MoDNN: Local Distributed Mobile Computing System for Deep Neural Network

Jiachen Mao[†], Xiang Chen[‡], Kent W. Nixon[†], Christopher Krieger[II], and Yiran Chen[†]
[†]University of Pittsburgh, USA; [‡]George Mason University, USA; [II]University of Maryland, USA
[†]{jim35, kwn2, yic52}@pitt.edu; [‡]xchen26@gmu.edu; [II]Krieger@lps.umd.edu

***Abstract*—Although Deep Neural Networks (DNN) are ubiquitously utilized in many applications, it is generally difficult to deploy DNNs on resource-constrained devices, e.g., mobile platforms. Some existing attempts mainly focus on client-server computing paradigm or DNN model compression, which require either infrastructure supports or special training phases, respectively. In this work, we propose MoDNN – a local distributed mobile computing system for DNN applications. MoDNN can partition already trained DNN models onto several mobile devices to accelerate DNN computations by alleviating device-level computing cost and memory usage. Two model partition schemes are also designed to minimize non-parallel data delivery time, including both wakeup time and transmission time. Experimental results show that when the number of worker nodes increases from 2 to 4, MoDNN can accelerate the DNN computation by 2.17-4.28×. Besides the parallel execution, the performance speedup also partially comes from the reduction of the data delivery time, e.g., 30.02% w.r.t. conventional 2D-grids partition.**

I. Introduction

The ever-increasing bandwidth of mobile networks inspired rapid growth of multimedia interactive applications on mobile devices, which involve intensive object recognition and classification tasks. Deep Neural Networks (DNN) have been widely used in performing these tasks due to their high accuracy and self-adaptiveness property. However, execution of DNN incurs considerably resources. A representative example is VGG [1], which demonstrates state-of-the-art performance in ImageNet Large Scale Visual Recognition Challenge 2014 (ILSVRC14). VGG has 15M neurons, 144M parameters, and 3.4B connections. When deployed on a mobile device, VGG spends approximately 16 seconds to complete the identification procedure for one image, which is intolerable in practical.

The gap between large computing workloads of DNN and limited computing resources of mobile devices adversely impact user experience and inspired some research works to fill the gap. For example, client-server paradigm is a straightforward solution to efficiently offload the high computing cost to external infrastructure: In [2], Hauswald *et al.* proposed a data offloading scheme in a pipelined machine learning structure; In [3], Li *et al.* established an efficient distributed parameter server framework for DNN training. In addition, many studies have been performed to reduce the computing workloads of DNN, such as model compression: In [4], Han *et al.* deeply compressed the DNN models using a three stage pipeline: pruning, trained quantization, and Huffman coding; In [5], Chen *et al.* introduced a low-cost hash function to group weights into hash buckets for parameter sharing purpose.

We note that there is an important scenario that has not been fully explored yet in all the previous works, say, running DNN on a local distributed mobile computing system. Compared to client-server paradigm where a single mobile device is supported by external infrastructure, local distributed mobile computing systems offer several important advantages, including more local computing resources, higher privacy, less dependency on network bandwidth, etc.

In this work, we propose MoDNN – a local distributed mobile computing system for DNN that can work over a Wireless Local Area Network (WLAN). MoDNN can significantly speedup the computation of DNN by introducing execution parallelism among multiple mobile devices. Our contributions include: 1) We investigate the method of building a computing cluster in WLAN with multiple authorized WiFi-enabled mobile devices for DNN computations. The mobile device that carries the testing data (e.g., image) acts as the Group Owner (GO) and the other devices act as the worker nodes; 2) We propose two partition schemes to minimize the data delivery time between the mobile devices based on the unique properties of two types of DNN layers (convolutional layers and fully-connected layers) and various mobile computing abilities; 3) We employ a middleware on each mobile device in the computing cluster to schedule the whole execution process.

To the best of our knowledge, this is the first work that utilizes heterogeneous mobile devices in WLAN as computing resources for DNN with several innovations in execution parallelism enhancement and data transmission. Experimental results show that when the number of worker nodes increases from 2 to 4, MoDNN can speedup the DNN computation by 2.17-4.28×, thanks to the achieved high execution parallelism and the significantly reduced data delivery time.

II. Preliminary

A. Opportunistic Mobile Network

Benefiting from high transmission bandwidth and robust protocol, WLAN serves as an ideal environment to create opportunistic mobile networks for cluster computing. The topology of opportunistic mobile networks allows mobile devices to communicate over a WLAN. Such a proximity-based communication characteristic also enables high data transmission bandwidth between the mobile devices. In this work, we adopt opportunistic mobile networks as our network foundation and implement MoDNN using WiFi Direct [6], which allows for a transfer speed of up to 250 Mbps with an energy efficiency better than a cellular network.

