A property of eigenvectors of nonnegative symmetric matrices and its application to graph theory," *Czechoslovak Math. J.*, vol. 25, no. 4, pp. 619–633, 1975.

[14] H. D. Simon, "Partitioning of unstructured problems for parallel processing," *Comput. Syst. Eng.*, vol. 2, no. 2, pp. 135–148, 1991.

[15] G. Karypis and V. Kumar, "A fast and high quality multilevel scheme for partitioning irregular graphs," *SIAM J. Scientific Comput.*, vol. 20, no. 1, pp. 359–392, 1998.

[16] A. Buluç, H. Meyerhenke, I. Safro, P. Sanders, and C. Schulz, "Recent advances in graph partitioning," in *Proc. Algorithm Eng.*, 2016, pp. 117–158.

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

[18] S. Prakash, A. Reisizadeh, R. Pedarsani, and S. Avestimehr, "Coded computing for distributed graph analytics," in *Proc. IEEE Int. Symp. Inf. Theory*, 2018, pp. 1221–1225.

[19] S. Li, M. A. Maddah-Ali, and A. S. Avestimehr, "Compressed coded distributed computing," in *Proc. IEEE Int. Symp. Inf. Theory*, 2018, pp. 2032–2036.

[20] A. Clauset, C. R. Shalizi, and M. E. Newman, "Power-law distributions in empirical data," *SIAM Rev.*, vol. 51, no. 4, pp. 661–703, 2009.

[21] F. Chung and L. Lu, "The average distances in random graphs with given expected degrees," *Nat. Academy Sci.*, vol. 99, no. 25, pp. 15 879–15 882, 2002.

[22] F. Chung and L. Lu, "Connected components in random graphs with given expected degree sequences," *Ann. Combinatorics*, vol. 6, pp. 125–145, 2002.

[23] T. Britton, M. Deijfen, and A. Martin-Löf, "Generating simple random graphs with prescribed degree distribution," *J. Statist. Phys.*, vol. 124, no. 6, pp. 1377–1397, 2006.

[24] J. Leskovec and A. Krevl, "SNAP Datasets: Stanford large network dataset collection," 2014. [Online]. Available: http://snap.stanford.edu/data

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

[26] S. Dutta, V. Cadambe, and P. Grover, "Short-dot: Computing large linear transforms distributedly using coded short dot products," in *Proc. Adv. Neural Inf. Process. Syst.*, 2016, pp. 2100–2108.

[27] Q. Yu, M. Maddah-Ali, and S. Avestimehr, "Polynomial codes: An optimal design for high-dimensional coded matrix multiplication," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 4406–4416.

[28] Q. Yu, S. Li, N. Raviv, S. M. M. Kalan, M. Soltanolkotabi, and A. S. Avestimehr, "Lagrange coded computing: Optimal design for resiliency, security and privacy," in *Proc. Int. Conf. Artif. Intell. Statist.*, Naha, Okinawa, Japan, vol. 89, 2019, pp. 1215–1225.

[29] C. Karakus, Y. Sun, S. Diggavi, and W. Yin, "Straggler mitigation in distributed optimization through data encoding," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 5434–5442.

[30] C. Karakus, Y. Sun, and S. Diggavi, "Encoded distributed optimization," in *Proc. IEEE Int. Symp. Inf. Theory*, 2017, pp. 2890–2894.

[31] C. Karakus, Y. Sun, S. N. Diggavi, and W. Yin, "Redundancy techniques for straggler mitigation in distributed optimization and learning." *J. Mach. Learn. Res.*, vol. 20, no. 72, pp. 1–47, 2019.

[32] R. Tandon, Q. Lei, A. G. Dimakis, and N. Karampatziakis, "Gradient coding: Avoiding stragglers in distributed learning," in *Proc. Int. Conf. Mach. Learn.*, 2017, pp. 3368–3376.

[33] Y. Yang, P. Grover, and S. Kar, "Coded distributed computing for inverse problems," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 709–719.

[34] A. Mallick and G. Joshi, "Rateless codes for distributed computations with sparse compressed matrices," in *Proc. IEEE Int. Symp. Inf. Theory*, 2019, pp. 2793–2797.

[35] A. Mallick, M. Chaudhari, and G. Joshi, "Fast and efficient distributed matrix-vector multiplication using rateless fountain codes," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process.*, 2019, pp. 8192–8196.

