MODEL-DISTRIBUTED INFERENCE IN MULTI-SOURCE EDGE NETWORKS

Pengzhen Li, Hulya Seferoglu, Erdem Koyuncu

Department of Electrical and Computer Engineering, University of Illinois Chicago

ABSTRACT

Distributed inference techniques can be broadly classified into data-distributed and model-distributed schemes. In data-distributed inference (DDI), each worker carries the entire deep neural network (DNN) model, but processes only a subset of the data. However, feeding the data to workers results in high communication costs especially when the data is large. An emerging paradigm is model-distributed inference (MDI), where each worker carries only a subset of DNN layers. In MDI, a source device that has data processes a few layers of DNN and sends the output to a neighboring device. This process ends when all layers are processed in a distributed manner. In this paper, we investigate MDI with multiple sources, *i.e.,* when more than one device has data. We design a multi-source MDI (MS-MDI), which optimizes task scheduling decisions across multiple source devices and workers. Experimental results on a real-life testbed of NVIDIA Jetson TX2 edge devices show that MS-MDI improves the inference time significantly as compared to baselines.

Index Terms— Model distribution, model distributed inference, deep neural network (DNN), distributed DNN.

1. INTRODUCTION

The traditional approach to perform deep neural network (DNN) inference in a distributed fashion is data-distributed inference (DDI). A typical DDI scenario is when an end user or edge server, *i.e.,* a source device, would like to classify its available data. Workers may correspond to other end users, edge servers, or remote cloud. The source device itself could function as one of the workers by processing some of its own data. The source partitions and transmits the available data to the multiple workers, all of which carry the same DNN model. The workers then perform inference on their received data via the available DNN. The outputs are finally sent back to the source. This strikingly simple approach, although very effective in certain cases, incurs very high communication costs especially when the input data size is large.

An alternative to DDI is model-distributed inference (MDI), which is often referred to as model parallelism. In this scenario, the DNN model itself is partitioned to blocks of layers and then distributed across multiple workers. Unlike DDI, the data is not distributed and resides at the source. Instead, given an input to the DNN, the source may process the input through the first few layers of the DNN. The resulting feature vectors (intermediate outputs of the DNN after the first few layers) are then passed to the next worker that is responsible for the next few subsequent layers. Inference is completed sequentially in the same manner by passing feature vectors to the workers responsible for processing them. After the output of the DNN is finally computed, it is transmitted back to the source. Note that the processing at the workers is done in parallel to take advantage of pipelining [1].

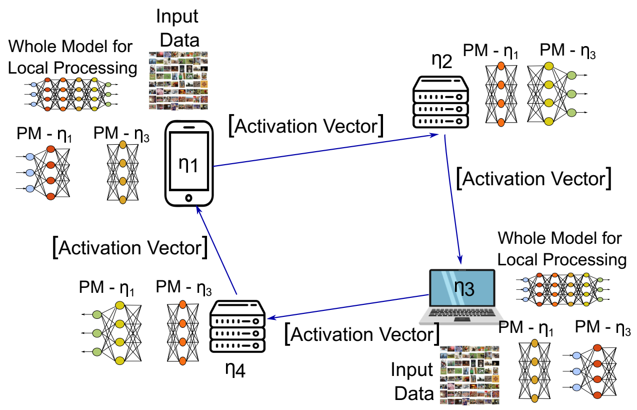


Fig. 1. Model-distributed inference with multiple sources. Workers η_1 and η_3 are source workers, while η_2 and η_4 are just workers. PM is an abbreviation for "partial model".

The performance of MDI with heterogeneous resources is investigated [1] and an adaptive layer allocation mechanism across workers is designed. It is shown that MDI significantly reduces the inference time as compared to data distributed inference when the size of data is large [1]. Despite its potential, the performance of model distributed inference (MDI) with more than one source device is an unexplored area. In this paper, we investigate MDI with multiple sources, *i.e.,* when more than one device has data. The next example illustrates the problem of multi-source MDI.