978-3-9815370-8-6/17/$31.00 ©2017 IEEE

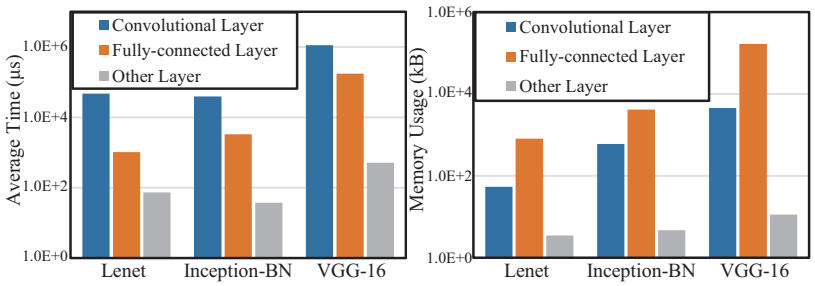


Fig. 1. Average computing time and memory usage of the layers in DNNs.

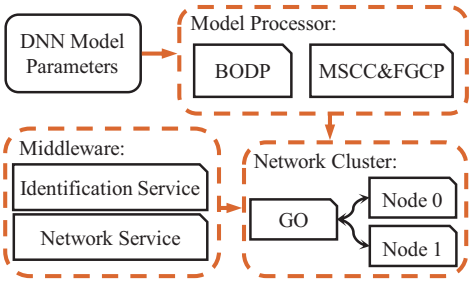


Fig. 2. System overview of MoDNN.

B. Distributed Programming Model

MapReduce is a programming model for simplifying parallel data processing in distributed systems [7]. Its effectiveness has been proven in many machine learning applications through maximizing the usage of computing resources of the nodes in a computing cluster [8]. There are two primitives in MapReduce applications: Map and Reduce. The Map procedure partitions a task to pieces that can be executed in parallel while the Reduce procedure merges the intermediate data from the Map procedure. In this work, we use these two primitives in our single-GO multiple-clients network topology to describe the data transmissions between DNN layers.

C. Properties of Two Types of Layers in DNN

In a DNN, the most computing-intensive and memory-intensive layers are Convolutional Layers (CL) and Fully-connected Layers (FL). Fig. 1 depicts our measurement results of the computing time and the memory usage of different network layers from three popular DNN models running on smartphones – $Lenet$ [9], $Inception\text{-}BN$ [10], and VGG [1]. Although these three models have very different scales, two common properties are observed in all three measurements:

- CLs contribute to the majority (e.g., 86.5%∼97.8%) of the total computing time;
- FLs contribute to more than 87.1% of the total memory that are used to store the parameters of the DNN model.

Hence, in this work, we will particularly investigate the partition schemes of these two types of layers in the DNN.

Sparsifying FLs is a promising technique that can effectively reduce the associated computing cost [11], [4]. For example, the connectivity of FLs in VGG-16 can be reduced by 95.6% without incurring any accuracy loss [4]. As we shall show in Section III, model sparsity in DNNs offer a great opportunity for MoDNN to reduce the data delivery time by optimizing the network weight partition of sparse FLs.

III. System Framework of MoDNN

Fig. 2 presents an overview of MoDNN which includes three main components: 1) A local distributed network cluster formed by GO and multiple worker nodes; 2) A model processor which partitions the DNN model onto the worker nodes; and 3) A middleware that performs data delivery and identification services of the DNN.

We note that the computing cost of CLs is primarily dependent on its input size. Hence, we introduce a Biased One-Dimensional Partition (BODP) scheme to partition the CLs. On the contrary, the memory usage of FLs is mainly decided by the number of weights in the layer. As a result, a weight partition scheme that consists of Modified Spectral Co-Clustering (MSCC) and Fine-Grain Cross Partition (FGCP) is introduced specifically for sparse FLs. It is worth noting that here the DNN model partition only need to be performed once in the application once the DNN is trained. Thus, the partition cost can be amortized over the execution of the system as long as the trained DNN keeps the same.

