Coding for a Single Sparse Inverse Problem

Yaoqing Yang, Pulkit Grover, Soummya Kar
Email: {yyaoqing, pgrover, soummyak}@andrew.cmu.edu

***Abstract*—We propose a coded computing technique for making the power-iteration method of solving a single sparse linear inverse problem robust to erasure-type noise. We observe that for sparse inverse problems, codes with dense generator matrices can significantly increase storage costs. Thus, we propose coding the power-iteration computation using *sparse* generator matrices. Surprisingly, despite the poor error-correction ability of codes with sparse generator matrices, we show through both theoretical analysis and simulations that these codes are sufficient to achieve almost the same convergence rate as noiseless power iterations, provided that a new decoding algorithm that we call "substitute decoding" is used.**

I. INTRODUCTION

Sparse linear inverse problems solved by power iterations (see (1)) have been widely used in different applications, such as webpage ranking [1], semi-supervised learning [2] and clustering [3]. In this work, we utilize error correcting codes to make the parallel computing of power iterations robust to system noise such as stragglers and erasures. Researchers from the coded computing field have investigated a wide range of applications [4]–[9]. A comprehensive literature survey is in the extended version of the paper [10], including full proofs of theorems and extended applications. Our previous work [11] considered coded computing for power iterations when many problem instances with the same linear system matrix $\mathbf{M}\mathbf{x}_i = \mathbf{y}_i, i = 1, 2, \ldots, m$ are computed in parallel. However, a more common setting is the computation of a *single* linear inverse problem $\mathbf{M}\mathbf{x} = \mathbf{y}$ where the linear system matrix $\mathbf{M}$ is square and sparse[1]. In this case, usually, the system matrix is too large to fit in the memory of a single machine, and thus distributed computing is necessary.

Existing results in coded computing typically use dense generator matrices, such as those of MDS codes, to ensure good error-correcting capability. However, a dense encoding of the sparse matrix $\mathbf{M}$ can significantly increase the number of non-zero entries, and hence increase storage cost and computation time. In fact, dense encoding using MDS codes can easily make the sparse problem completely non-sparse. For example, the necessity of using sparse codes for sparse data and the idea of storing uncoded relevant sparse data at each worker as specified by the encoding matrix is discussed in [9]. In this work, we use codes with sparse generator matrices (low-density generator matrices, or LDGM) and show that, despite the bad error-correcting ability of LDGM codes, these codes can be made surprisingly efficient in maintaining the convergence rate of power iterations. As we show in simulations (see Section III-D), even for sparse generator matrices with only 2 non-zeros in each row, coded power iterations work significantly better than replication-based and uncoded power iterations in convergence rate. When there are 3 non-zeros in each row, the convergence rate of noiseless power iteration can be approximately achieved.

The key is to use a novel decoding algorithm that we call "substitute decoding". The intuitive idea is to extract the largest amount of available information from partial coded results in the computation of the power iteration $\mathbf{x}_{t+1} = \mathbf{y} + \mathbf{B}\mathbf{x}_t$, and substitute the complementary unknown information by the available side information $\mathbf{x}_t$. As we show in our main theorem (Theorem III.1), substitute decoding can reduce error by a multiple factor δ. This δ is linear in the rank of the (partial) generator matrix formed by the linear combinations from non-erased workers, and δ drops to 0 when the (partial) generator matrix is close to full-rank. The convergence rate of noiseless computation can be achieved when δ is small and for certain type of sparse linear system matrices $\mathbf{B}$.

II. PROBLEM FORMULATION

A. Preliminaries on the Power Iteration Method

We use the PageRank problem [1], [12] as an example to introduce power iterations. The PageRank and the more general personalized PageRank problem aim to measure the importance score of the nodes on a graph by solving the linear problem $\mathbf{x} = c\mathbf{r} + (1-c)\mathbf{A}\mathbf{x}$, where $c = 0.15$ is a constant, N is the number of nodes, $\mathbf{A}$ is the column-normalized (stochastic) adjacency matrix and $\mathbf{r} \in \mathbb{R}^N$ represents the preference of different users or topics. The classical method to solve PageRank is power iteration, which iterates $\mathbf{x}_{t+1} = c\mathbf{r} + (1-c)\mathbf{A}\mathbf{x}_t$ until convergence [1]. If we define $\mathbf{B} = (1-c)\mathbf{A}$ and $\mathbf{y} = c\mathbf{r}$. Then, we obtain the general form of power iteration:

$$\mathbf{x}_{t+1} = \mathbf{y} + \mathbf{B}\mathbf{x}_t. \tag{1}$$

The condition to ensure that (1) converges to the true solution $\mathbf{x}^*$ is that the spectral radius $\rho(\mathbf{B}) < 1$. For the PageRank problem, $\rho(\mathbf{B}) = 1 - c$ and (1) always converges to $\mathbf{x}^*$.

B. Uncoded Distributed Computing of Power Iteration

The most straightforward uncoded way is to partition $\mathbf{B}$ row-wise into several row blocks and store them in the memory of several workers. Denote the number of workers by P, and this is also the number of row blocks for uncoded computation. At the beginning of the t-th iteration, a central node sends the current result $\mathbf{x}_t$ to all workers. Then, the worker that has the row block $\mathbf{B}_i$ computes $\mathbf{B}_i\mathbf{x}_t$ and sends it back to the central node. At the end of the iteration, the central node concatenates all the results $\mathbf{B}_i\mathbf{x}_t, i = 1, 2, \ldots, P$ from the P workers to obtain $\mathbf{B}\mathbf{x}_t$, and computes $\mathbf{x}_{t+1} = \mathbf{B}\mathbf{x}_t + \mathbf{y}$.