Example 1 *Let us consider a scenario in Fig. 1, where four workers are connected to each other in a circular topology. Workers η_1 and η_3 are source devices,* i.e., *they have data to process. The other devices,* i.e., *η_2 and η_4 help η_1 and η_3 to*

This work was supported in part by the Army Research Lab (ARL) under Grant W911NF-2120272 and in part by National Science Foundation (NSF) under Grants CCF-1942878, CNS-2148182, and CNS-2112471.

process their data. The source devices η_1 and η_3 may prefer to process their data locally, so they keep the whole pre-trained DNN model. All workers have partial models (PM) for all sources, i.e., *η_1 and η_3. These partial models are used for MDI. In particular, at time slot t, source device η_1 may prefer to process (i) the whole model locally for its own data, (ii) a couple of layers of DNN for its own data, or (iii) a few layers of DNN for η_3's data. These tasks should be scheduled in η_1 as well as in other workers in parallel.*

In this paper, we design a multi-source MDI (MS-MDI), which optimizes task scheduling decisions across multiple workers. Our approach builds on Network Utility Maximization (NUM) [2], where each task is associated with a queue, and task scheduling decisions are made based on queue sizes. We implement MS-MDI in a real testbed consisting of NVIDIA Jetson TX2s. The experimental results show that MS-MDI improves the inference time significantly as compared to baselines for the MobileNetV2 [3] model and the ImageNet [4] dataset.

The rest of the paper is organized as follows. In Section 2, we present the related work. We introduce our system model and provide background in Section 3. Section 4 presents our MS-MDI design. In Section 5, we provide experimental results. We draw our main conclusions in Section 6.

2. RELATED WORK

A defining characteristic of edge computing networks is the severe limitations of the constituent nodes in terms of their communication bandwidth, battery/power limitations, computing/transmission power, etc. Such limitations make centralized approaches to learning or inference infeasible. Hence, developing novel distributed learning and inference methods is crucial, especially at the edge.

An earlier work on distributed inference is a two-part DNN partition [5]. Specifically, a DNN is split into two parts, where the first few layers are placed at the end user, while the remaining layers are located at the cloud. The processing in the end user and cloud could be iterative or parallel. In iterative processing, an end user processes a few layers, and if it is confident with the output, it terminates the processing. Otherwise, feature vectors are sent to the cloud for further processing at subsequent DNN layers. In parallel processing, layers are partitioned and assigned to end user and edge server (or cloud), which operate in parallel [6] and [7].

One limitation of the aforementioned line of work on distributed inference is that they are limited to two parallel processing devices. As a result, these works may not take full advantage of edge computing systems, where there can be possibly several end users and edge servers. In fact, a general MDI system can consists of more than two workers. A model partitioning problem for MDI is formulated in [8] using dynamic programming. An adaptive and resilient layer allocation mechanism is designed in [1] to deal with the heterogeneous and time-varying nature of resources. As compared to this work, we consider multiple sources, and design a task scheduling mechanism MS-MDI.

It should be mentioned that various other methods to reduce the memory, communication, and computation footprints of DNNs have been proposed, including conditional architectures [9, 10], pruning [11, 12], quantization [13, 14], and gradient sparsification [15, 16]. Our work is complementary to these alternate techniques in the sense that the performance of MS-MDI can be further improved by incorporating one or more of these methods.

3. SYSTEM MODEL AND BACKGROUND

Setup. We consider a multi-source edge network with end users and edge servers. We name these devices as workers. The set of workers is $\mathcal{N} = \{\eta_0, \dots, \eta_{N-1}\}$, where η_n is the nth worker. We name a subset of workers as a source set, *i.e.,* $\mathcal{S} \subseteq \mathcal{N}$, where the sources in the source set collect data (images) in real time from the environment. The source devices would like to classify their data as soon as possible by exploiting their own and other workers' computing power.

Workers form a circular overlay topology [17], which naturally arises in MDI [1]. We note that topology construction, model allocation, and task scheduling decisions can be jointly optimized, but it introduces higher communication and computation cost. Therefore, we assume that overlay topology is determined using existing mechanisms [18, 19].

We consider a dynamic edge computing setup where workers join and adjourn the system anytime, *i.e.,* workers can leave the system before finishing their assigned tasks. Also, computing and communication delay of workers could be potentially heterogeneous and vary over time.

DataSet and DNN Model. Suppose that worker η_i is a source. It collects data from the environment according to a Poisson distribution with rate α_i, and A_k^i is the kth data that worker η_i collects.

We assume that the same pre-trained DNN model consisting of L layers is used by all sources. The set of layers is determined by $\mathcal{L} = \{l_1, \dots, l_L\}$, where l_i is the ith layer. The weights of the pre-trained DNN model are $\mathbf{w} = \{\mathbf{w}_i\}_{i=1}^{L}$, where $\mathbf{w}_i$ is the vector of weights corresponding to layer l_i. The number of parameters (weights) at layer i is W_i, *i.e.,* $|\mathbf{w}_i| = W_i$. The total number of parameters is $W = \sum_{i=1}^{L} W_i$, where $|\mathbf{w}| = W$.