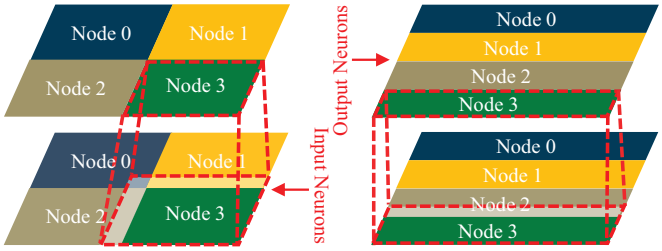
A. Definition of Terminologies and Variables

We define the terminologies and variables that are referred to in following sections as follows:

- **Total Worker Nodes (k):** Total number of the available worker nodes within the computing cluster;
- **Workload ($W_{[i]}$):** The workload assigned to node i;
- **Estimated Time ($ET_{[i]}$):** Estimated time for node i to execute workload $W_{[i]}$ plus data delivery time;
- **Computing Ability ($CA_{[i]}$):** The normalized performance of node i, e.g., FLOPS;
- **SpMV Time ($SPT_{[i]}(n)$):** Time for node i to do Sparse Matrix-Vector multiplication (SpMV) in which the matrix is represented by a linked list of size n;
- **GEMV Time ($GET_{[i]}(r,c)$):** Time for node i to perform General Matrix-Vector multiplication (GEMV) in which the matrix is represented by $r \times c$ array;
- **Sparsity Threshold ($Thld_{[i]}(r,c)$):** Sparsity threshold of node i that achieves equivalent computing time of the $r \times c$ matrix using SpMV and GEMV;
- **Data Delivery Time:** Data delivery time denotes the total time consumption for the data being transmitted between nodes. Data delivery time includes two parts: wakeup time and transmission time. Wakeup time represents the amount of time for the head of the data traveling from the sender to the receiver and transmission time denotes the amount of time for the receiver receiving from the first bit to the last bit of the data.

B. Network Establishment and Setup

1) Network Topology of MoDNN: In our proposed MapReduce-based topology, each worker node is mapped with a part of the layer inputs and the outputs are reduced back to the GO, which generates the inputs of the new layer in the following map procedure.



(a) Neurons in 2D-grids of 4 nodes. (b) BODP for 4 worker nodes.

Fig. 3. Two partition schemes for CLs.

In order to form a cluster, the GO enables its WiFi module to act as an AP that is prepared for responding to potential worker nodes. In the mean time, the available worker nodes with extra computing resources are searching for the GO in an opportunistic mobile network domain.

2) Connection and Registration: In our design, after getting the permission of the user, a mobile device will be permanently trusted by the GO and automatically connected to the group when it is within the reachable WiFi range. Once connected, in addition to the device IP address, performance-related meta data are also sent to the GO for later utilization in partition schemes. The meta data includes the previously defined variables and functions like $CA_{[i]}$, $SPT_{[i]}(n)$, $GET_{[i]}(r,c)$, $Thld_{[i]}(r,c)$, etc. The motivations to define and generate $SPT_{[i]}(n)$, $GET_{[i]}(r,c)$ and $Thld_{[i]}(r,c)$ will be discussed in detail in Section III-D1.

C. Input Partition for CLs

Conventional partition schemes of CLs on other platforms usually maintain a structural symmetry for the layer inputs. For example, in [12], Coates *et al.* arranged a GPU cluster into 2D-grids and partitioned the input neurons along the two-dimensional space, as shown in Fig. 3(a). However, such a two-dimensional partition may not be suitable for the proposed local distributed mobile computing system.

Unlike in the GPU cluster, the wakeup time, rather than the transmission time, dominates the data delivery time in MoDNN. It is because of the Opportunistic Power Save Protocol that support the sleep mode of the clients: If a mobile device has not been used for a certain time period, it will turn off its radio modules automatically [13]. Turning on the radio modules and establish the transmission channel takes a time period significantly longer than the data transmission itself.

In MoDNN, BODP is proposed to partition the input neurons along the longer edge of the input matrix according to the computing abilities $CA_{[i]}$ of individual node, as illustrated in Fig. 3(b). There are two set of neurons in the figures: input neurons and output neurons of a CL. Using $node3$ as an example: the input of $node3$ overlaps all the other three nodes in the 2D partition in Fig. 3(a), while the input of $node3$ in Fig. 3(b) only overlaps with that of $node2$. Note that only the overlapped parts of the layer inputs need to be transferred during the computation.

Since the wakeup time in MoDNN is greatly impacted by the number of the established transmission channels, reducing the number of the neighbor nodes from 4 (in conventional 2D partition) to 2 (in BODP) will effectively minimize the associated high propagation delay. The procedure of BODP is detailed in Algorithm I and more analysis on the effectiveness of BODP can be found in Section IV-B.