[1]Notice the difference between this problem setting and coding for gradient descent [5], [9]. Power-iteration, and the more general Jacobi iteration, are not gradient descent methods.

978-1-5386-4780-6/18/$31.00©2018 IEEE

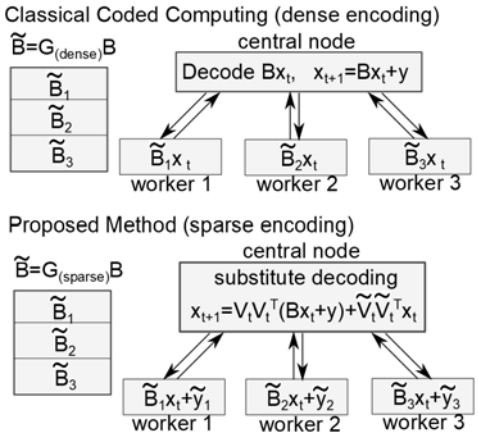

Fig. 1. A comparison between the classical coded computing and the proposed coded computing technique. In the proposed method, we only show the scalar version as mentioned in Remark 1.

C. Preliminaries and Notation on Coded Computing

Error correcting codes can help make distributed computing robust to stragglers and erasures. We first present the straightforward way of coded computing of the power iteration (1), and point out a drawback of it. As shown in the upper part of Fig. 1, $\mathbf{B}_{N\times N}$ is partitioned into k row blocks and linearly combined into $P > k$ row blocks $\widetilde{\mathbf{B}}_i, i = 1, 2, \ldots, P$ using a (P, k) linear code with a generator matrix $\mathbf{G}$ of size $P \times k$. Each block $\widetilde{\mathbf{B}}_i$ is stored at one of the P workers. For simplicity, we assume that N is divisible by k, and denote the number of rows in each row block by $b = \frac{N}{k}$. The encoding can be mathematically represented as

$$\widetilde{\mathbf{B}} = (\mathbf{G}_{P\times k} \otimes \mathbf{I}_b)\mathbf{B}, \tag{2}$$

where the Kronecker product is because we encode row blocks.

Example 1. Suppose $\mathbf{B}$ is partitioned into $\mathbf{B}_{N\times N} = \begin{bmatrix}\mathbf{B}_1\\ \mathbf{B}_2\end{bmatrix}$ and encoded into $\widetilde{\mathbf{B}}_{\frac{3}{2}N\times N} = [\mathbf{B}_1^\top, \mathbf{B}_2^\top, \mathbf{B}_1^\top + \mathbf{B}_2^\top]^\top$. The generator matrix is $\mathbf{G}_{3\times 2} = \begin{bmatrix}1 & 0\\ 0 & 1\\ 1 & 1\end{bmatrix}$. The number of row blocks in $\mathbf{B}$ is $k = 2$, the number of workers is $P = 3$ (which is also the code length), and the number of rows in each row block is $b = \frac{N}{2}$. The encoding can indeed by written as (2), because

$$\begin{bmatrix}\mathbf{B}_1\\ \mathbf{B}_2\\ \mathbf{B}_1+\mathbf{B}_2\end{bmatrix} = \begin{bmatrix}\mathbf{I}_b & \\ & \mathbf{I}_b\\ \mathbf{I}_b & \mathbf{I}_b\end{bmatrix}\begin{bmatrix}\mathbf{B}_1\\ \mathbf{B}_2\end{bmatrix} = \left(\begin{bmatrix}1 & 0\\ 0 & 1\\ 1 & 1\end{bmatrix}\otimes \mathbf{I}_b\right)\begin{bmatrix}\mathbf{B}_1\\ \mathbf{B}_2\end{bmatrix}.$$

At each iteration, the i-th worker computes $\widetilde{\mathbf{B}}_i\mathbf{x}_t$. Now, we define two operations (shown in Fig. 2). Denote by $\mathbf{v} = \text{vec}(\mathbf{X})$ the operation to vectorize the matrix $\mathbf{X}$ into the concatenation of its transposed rows, and denote by $\mathbf{X} = \text{mat}(\mathbf{v})$ the operation to partition the column vector $\mathbf{v}$ into small vectors and stack the transposed small vectors into the rows of $\mathbf{X}$. We always partition the vector $\mathbf{v}$ into smaller ones of length $b = \frac{N}{k}$ which represents the row-block size at each worker. Then, it is straightforward to show that any operation of the form $\mathbf{x} = (\mathbf{A} \otimes \mathbf{I}_b) \cdot \mathbf{v}$ can be rewritten in a compact form $\mathbf{x} = \text{vec}(\mathbf{A}\text{mat}(\mathbf{v}))$. Therefore, from (2), the obtained results at the central node is

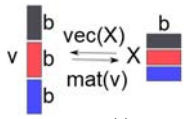

Fig. 2. An illustration on the vec $(\cdot)$ and the mat $(\cdot)$ operations.

$$\widetilde{\mathbf{B}}\mathbf{x} = (\mathbf{G}_{P\times k} \otimes \mathbf{I}_b)\mathbf{B}\mathbf{x} = \text{vec}\left(\mathbf{G} \cdot \text{mat}\left(\mathbf{B}\mathbf{x}\right)\right). \tag{3}$$

The matrix-version of $\widetilde{\mathbf{B}}\mathbf{x}$ is hence $\mathbf{G}\cdot\text{mat}(\mathbf{B}\mathbf{x})$, which means each column of the matrix-version of $\widetilde{\mathbf{B}}\mathbf{x}$ is a codeword. In the presence of stragglers or erasures, the coded results $\mathbf{G}\cdot\text{mat}(\mathbf{B}\mathbf{x})$ would lose some of the rows, and the decoding can be done in a parallel fashion on each column. There are b columns in $\mathbf{G}\cdot\text{mat}(\mathbf{B}\mathbf{x})$. Thus, the decoding complexity is $b\mathcal{N}_{\text{dec}}$, where $\mathcal{N}_{\text{dec}}$ is the complexity of decoding a single codeword.