Model-Distributed Inference. Suppose that the source node η_i wishes to process its kth data, *i.e.,* A_k^i. Let $\Lambda_n^i(k) = \{\lambda_n^{1,i}(k), \dots, \lambda_n^{|\Lambda_n^i(k)|,i}(k)\}$ with $\lambda_n^{l,i}(k) \in \mathcal{L}$ denote the set of layers that worker η_n computes for processing data A_k^i. The last layer of $\Lambda_n^i(k)$ is represented by $l_n^i(k) = \lambda_n^{|\Lambda_n^i(k)|,i}(k)$. We consider that $\Lambda_n^i(k) \cap \Lambda_m^i(k) = \emptyset, \forall n \neq m$ as workers should process independent set of layers.

The source node η_i processes all the layers in $\Lambda_1^i(k)$ for data A_k^i and determines the corresponding activation vectors.

The activation vector of the first layer *i.e.,* $a_1^i(k)$ is calculated as $a_1^i(k) = \mathbf{w}_1^i A_k^i$. The activation vectors of the next layers are calculated as $a_l^i(k) = \mathbf{w}_l^i a_{l-1}^i(k)$, $\forall l \in \Lambda_1^i(k)$ and $l \neq 1$. The activation vector of the the last layer *i.e.,* $a^i_{l_i^i(k)}(k)$ is transmitted by the source node η_i to its next hop neighbor in the circular topology, which is $\eta_{(i+1)\%N}$, where $\%$ is a modulo symbol.

Worker η_n ($n \neq i$) calculates $a_l^i(k) = \mathbf{w}_l^i a_{l-1}^i(k)$, $\forall l \in \Lambda_n^i(k)$ and sends the output of the last layer, *i.e.,* $a^i_{l_n^i(k)}(k)$ to the next worker. If $a^i_{l_n^i(k)}(k)$ is the output of the last layer, it is transmitted to the source node η_i.

The MDI process can be pipe-lined so that η_i can start processing $k+1$th data immediately after calculating $a_k^i(k)$. Thus, workers are kept busy in a setup with homogeneous resources, and layers are processed in parallel. We use adaptive and resilient model distributed inference (AR-MDI) [1] in heterogeneous setups by particularly focusing on multiple sources. Next, we will briefly explain AR-MDI.

Adaptive and Resilient Model Distributed Inference (AR-MDI) [1]. AR-MDI is a layer allocation mechanism that determines the layers $\Lambda_n^i(k)$ that should be activated at worker η_n for processing data A_k^i of source η_i. AR-MDI allocates $|\rho_n^i(k)|_{\mathcal{L}}$ layers to worker η_n for data A_k^i originated from source η_i, where $|.|_{\mathcal{L}}$ rounds $\rho_n^i(k)$ to a closest integer that is a feasible layer allocation according to DNN layer set $\mathcal{L}$. AR-MDI determines $\rho_n^i(k)$ as

$$\rho_n^i(k) = W \frac{1/\gamma_n^i(k)}{\sum_{n=0}^{N-1} 1 \, \gamma_n^i(k)}, \qquad (1)$$

where W is the total number of parameters in the DNN model, $\gamma_n^i(k)$ is the per parameter computing delay. AR-MDI performs layer allocation in a decentralized way as each worker can determine their share of layers by calculating $|\rho_n^i(k)|_{\mathcal{L}}$. The per parameter computing delay $\gamma_n^i(k)$ can be measured by each worker and shared with other workers.

4. MULTI-SOURCE MDI (MS-MDI)

In this section, we present our Multi-Source Model Distributed Inference (MS-MDI) design. Our approach builds on Network Utility Maximization (NUM) [2] and its solution, where each task is associated with a queue, and task scheduling decisions are made based on queue sizes. Next, we discuss the construction of queues.