[36] A. Severinson, A. G. i Amat, and E. Rosnes, "Block-diagonal and lt codes for distributed computing with straggling servers," *IEEE Trans. Commun.*, vol. 67, no. 3, pp. 1739–1753, Mar. 2019.
[37] M. A. Maddah-Ali and U. Niesen, "Decentralized coded caching attains order-optimal memory-rate tradeoff," *IEEE/ACM Trans. Netw.*, vol. 23, no. 4, pp. 1029–1040, Aug. 2015.
[38] M. M. Amiri and D. Gündüz, "Fundamental limits of coded caching: Improved delivery rate-cache capacity tradeoff," *IEEE Trans. Commun.*, vol. 65, no. 2, pp. 806–815, Feb. 2017.
[39] M. Ji, A. M. Tulino, J. Llorca, and G. Caire, "Order-optimal rate of caching and coded multicasting with random demands," *IEEE Trans. Inf. Theory*, vol. 63, no. 6, pp. 3923–3949, Jun. 2017.
[40] A. M. Ibrahim, A. A. Zewail, and A. Yener, "Coded caching for heterogeneous systems: An optimization perspective," *IEEE Trans. Commun.*, vol. 67, no. 8, pp. 5321–5335, Aug. 2019.
[41] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE Trans. Neural Netw.*, vol. 20, no. 1, pp. 61–80, Jan. 2009.
[42] G. Karypis and V. Kumar, "Metis–unstructured graph partitioning and sparse matrix ordering system, version 2.0," 1995.
[43] S. Prakash, A. Reisizadeh, R. Pedarsani, and S. Avestimehr, "Coded computing for distributed graph analytics," 2018, *arXiv:1801.05522*.
[44] P. Erdös and A. Rényi, "On random graphs, i," *Publicationes Mathematicae (Debrecen)*, vol. 6, pp. 290–297, 1959.
[45] E. N. Gilbert, "Random graphs," *Ann. Math. Statist.*, vol. 30, no. 4, pp. 1141–1144, 1959.
[46] R. Albert and A.-L. Barabási, "Statistical mechanics of complex networks," *Rev. Modern Phys.*, vol. 74-1, pp. 47–97, 2002.
[47] J. Riley, "StarCluster-numPy/SciPy computing on Amazon's elastic compute cloud (EC2)," in *Proc. 9th Python Sci. Conf.*, 2010. [Online]. Available: http://star.mit.edu/cluster/index.html.
[48] L. Dalcín, R. Paz, and M. Storti, "MPI for python," *J. Parallel and Distrib. Comput.*, vol. 65, no. 9, pp. 1108–1115, 2005.
[49] K. Konstantinidis and A. Ramamoorthy, "Leveraging coding techniques for speeding up distributed computing," in *Proc. IEEE Global Commun. Conf.*, Abu Dhabi, United Arab Emirates, 2018, pp. 1–6.
[50] T. M. Cover and J. A. Thomas, *Elements of Information Theory*. Hoboken, NJ, USA: Wiley, 2012.

Başak Güler (Member, IEEE) received the B.Sc. degree in electrical and electronics engineering from Middle East Technical University (METU), Ankara, Turkey, in 2009 and the M.Sc. and Ph.D. degrees in electrical engineering from Wireless Communications and Networking Laboratory, Pennsylvania State University, University Park, PA, in 2012 and 2017, respectively. She is currently a Postdoctoral Scholar at the Department of Electrical Engineering, University of Southern California. Her research interests include distributed computing, graph processing, machine learning, source coding, and interference management in heterogeneous wireless networks.

A. Salman Avestimehr (Fellow, IEEE) received the B.S. degree in electrical engineering from the Sharif University of Technology, in 2003 and the M.S. and Ph.D. degree in electrical engineering and computer science from the University of California, Berkeley, in 2005 and 2008, respectively. He is a Professor and director of the Information Theory and Machine Learning (vITAL) research lab at the Electrical and Computer Engineering Department of University of-Southern California. His research interests include information theory, coding theory, and large-scale distributed computing and machine learning.

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

Antonio Ortega (Fellow, IEEE) received the Telecommunications Engineering degree from the Universidad Politecnica de Madrid, Madrid, Spain, in 1989 and the Ph.D. degree in electrical engineering from Columbia University, New York, NY, in 1994. In 1994 he joined the Electrical Engineering department at the University of Southern California (USC), where he is currently a Professor and has served as Associate Chair. He is a Fellow of the IEEE and EURASIP, and a member of ACM and APSIPA and has served as a member of the Board of Governors of the IEEE Signal Processing Society. He was Technical Program Co-Chair of ICIP 2008, PCS 2013, PCS 2018 and DSW 2018. He was the Inaugural Editor-in-Chief of the APSIPA Transactions on Signal and Information Processing and is now Editor-in-Chief of the IEEE TRANSACTIONS ON SIGNAL AND INFORMATION PROCESSING OVER NETWORKS. He has received several paper awards, including most recently the 2016 Signal Processing Magazine award, and was a Plenary Speaker at ICIP 2013. His recent research work is focusing on graph signal processing, machine learning, multimedia compression and wireless sensor networks. Over 40 PhD students have completed their PhD thesis under his supervision at USC and his work has led to over 400 publications in international conferences and journals, as well as several patents.