A main drawback of the above method is that the generator matrix $\mathbf{G}$ is usually dense, such as MDS codes. However, the system matrix $\mathbf{B}$ is often sparse, as in the PageRank problem. Therefore, using a dense $\mathbf{G}$ may significantly increase the number of non-zeros in $\widetilde{\mathbf{B}}$. For example, if $\mathbf{G}$ has 20 non-zeros in each row, it means that the sub-matrices $\widetilde{\mathbf{B}}_i$ stored at each worker can have at most 20 times larger size than the uncoded case. In section III-A, we show that $\mathbf{G}$ can actually be very sparse (such as 2 ones in each row), while the computation of power iterations can still remain robust to system noise.

D. Preliminaries on the Proposed Technique

In our technique, similar to standard coded computing, the linear system matrix $\mathbf{B}$ is partitioned into k row blocks and encoded into P row blocks using a (P, k) code with rate $R = \frac{k}{P}$. Each encoded row block is stored at one worker. We now state an important difference in our code: *at each iteration, we use a different generator matrix* $\mathbf{G}^{(t)}$*, but its sparsity pattern remains the same across iterations.* In Example 1, $\mathbf{G} = \begin{bmatrix}1 & 0\\ 0 & 1\\ 1 & 1\end{bmatrix}$, so now at each iteration t we have a different generator matrix $\mathbf{G}^{(t)} = \begin{bmatrix}g_{11}(t) & 0\\ 0 & g_{22}(t)\\ g_{31}(t) & g_{32}(t)\end{bmatrix}$. We choose each non-zero $g_{ij}(t)$ to be a standard Gaussian r.v., and all of these r.v.s are independent of each other. The fixed sparsity pattern $\mathbf{G}$ determines which (sparse) row blocks of the uncoded matrix $\mathbf{B}$ are stored at each worker. In particular, $\mathbf{B}_j, j = 1, \ldots, k$ is stored at worker-i, $i = 1 \ldots P$, when $\mathbf{G}_{i,j} = 1$. In Example 1, $\mathbf{B}_1$ is stored at worker-1 and worker-3, and $\mathbf{B}_2$ is stored at worker-2 and worker-3. However, instead of precomputing the encoded sub-matrices $\widetilde{\mathbf{B}}_i$, the i-th worker just stores its required row blocks in $\mathbf{B}$, because the code is time-varying. Similarly, we also partition the vector $\mathbf{y}$ into k sub-vectors of length b and store them in the P workers in the exactly same fashion as $\mathbf{B}$. At the t-th iteration, worker-i computes $(\mathbf{G}^{(t)})_{i\text{-th row}}\text{mat}(\mathbf{B}\mathbf{x} + \mathbf{y})$. In Example 1, this means that worker-3 stores $\mathbf{B}_1$ and $\mathbf{B}_2$ at the local memory. At the t-th iteration, it computes $\mathbf{B}_1\mathbf{x} + \mathbf{y}_1$ and $\mathbf{B}_2\mathbf{x} + \mathbf{y}_2$

and encodes them to $g_{31}(t)(\mathbf{B}_1\mathbf{x}+\mathbf{y}_1)+g_{32}(t)(\mathbf{B}_2\mathbf{x}+\mathbf{y}_2)$ using the linear coefficients in $\mathbf{G}^{(t)}$. Since the sparsity pattern is fixed, although the code is time-varying, stored data at each worker remains the same.

At each iteration, a random fraction ϵ of the workers fails to send their results back due to either erasures (packet losses) or stragglers (the communications with slow workers are discarded to save time). Then, at the central node, available results are $\mathbf{G}_s^{(t)}\text{mat}\,(\mathbf{B}\mathbf{x}_t+\mathbf{y})$, where $\mathbf{G}_s^{(t)}$ is the sub-matrix of $\mathbf{G}^{(t)}$ formed by the linear combinations at the non-erased workers. We call $\mathbf{G}_s^{(t)}$ a "partial generator matrix". In classical coded computing, if a dense Vandermonde-type code is used, the desired result $\mathbf{B}\mathbf{x}_t+\mathbf{y}$ can be decoded from $\mathbf{G}_s^{(t)}\text{mat}\,(\mathbf{B}\mathbf{x}_t+\mathbf{y})$ if $1-\epsilon>R$, because any square sub-matrix $\mathbf{G}_s^{(t)}$ of a Vandermonde matrix is invertible. However, if $\mathbf{G}$ is sparse, even if ϵ is very small, it is possible that $\mathbf{B}\mathbf{x}_t+\mathbf{y}$ cannot be decoded because $\mathbf{G}_s^{(t)}$ may not be invertible.