Queue Construction. Each source worker η_i constructs a queue q_l^i, which stores data that are supposed to be processed locally at worker η_i. In other words, the data in q_l^i is not processed in a distributed manner, but locally at η_i. Each worker (either a source or not) η_n, $n \in \mathcal{N}$ constructs $|\mathcal{S}|$ queues q_n^i for each source node $i \in \mathcal{S}$. The queue q_n^i stores data or activation vectors that node η_n is supposed to process for source node η_i in a distributed manner. Let us consider the example topology in Fig. 1 again in Fig. 2 and focus on η_1. The

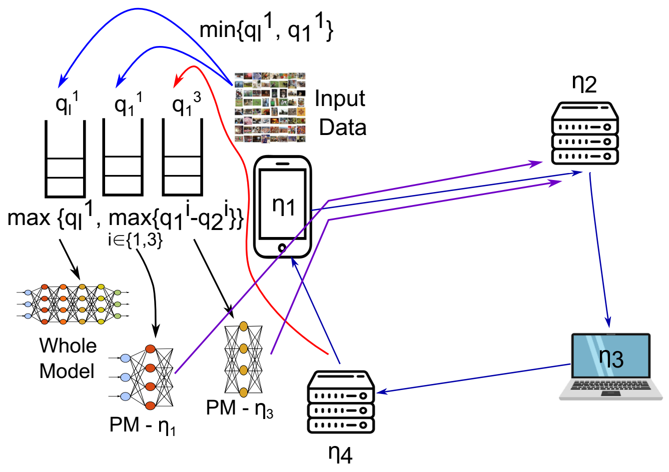


Fig. 2. Queue constructions and task scheduling decisions in an example topology of Fig. 1. We focus on the operations of worker 1, *i.e.,* η_1, in this figure.

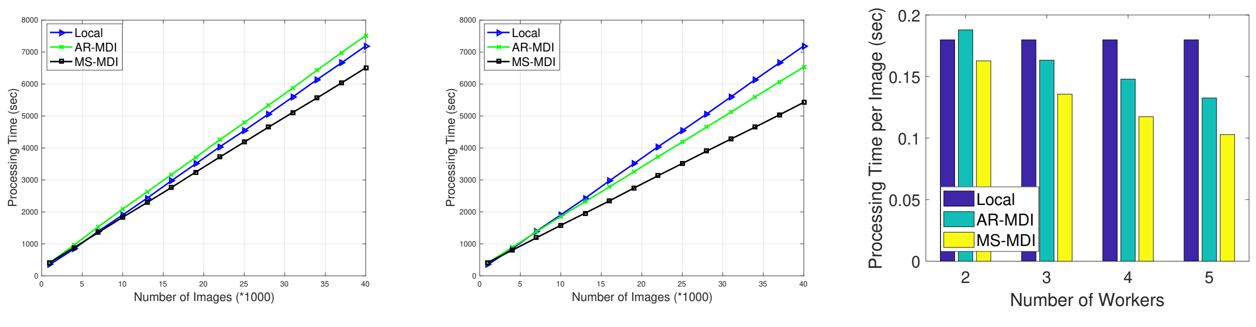
source device η_1 constructs three queues; (i) q_l^1 for processing its own data locally, (ii) q_1^1 for processing its own data via MDI, and (iii) q_1^3 for processing η_3's data via MDI.

Data arrival in sources. In source worker η_i, if data is collected or arrives according to Poisson distribution, such data is inserted in a reservoir queue μ_i. Then, at time slot t, fixed amount of data is taken from μ_i, and inserted in a queue that satisfies $\min\{q_l^i(t), q_i^i(t)\}$. We note that $q_l^i(t)$ represents local processing (inference) queue at time t, while $q_i^i(t)$ corresponds to the distributed processing queue. Our design of MS-MDI selects the smallest queue among $q_l^i(t)$ and $q_i^i(t)$ for inserting new data arrivals, because the smallest queue likely corresponds to the policy, *i.e.,* either local or MDI, that processes data faster. For example, η_1 in Fig. 2 checks $\min\{q_l^1, q_1^1\}$ and inserts the new arrivals to the smallest queue.

Selection of queues for inference. At slot t, each node η_n decides which data should be processed according to

$$\max\{q_l^n(t), \max_{j \in \mathcal{S}}\{q_n^j(t) - q_{(n+1)\%N}^j(t)\}\}, \qquad (2)$$

where $q_n^j(t) = -M$ if $n \notin \mathcal{S}$ and M is a large positive constant. If $q_l^n(t)$ is the maximum term in (2), data from $q_l^n(t)$ is taken and processed locally at node n. Otherwise, *i.e.,* if $\max_{\forall j \in \mathcal{S}}\{q_n^j(t) - q_{(n+1)\%N}^j(t)\}$ is the maximum term, $z = \arg\max_{\forall j \in \mathcal{S}}\{q_n^j(t) - q_{(n+1)\%N}^j(t)\}$ is determined. Then, data from $q_n^z(t)$ is processed at worker η_n according to MDI. The output of the process, *i.e.,* the activation vector of the last layer $a^z_{l_n^z}(k)$ is sent to the next node $\eta_{(n+1)\%N}$. Let us consider η_1 in Fig. 2 again. If the maximum term of $\max\{q_l^1, \max_{i \in 1,3}\{q_1^i - q_2^i\}\}$ is q_l^1, η_1's data is processed locally at η_1. If the maximum term is $q_1^i - q_2^i$, $i \in \{1,3\}$ then η_i's data is processed partially, and the activation vector of the last layer is transmitted to η_2. We note that activation vectors coming from η_4 are an input to q_1^3. The rationale