III. Substitute Decoding of Coded Power Iterations

A. Substitute Decoding Algorithm for LDGM codes

In the previous section, we suggested a plausible tradeoff on the sparsity of $\mathbf{G}$. If it is dense, storage cost is high. If it is sparse, $\mathbf{G}_s^{(t)}$ may not be inverted to get the desired results $\mathbf{B}\mathbf{x}_t+\mathbf{y}$. Surprisingly, we show that $\mathbf{G}_s^{(t)}$ can actually be made sparse, while its noise-tolerance is maintained. The key observation is that although $\mathbf{G}_s^{(t)}$ may not have full column rank, we can get partial information of $\mathbf{B}\mathbf{x}_t+\mathbf{y}$ from $\mathbf{G}_s^{(t)}\text{mat}\,(\mathbf{B}\mathbf{x}_t+\mathbf{y})$. Suppose that the SVD of $\mathbf{G}_s^{(t)}$ is

$$(\mathbf{G}_s^{(t)})_{(1-\epsilon)P\times k}=\mathbf{U}_t\mathbf{D}_t\mathbf{V}_t^\top, \tag{4}$$

where the matrix $\mathbf{V}_t$ has orthonormal columns and has size $k\times\text{rank}(\mathbf{G}_s^{(t)})$. By multiplying $\mathbf{L}_t=\mathbf{D}_t^{-1}\mathbf{U}_t^\top$ to the partial coded results $\mathbf{G}_s^{(t)}\text{mat}\,(\mathbf{B}\mathbf{x}_t+\mathbf{y})$, the central node obtains

$$(\mathbf{D}_t^{-1}\mathbf{U}_t^\top)\mathbf{G}_s^{(t)}\text{mat}\,(\mathbf{B}\mathbf{x}_t+\mathbf{y})\overset{(a)}{=}\mathbf{V}_t^\top\text{mat}\,(\mathbf{B}\mathbf{x}_t+\mathbf{y}), \tag{5}$$

where (a) follows from (4). Then, the central node finds an orthonormal basis of the orthogonal complementary space of the column space of $\mathbf{V}_t$, i.e., an orthonormal basis of $\mathcal{R}^\perp(\mathbf{V}_t)$, and forms the basis into matrix $\widetilde{\mathbf{V}}_t$, such that $[\mathbf{V}_t,\widetilde{\mathbf{V}}_t]$ is an orthonormal matrix[2] of size $k\times k$, i.e., $\mathbf{V}_t$, $\widetilde{\mathbf{V}}_t$ are orthogonal to each other, and

$$\mathbf{V}_t\mathbf{V}_t^\top+\widetilde{\mathbf{V}}_t\widetilde{\mathbf{V}}_t^\top=\mathbf{I}_k. \tag{6}$$

The central node uses $\mathbf{V}_t^\top\text{mat}\,(\mathbf{B}\mathbf{x}_t+\mathbf{y})$ obtained from (5) and the stored $\mathbf{x}_t$ as side information to obtain a good estimate of $\mathbf{B}\mathbf{x}_t+\mathbf{y}$ to compute $\mathbf{x}_{t+1}$. In particular, $\mathbf{x}_{t+1}$ is

$$\mathbf{x}_{t+1}=\text{vec}\left([\mathbf{V}_t,\widetilde{\mathbf{V}}_t]\cdot\begin{bmatrix}\mathbf{V}_t^\top\text{mat}\,(\mathbf{B}\mathbf{x}_t+\mathbf{y})\\ \widetilde{\mathbf{V}}_t^\top\text{mat}\,(\mathbf{x}_t)\end{bmatrix}\right). \tag{7}$$

This is equivalent to

$$\mathbf{x}_{t+1}=\text{vec}\left(\mathbf{V}_t\mathbf{V}_t^\top\text{mat}\,(\mathbf{B}\mathbf{x}_t+\mathbf{y})+\widetilde{\mathbf{V}}_t\widetilde{\mathbf{V}}_t^\top\text{mat}\,(\mathbf{x}_t)\right). \tag{8}$$

[2]If $\mathbf{G}_s^{(t)}$ has full rank, $\mathbf{V}_t$ is already a square orthonormal matrix and in this case $\widetilde{\mathbf{V}}_t$ is the NULL matrix, because $\mathcal{R}^\perp(\mathbf{V}_t)$ is the trivial space $\{\mathbf{0}\}$.

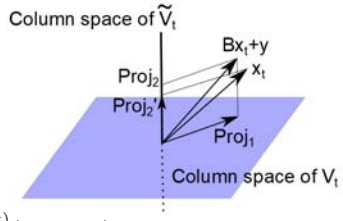

Fig. 3. From $\mathbf{G}_s^{(t)}(\mathbf{B}\mathbf{x}_t+\mathbf{y})$, we can get the projection of $\mathbf{B}\mathbf{x}_t+\mathbf{y}$ onto the column space of $\mathbf{V}_t$ (see Proj$_1$). For the unknown part Proj$_2$, we use the projection of $\mathbf{x}_t$ instead, which is Proj$_2$'.

Remark 1. (**Intuition underlying substitute decoding**) We provide intuition by looking at a scalar-version as shown in the lower part of Fig. 1. In the scalar version, $b=1$, $\mathbf{I}_b=1$ and (8) becomes

$$\mathbf{x}_{t+1}=\mathbf{V}_t\mathbf{V}_t^\top(\mathbf{B}\mathbf{x}_t+\mathbf{y})+\widetilde{\mathbf{V}}_t\widetilde{\mathbf{V}}_t^\top\mathbf{x}_t. \tag{9}$$

Since $\mathbf{V}_t$ and $\widetilde{\mathbf{V}}_t$ have orthogonal columns and they are orthogonal to each other, $\mathbf{V}_t\mathbf{V}_t^\top$ and $\widetilde{\mathbf{V}}_t\widetilde{\mathbf{V}}_t^\top$ are two projection matrices onto the column spaces of $\mathbf{V}_t$ and $\widetilde{\mathbf{V}}_t$ respectively. Since $\mathbf{V}_t$ is obtained from SVD of $\mathbf{G}_s^{(t)}$ (see equation (4)), the projection $\mathbf{V}_t\mathbf{V}_t^\top$ is the projection to the row space of $\mathbf{G}_s^{(t)}$. The intuition of substitute decoding is that even if we cannot get the exact value of $\mathbf{B}\mathbf{x}_t+\mathbf{y}$ by inverting the sparse $\mathbf{G}_s^{(t)}$, we can at least obtain the projection of $\mathbf{B}\mathbf{x}_t+\mathbf{y}$ onto the row space of $\mathbf{G}_s^{(t)}$. Then, for the remaining unknown part of $\mathbf{B}\mathbf{x}_t+\mathbf{y}$, i.e., the projection of $\mathbf{B}\mathbf{x}_t+\mathbf{y}$ onto the right null space of $\mathbf{G}_s^{(t)}$, we use the projection $\widetilde{\mathbf{V}}_t\widetilde{\mathbf{V}}_t^\top\mathbf{x}_t$ of the side information $\mathbf{x}_t$ to substitute. This intuition is illustrated in Fig. 3. We give an outline of the substitute decoding algorithm in Algorithm 1.

B. Focus on communication time: an explanation

First, we analyze communication cost. For each worker, the communication cost at each iteration comes from the transmission of $\mathbf{x}_t$ of length N and the transmission of the result of length $b=N/k$. So the total number of communicated floating point numbers is $N(1+1/k)$ which is linear in N.

For the computation cost at the central node, the SVD has complexity $O(kP^2)$ which is negligible. The computation of (5) and the substitute decoding given in (7) together have complexity $O(k^2b)=O(kN)$ which is linear in N. The vectorization and matricization steps have a negligible cost.

The i-th worker computes a sub-vector of the entire $\mathbf{B}\mathbf{x}$. Denote by E the number of non-zeros in $\mathbf{B}$. Suppose the sparse generator matrix $\mathbf{G}$ has d ones in each row. Then, the complexity at each worker is $O((dE)/k)$. This complexity can be superlinear in N for dense graphs, but usually, the average degree of a graph is a large constant. This means the computation cost at each worker is also linear in N, but with a large constant.

Since communication and computation complexity are both linear in N, constant factors matter in determining which cost is dominant. When the code size is small, the communication time can dominate even when the average degree of the graph is much larger than 1, because, in practice, communication time can dominate computing time even if computing is

in scaling sense more expensive, simply due to the fact that communication is slower than computation in current technologies [5], [7]. In our simulations, we focus on the communication cost and analyze the cost mostly from this perspective. However, we still cannot use a dense code for encoding due to increased storage cost.

Algorithm 1 Coded Power Iterations

Input: Input $\mathbf{y}$, matrix $\mathbf{B}$ and sparse pattern $\mathbf{G}$.
Preprocessing: Partition $\mathbf{B}$ into row blocks and $\mathbf{y}$ into sub-vectors and store them distributedly as specified by the sparse pattern $\mathbf{G}$. Generate a series of random generator matrices $\mathbf{G}^{(t)}, t = 1, 2, \ldots, T$.
Central Node: Send out $\mathbf{x}_t$ at each iteration and receive partial coded results. Compute $\mathbf{x}_{t+1}$ using substitute decoding (8), where $\mathbf{V}$ and $\widetilde{\mathbf{V}}$ are obtained from SVD (4) and $\text{mat}\left(\mathbf{B}\mathbf{x}_t + \mathbf{y}\right)$ is computed using (5).
Workers: Worker-i computes $(\mathbf{G}^{(t)})_{i\text{-th row}}\text{mat}\left(\mathbf{B}\mathbf{x} + \mathbf{y}\right)$.
Output: The central node outputs $\mathbf{x}_T$.

C. Convergence Analysis of the Coded Power Iterations

Denote by $\mathbf{x}^*$ the true solution of $\mathbf{x} = \mathbf{B}\mathbf{x} + \mathbf{y}$. Then,

$$\begin{aligned}\mathbf{x}^* \overset{(a)}{=} &\text{vec}\left(\mathbf{V}_t\mathbf{V}_t^\top \text{mat}\left(\mathbf{x}^*\right)\right) + \text{vec}\left(\widetilde{\mathbf{V}}_t\widetilde{\mathbf{V}}_t^\top \text{mat}\left(\mathbf{x}^*\right)\right)\\ =&\text{vec}\left(\mathbf{V}_t\mathbf{V}_t^\top \text{mat}\left(\mathbf{B}\mathbf{x}^* + \mathbf{y}\right) + \widetilde{\mathbf{V}}_t\widetilde{\mathbf{V}}_t^\top \text{mat}\left(\mathbf{x}^*\right)\right),\end{aligned} \tag{10}$$

where (a) holds because of (6). Defining the remaining error as $\mathbf{e}_t = \mathbf{x}_t - \mathbf{x}^*$ and subtracting (10) from (8), we have

$$\mathbf{e}_{t+1} = \text{vec}\left(\mathbf{V}_t\mathbf{V}_t^\top \text{mat}\left(\mathbf{B}\mathbf{e}_t\right)\right) + \text{vec}\left(\widetilde{\mathbf{V}}_t\widetilde{\mathbf{V}}_t^\top \text{mat}\left(\mathbf{e}_t\right)\right). \tag{11}$$