(a) Two-worker (b) Three-worker (c) Increasing number of workers

Fig. 3. Inference/processing time versus the number of images in a (a) two worker setup, and (b) three worker setup. (c) Inference/processing time versus the number of workers. Two out of all workers are source workers in all experiments.

behind selecting the largest queue is to ramp up the speed of data processing for large queues.

5. EXPERIMENTAL RESULTS

In this section, we evaluate the performance of our algorithm MS-MDI in a real life testbed consisting of NVIDIA Jetson TX2 devices. First, we describe the testbed architecture, DNN models, and the datasets. We then provide the corresponding experimental results.

Datasets, DNN Models and Testbed Description. We utilize the ImageNet [4] dataset with $1,000$ classes for image classification. The dataset contains more than 1 million training, $50,000$ validation, and $100,000$ test images. We downsample the dataset images to a fixed size of 256×256 as the input sizes are larger and have variable variable dimensions. We use the MobileNetV2 [3] as the inference model.

Our testbed consists of NVIDIA Jetson TX2 computing cards that contain GPUs. Wi-Fi link provides the connections between the Jetsons. An access point in the link layer provides a connected topology. We implement this topology as a proof-of-concept: MS-MDI operates over a circular topology in the overlay and works with any lower-layer topology.

We compare our algorithm MS-MDI with baselines: (i) AR-MDI [1], where adaptive and resilient layer allocation is suggested against heterogeneous and time-varying resources; and (ii) Local, where the source node has the complete DNN model and prefers to process all the data by itself.

Results. We first consider a setup with two workers η_1 and η_2. Each worker is a source worker, *i.e.,* they have their own data to process. The first and second workers receive data with rates $\alpha_1 = 5$ images/sec and $\alpha_2 = 10$ images/sec, respectively. The workers η_1 and η_2 receive 10K and 30K data in total. Fig. 3(a) shows the inference time (which is referred as processing time) versus the number of images. As seen, MS-MDI improves over both Local and AR-MDI by 11% and 16%, respectively. Local performs better than AR-MDI as both workers have their own data to process, so Local processes data more efficiently while AR-MDI wastes time for communication. MS-MDI performs the best thanks to switching between Local and AR-MDI depending on queue occupancy.

Next, we consider a setup with three workers. η_1 and η_2 are the source workers with the same data arrival rates as in Fig. 3(a). The third worker η_3 is not a source, *i.e.,* just a worker. Fig. 3(b) shows that AR-MDI performs better than Local as AR-MDI can exploit the computing resources in η_3, while η_3 stays idle in Local. MS-MDI improves over AR-MDI and Local by 20% and %32, which is significant. We see that the improvement of MS-MDI is more in three worker setup as compared to the two worker thanks to adaptive switching between Local and AR-MDI when needed, *i.e.,* depending on queue sizes.

We also consider a setup of increasing number of workers. Two sources out of all the workers are selected as source workers, and the arrival rates are the same as in Fig. 3(a) setup. Fig. 3(c) presents the processing time per image, which is averaged over $40K$ images versus the number of workers. As seen, the improvement of MS-MDI over both AR-MDI and Local increases with the increasing number of workers. The improvements of MS-MDI as compared to AR-MDI and Local for five-worker setup are 29% and 74%, respectively, while the improvements are 16% and 11% for the two-worker setup. MS-MDI takes the advantage of increasing number of workers, so its improvement as compared to Local increases significantly. MS-MDI also improves as compared to AR-MDI as it switches to Local when there is any communication or computation bottleneck in the system.

6. CONCLUSION

In this paper, we investigated MDI with multiple sources, *i.e.,* when more than one device has data. We designed a multi-source MDI (MS-MDI), which optimizes task scheduling decisions across multiple source devices and workers. Our approach builds on NUM formulation and its solution, where each task is associated with a queue, and task scheduling decisions are made based on queue sizes. We implemented MS-MDI in a real testbed consisting of NVIDIA Jetson TX2s and showed that MS-MDI improves the inference time significantly as compared to baselines. m, ““

7. REFERENCES

[1] Pengzhen Li, Erdem Koyuncu, and Hulya Seferoglu, "Adaptive and resilient model-distributed inference in edge computing systems," 2022.