Remark 2. (Why substitute decoding suppresses error) Before presenting formal proofs, we show the underlying intuition on why the substitute decoding (7) approximates the noiseless power iteration $\mathbf{x}_{t+1} = \mathbf{B}\mathbf{x}_t + \mathbf{y}$ well. Again, we look at the scalar version, i.e., $\mathbf{I}_b = 1$. In this case, (11) becomes

$$\mathbf{e}_{t+1} = \mathbf{V}_t\mathbf{V}_t^\top\mathbf{B}\mathbf{e}_t + \widetilde{\mathbf{V}}_t\widetilde{\mathbf{V}}_t^\top\mathbf{e}_t. \tag{12}$$

Ideally, we want $\mathbf{e}_{t+1} = \mathbf{B}\mathbf{e}_t$ since $\mathbf{B}$ is a contraction matrix (recall that $\rho(\mathbf{B}) < 1$). Due to noise, we can only realize this contraction in the column space of $\mathbf{V}_t$, which is the first term $\mathbf{V}_t\mathbf{V}_t^\top\mathbf{B}\mathbf{e}_t$. Although the partial generator matrix $\mathbf{G}_s^{(t)}$ may not have full rank due to being sparse (i.e., $\dim(\mathcal{R}(\mathbf{V}_t)) < k$), $\mathbf{G}_s^{(t)}$ can be close to full rank. This means that the column space $\mathcal{R}(\widetilde{\mathbf{V}}_t)$ can have low dimension. Therefore, for the second term in (12), the projection matrix $\widetilde{\mathbf{V}}_t\widetilde{\mathbf{V}}_t^\top$ suppresses the larger error $\mathbf{e}_t$ (compared to $\mathbf{B}\mathbf{e}_t$) by projecting it onto the low-dimensional space $\mathcal{R}(\widetilde{\mathbf{V}}_t)$. In Theorem III.1, we will show $\widetilde{\mathbf{V}}_t\widetilde{\mathbf{V}}_t^\top$ multiplies $\mathbb{E}[\|\mathbf{e}_t\|^2]$ with a small multiple factor that decreases to 0 linearly as $\text{rank}(\mathbf{G}_s^{(t)})$ increases.

Definition 1. (Combined cyclic sparsity pattern) The sparsity pattern matrix satisfies $\mathbf{G} = \begin{bmatrix}\mathbf{S}_1\\ \mathbf{S}_2\end{bmatrix}$, where $\mathbf{S}_1$ and $\mathbf{S}_2$ are both $k \times k$ square cyclic matrices with d non-zeros in each row.

Assumption 1. (Random failures) At each iteration, a random subset of the workers fails to compute the result due to either stragglers or erasure-type errors. Failure events are independent across iterations.

Theorem III.1. (Convergence Rate) If the sparsity pattern $\mathbf{G}$ in Definition 1 is used and Assumption 1 holds, the remaining error $\mathbf{e}_t = \mathbf{x}_t - \mathbf{x}^*$ of Algorithm 1 satisfies

$$\mathbb{E}[\|\mathbf{e}_{t+1}\|^2] = (1 - \delta_t)\mathbb{E}[\|\mathbf{B}\mathbf{e}_t\|^2] + \delta_t\mathbb{E}[\|\mathbf{e}_t\|^2], \tag{13}$$

where

$$\delta_t = 1 - \frac{\mathbb{E}[\text{rank}(\mathbf{G}_s^{(t)})]}{k}. \tag{14}$$

The proof is in Section IV. From Theorem III.1, we can simply upper-bound $\mathbb{E}[\|\mathbf{B}\mathbf{e}_t\|^2]$ by $\|\mathbf{B}\|_2^2\,\mathbb{E}[\|\mathbf{e}_t\|^2]$ and hence

$$\mathbb{E}[\|\mathbf{e}_{t+1}\|^2] \le [(1 - \delta_t)\|\mathbf{B}\|_2^2 + \delta_t] \cdot \mathbb{E}[\|\mathbf{e}_t\|^2]. \tag{15}$$

This means that when δ_t is close to 0, i.e., when $\mathbf{G}_s^{(t)}$ is close to full rank, $\mathbb{E}[\|\mathbf{e}_t\|^2]$ converges to 0 with rate close to $\|\mathbf{B}\|_2^{2t}$. In Table I, we show how δ_t changes with the degree d in Definition 1. Notice that the noiseless power iteration converges with rate $(\rho(\mathbf{B}))^{2t}$. For the PageRank problem, $\mathbf{B} = (1 - c)\mathbf{A}$ and $\mathbf{A}$ is the column-normalized adjacency matrix. We show in Appendix B in [10] that for Erdös-Rényi model $G(N, p)$, $\Pr\left(\|\mathbf{A}\| > \sqrt{\frac{1+\epsilon}{1-\epsilon}}\rho(\mathbf{A})\right) < 3Ne^{-\epsilon^2 Np/8}$. This means that with high probability $\|\mathbf{A}\|_2 \approx \rho(\mathbf{A})$ and hence $\|\mathbf{B}\|_2 \approx \rho(\mathbf{B})$, and the convergence rates of coded power iteration and noiseless power iteration are close. Here, in $G(N, p)$, it suffices for p to be $\Omega(\log n/(n\epsilon^2))$.

D. Simulation Results on the Twitter Graph

To support Theorem III.1, we compare uncoded, replication-based power iterations and Algorithm 1 on the Twitter graph [13]. We also show the result of noiseless power iterations. We run 100 independent simulations and average the results. There are $P = 20$ workers. In each iteration, 50% of the workers are disabled randomly. In the uncoded simulation, the graph matrix is partitioned into $P = 20$ row blocks. The central node updates $\mathbf{x}_{t+1} = \mathbf{B}\mathbf{x}_t + \mathbf{y}$ on the row blocks where results are available, and maintains the unavailable rows as $\mathbf{x}_t$. In the replication-based simulation, $\mathbf{B}$ is partitioned into 10 row blocks and each one is replicated in 2 workers. Therefore, in each iteration, effectively 50% of the entries in $\mathbf{x}_t$ get updated in the uncoded simulation and about 75% of the entries in $\mathbf{x}_t$ get updated in the replication-based simulation. For the coded case, the sparse pattern matrix $\mathbf{G}$ is randomly generated using Definition 1 with degree $d = 2$ and $d = 3$. The code is a $(20, 10)$ code with rate $1/2$. We show in Table I how the sample average estimate of δ_t changes with the degree of $\mathbf{G}$.

Cost Analysis: We also compare the convergence rates against communication cost (see Fig. 4; right). For Algorithm 1, $\mathbf{B}$ is partitioned into $k = 10$ row blocks and encoded into 20. The communication complexity in each iteration is $N(1 + 1/k) = 1.1N$. Similarly, it can be shown that the communication complexity of uncoded and replication-based power iterations are respectively $N(1 + 1/P) = 1.05N$

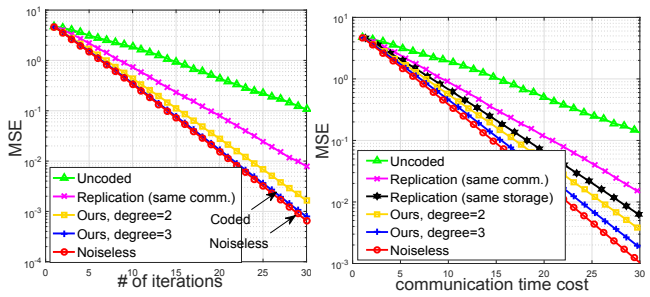

Fig. 4. The comparison between uncoded, replication-based and substitute-decoding-based power iterations on the Twitter graph. Substitute decoding (blue line) achieves almost exactly the same convergence rate as the noiseless case (red line) for the same number of iterations. Coded computing also beats the other techniques for the same communication time cost.

$d(\mathbf{G})$	2	3	4	5
δ_t	0.1294	0.0442	0.0243	0.0040

TABLE I
THE FACTOR δ_t DECREASES WHEN THE DEGREE OF $\mathbf{G}$ INCREASES.

and $N(1+1/k) = 1.1N$. Since the average degree of the sparse pattern is $d = 2 \sim 3$, computation cost and memory consumption only increase by a constant. We also plot the tradeoff for replication scheme with the same storage cost as $d = 3$. However, in this case, the communication complexity for replication is larger, which is $N(1+d/k) = 1.3N$. In the extended paper [10], we compare substitute-decoding-based coded computing with replication-based schemes for not only row-splitting on the $\mathbf{B}$ matrix, but also column-splitting and SUMMA-type (i.e., both row and column) splitting, in which we obtain even larger reductions in communication cost using coded computing.

IV. Proof of Theorem III.1

Lemma IV.1. If the sparsity pattern in Definition 1 is used and Assumption 1 holds, the projection matrix $\mathbf{V}_t\mathbf{V}_t^\top$ satisfies

$$\mathbb{E}[\mathbf{V}_t\mathbf{V}_t^\top] = (1-\delta_t)\mathbf{I}_k, \tag{16}$$

where the expectation is taken respect to the randomness of non-zero entries' values (the sparsity pattern $\mathbf{G}$ is fixed) and the randomness of the workers' failure events.

Proof. See Appendix A of the extended paper [10] for the full proof. The proof relies on proving some symmetric properties of $\mathbb{E}[\mathbf{V}_t\mathbf{V}_t^\top]$. We prove that the symmetry of standard Gaussian pdf on the real line ensures that all of the off-diagonal entries in $\mathbb{E}[\mathbf{V}_t\mathbf{V}_t^\top]$ are zero. Further, we prove that the "combined cyclic" structure of $\mathbf{G}$ in Definition 1 ensures that all diagonal entries on $\mathbb{E}[\mathbf{V}_t\mathbf{V}_t^\top]$ are identical. The two facts above show that $\mathbb{E}[\mathbf{V}_t\mathbf{V}_t^\top] = x\mathbf{I}_k$ for some constant x. Then, we can use a property of trace to compute x. □

We denote the projection $\mathbf{V}_t\mathbf{V}_t^\top$ by $\mathbf{P}_V$. Then, from (6), $\widetilde{\mathbf{V}}_t\widetilde{\mathbf{V}}_t^\top = \mathbf{I}_k - \mathbf{P}_V$. The first term $\text{vec}\left(\mathbf{V}_t\mathbf{V}_t^\top \text{mat}\left(\mathbf{Be}_t\right)\right) = \text{vec}\left(\mathbf{P}_V \text{mat}\left(\mathbf{Be}_t\right)\right)$ in (11) can be bounded by