[2] Michael J Neely, "Stochastic network optimization with application to communication and queueing systems," *Synthesis Lectures on Communication Networks*, vol. 3, no. 1, pp. 1–211, 2010.

[3] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 4510–4520.

[4] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 2009, pp. 248–255.

[5] Surat Teerapittayanon, Bradley McDanel, and Hsiang-Tsung Kung, "Distributed deep neural networks over the cloud, the edge and end devices," in *2017 IEEE 37th international conference on distributed computing systems (ICDCS)*. IEEE, 2017, pp. 328–339.

[6] Yiping Kang, Johann Hauswald, Cao Gao, Austin Rovinski, Trevor Mudge, Jason Mars, and Lingjia Tang, "Neurosurgeon: Collaborative intelligence between the cloud and mobile edge," *ACM SIGARCH Computer Architecture News*, vol. 45, no. 1, pp. 615–629, 2017.

[7] Amir Erfan Eshratifar, Mohammad Saeed Abrishami, and Massoud Pedram, "Jointdnn: an efficient training and inference engine for intelligent mobile cloud computing services," *IEEE Transactions on Mobile Computing*, vol. 20, no. 2, pp. 565–576, 2019.

[8] Yang Hu, Connor Imes, Xuanang Zhao, Souvik Kundu, Peter A Beerel, Stephen P Crago, and John Paul N Walters, "Pipeline parallelism for inference on heterogeneous edge computing," *arXiv preprint arXiv:2110.14895*, 2021.

[9] Yizeng Han, Gao Huang, Shiji Song, Le Yang, Honghui Wang, and Yulin Wang, "Dynamic neural networks: A survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021.

[10] Alperen Gormez, Venkat Dasari, and Erdem Koyuncu, "E^{2}CM: Early exit via class means for efficient supervised and unsupervised learning," in *International Joint Conference on Neural Networks (IJCNN)*, July 2022.

[11] Jiayi Liu, Samarth Tripathi, Unmesh Kurup, and Mohak Shah, "Pruning algorithms to accelerate convolutional neural networks for edge applications: A survey," *arXiv preprint arXiv:2005.04275*, 2020.

[12] Alperen Gormez and Erdem Koyuncu, "Pruning early exit networks," in *Workshop on Sparsity in Neural Networks*, July 2022.

[13] Wenlin Chen, James Wilson, Stephen Tyree, Kilian Weinberger, and Yixin Chen, "Compressing neural networks with the hashing trick," in *International conference on machine learning*. PMLR, 2015, pp. 2285–2294.

[14] Yunchao Gong, Liu Liu, Ming Yang, and Lubomir Bourdev, "Compressing deep convolutional networks using vector quantization," *arXiv preprint arXiv:1412.6115*, 2014.

[15] Jianqiao Wangni, Jialei Wang, Ji Liu, and Tong Zhang, "Gradient sparsification for communication-efficient distributed optimization," in *Advances in Neural Information Processing Systems*, 2018, pp. 1299–1309.

[16] Dan Alistarh, Torsten Hoefler, Mikael Johansson, Nikola Konstantinov, Sarit Khirirat, and Cédric Renggli, "The convergence of sparsified gradient methods," in *Advances in Neural Information Processing Systems*, 2018, pp. 5973–5983.

[17] Ion Stoica, Robert Morris, David Liben-Nowell, David R Karger, M Frans Kaashoek, Frank Dabek, and Hari Balakrishnan, "Chord: a scalable peer-to-peer lookup protocol for internet applications," *IEEE/ACM Trans. on networking*, vol. 11, no. 1, pp. 17–32, 2003.

[18] Quang Le-Dang, Jennifer McManis, and Gabriel-Miro Muntean, "Location-aware chord-based overlay for wireless mesh networks," *IEEE transactions on vehicular technology*, vol. 63, no. 3, pp. 1378–1387, 2013.

[19] Simone Burresi, Claudia Canali, M Elena Renda, and Paolo Santi, "Meshchord: A location-aware, cross-layer specialization of chord for wireless mesh networks (concise contribution)," in *2008 Sixth Annual IEEE International Conference on Pervasive Computing and Communications (PerCom)*. IEEE, 2008, pp. 206–212.