$$\begin{aligned}
&\mathbb{E}\left[\left\|\text{vec}\left(\mathbf{P}_V\text{mat}\left(\mathbf{Be}_t\right)\right)\right\|^2\right] \overset{(a)}{=} \mathbb{E}\left[\left\|(\mathbf{P}_V\otimes\mathbf{I}_b)\mathbf{Be}_t\right\|^2\right] \\
&\overset{(b)}{=}\mathbb{E}\left[\text{trace}\left((\mathbf{Be}_t)^\top(\mathbf{P}_V\otimes\mathbf{I}_b)^\top(\mathbf{P}_V\otimes\mathbf{I}_b)\mathbf{Be}_t\right)\right] \\
&\overset{(c)}{=}\mathbb{E}\left[\text{trace}\left(\mathbf{Be}_t(\mathbf{Be}_t)^\top(\mathbf{P}_V\otimes\mathbf{I}_b)^\top(\mathbf{P}_V\otimes\mathbf{I}_b)\right)\right] \\
&\overset{(d)}{=}\text{trace}\left(\mathbb{E}\left[\mathbf{Be}_t(\mathbf{Be}_t)^\top\right]\mathbb{E}\left[(\mathbf{P}_V\otimes\mathbf{I}_b)^\top(\mathbf{P}_V\otimes\mathbf{I}_b)\right]\right) \\
&=\text{trace}\left(\mathbb{E}\left[\mathbf{Be}_t(\mathbf{Be}_t)^\top\right]\mathbb{E}\left[(\mathbf{P}_V^\top\mathbf{P}_V)\otimes\mathbf{I}_b\right]\right) \\
&\overset{(e)}{=}\text{trace}\left(\mathbb{E}\left[\mathbf{Be}_t(\mathbf{Be}_t)^\top\right]\mathbb{E}\left[\mathbf{P}_V\otimes\mathbf{I}_b\right]\right) \\
&\overset{(f)}{=}\text{trace}\left(\mathbb{E}\left[\mathbf{Be}_t(\mathbf{Be}_t)^\top\right]((1-\delta_t)\mathbf{I}_k)\otimes\mathbf{I}_b\right) \\
&=(1-\delta_t)\text{trace}\left(\mathbb{E}\left[\mathbf{Be}_t(\mathbf{Be}_t)^\top\right]\right) = (1-\delta_t)\mathbb{E}[\|\mathbf{Be}_t\|^2],
\end{aligned} \tag{17}$$

where (a) is from the property of mat-vec operations, (b) is because $(\mathbf{P}_V\otimes\mathbf{I}_b)\mathbf{Be}_t$ is a vector, (c) is because $\text{trace}(AB) = \text{trace}(BA)$, (d) is because trace and $\mathbb{E}$ commutes and the projection $\mathbf{P}_V$ only depends on the random partial generator matrix $\mathbf{G}_s$ and is independent of $\mathbf{e}_t$, (e) is because $\mathbf{P}_V$ is a projection matrix and (f) is from Lemma IV.1 and $\mathbf{P}_V = \mathbf{V}_t\mathbf{V}_t^\top$. Similarly, we can prove

$$\mathbb{E}\left[\left\|\text{vec}\left((\mathbf{I}_k-\mathbf{P}_V)\text{mat}\left(\mathbf{e}_t\right)\right)\right\|^2\right] = \delta_t\mathbb{E}[\|\mathbf{e}_t\|^2]. \tag{18}$$

Therefore

$$\begin{aligned}
&\mathbb{E}[\|\mathbf{e}_{t+1}\|^2] \\
&\overset{(a)}{=}\mathbb{E}[\left\|\text{vec}\left(\mathbf{V}_t\mathbf{V}_t^\top\text{mat}\left(\mathbf{Be}_t\right)\right)\right\|^2] + \mathbb{E}[\left\|\text{vec}\left(\widetilde{\mathbf{V}}_t\widetilde{\mathbf{V}}_t^\top\text{mat}\left(\mathbf{e}_t\right)\right)\right\|^2] \\
&\overset{(b)}{=}(1-\delta_t)\mathbb{E}[\|\mathbf{Be}_t\|^2] + \delta_t\mathbb{E}[\|\mathbf{e}_t\|^2],
\end{aligned} \tag{19}$$

where (a) is from (11) and the Pythagorean theorem, and (b) is from (17) and (18). Thus, we have completed the proof.

References

[1] L. Page, S. Brin, R. Motwani, and T. Winograd, "The pagerank citation ranking: Bringing order to the web." Stanford InfoLab, Tech. Rep., 1999.

[2] X. Zhu, Z. Ghahramani, and J. D. Lafferty, "Semi-supervised learning using gaussian fields and harmonic functions," in *ICML*, 2003, pp. 912–919.

[3] F. Lin and W. W. Cohen, "Power iteration clustering," in *ICML*, 2010, pp. 655–662.

[4] K. Lee, M. Lam, R. Pedarsani, D. Papailiopoulos, and K. Ramchandran, "Speeding up distributed machine learning using codes," in *ISIT*, 2016, pp. 1143–1147.

[5] R. Tandon, Q. Lei, A. G. Dimakis, and N. Karampatziakis, "Gradient coding," in *ICML*, 2017.

[6] S. Dutta, V. Cadambe, and P. Grover, "Short-dot: Computing large linear transforms distributedly using coded short dot products," in *NIPS*, 2016, pp. 2092–2100.

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

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

[9] C. Karakus, Y. Sun, S. Diggavi, and W. Yin, "Straggler mitigation in distributed optimization through data encoding," in *NIPS*, 2017, pp. 5440–5448.

[10] Y. Yang, M. Chaudhari, P. Grover, and S. Kar, "Coded iterative computing using substitute decoding," in *arxiv, to appear*, 2018.

[11] Y. Yang, P. Grover, and S. Kar, "Coded distributed computing for inverse problems," in *NIPS*, 2017, pp. 709–719.

[12] T. H. Haveliwala, "Topic-sensitive pagerank," in *WWW*. ACM, 2002, pp. 517–526.

[13] J. Leskovec and J. J. Mcauley, "Learning to discover social circles in ego networks," in *NIPS*, 2012, pp. 539–547.