Algorithm 1: Input Partition for CL (BODP)

Input: Original matrix A of size $r \times c$, total available worker nodes k, filter size of $f \times f$ and computing ability $CA_{[i]}$ of each worker node
Initialization: Set $W_{[i]} = 0$, $CA_{[0]} = 0$
1: **do**
2: **if** $r > c$
3: A is divided by rows: $m = r$
4: **else**
5: A is divided by columns: $m = c$
6: Define the initial line index of node i:

$$Init(i) = \lfloor \frac{\sum_{k=0}^{i-1} CA_{[k]}}{\sum_{k=0}^{N} CA_{[k]}} \times (m - (f-1)) \rfloor$$

7: The one-dimensional index of A is bounded by:

$$W_{[i]} = A(Init(i-1), Init(i) + (f-1)],\ i = 1, 2...k$$

8: **end**
Output: Workload partition $W_{[i]}$, $i = 1, 2...k$

D. Weight Partition for Sparse FLs

The partition scheme of FLs in MoDNN targets the state-of-the-art sparse FLs. Because of the comparatively short execution time of FLs, mobile devices will keep the wireless radio in an active state and the transmission time dominates the data delivery time. The object of the proposed partition scheme is to reduce the size of the data to be transmitted for the reduction of the transmission time.

1) Hybrid Matrix Representation in MoDNN: There are two approaches to compute matrix-vector multiplication, which is the main operation in DNN: General Matrix-Vector multiplication (GEMV) and Sparse Matrix-Vector multiplication (SpMV). GEMV is usually used to compute a dense matrix which often uses arrays to represent the data while SpMV is effective in computing a sparse matrix that can be efficiently stored in a linked-list, as illustrated in Fig 4. The selection of the appropriate data representation can be decided by comparing the target matrix sparsity with $Thld_{[i]}(r,c)$, which is defined by:

$$Thld_{[i]}(r,c) = \frac{SPT_{[i]}^{-1}(GET_{[i]}(r,c))}{r \times c}. \quad (1)$$

Here $SPT_{[i]}(n)$ and $GET_{[i]}(r,c)$ are the time spent on the computation of the matrix-vector multiplication using SpMV and GEMV, respectively. They can be obtained from real measurements on the mobile devices via a linear regression

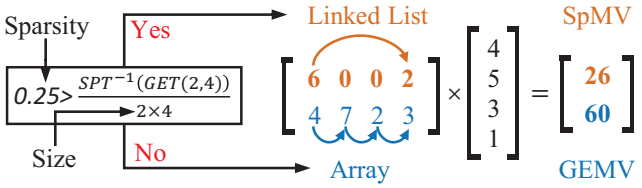
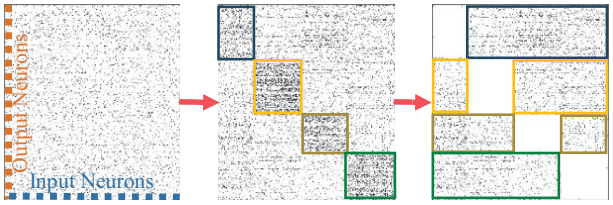


Fig. 4. Hybrid matrix representation for SpMV and GEMV.



(a) Original. (b) Clustered (MSCC). (c) Outliers (FGCP).

Fig. 5. Two-stage processing: MSCC&FGCP.

method. When the sparsity of the matrix is larger than $Thld_{[i]}(r, c)$, SpMV will be used for the computation; otherwise, GEMV will be applied.

2) Modified Spectral Co-Clustering (MSCC): We note that GEMV is more computationally efficient per matrix element than SpMV due to its higher computing parallelism. Hence, it will be beneficial to partition the weight matrices onto the worker nodes in a dense structure. In the weight partition scheme of FLs in MoDNN, a clustering algorithm is leveraged to group the nonzero weights into several clusters and minimize the number of the nonzero weights outside the clusters. If we consider the weight matrix as an undirected graph, generating k clusters with minimal connections between them is a NP hard problem [14]. In MoDNN, we use spectral clustering technique to find the solution heuristically.

Spectral clustering technique is widely used in graph partition problems, aiming at minimizing between-cluster similarities [15]. In MoDNN, the sparse FLs are treated as undirected graphs where the graph vertices represent the input and output neurons and the edges represent the network weights. Hence, we redefine the similarity in spectral clustering technique as the number of between-clusters connections. However, traditional spectral clustering technique works only on a matrix with the same row and column size, which greatly limits its applicability and scalability in DNN computations. Therefore, spectral co-clustering algorithm is introduced to address this drawback by normalizing the original connection matrix A to A_{norm} and performing Singular Value Decomposition (SVD) on A_{norm} [16]. Here the elements of weight matrix A are binary where '1' represents a connection between two neurons and '0' otherwise. As illustrated in Algorithm 2, this algorithm converts r rows and c columns of the original matrix A to a matrix Z of $r + c$ rows using the results generated in obtaining A_{norm}. Each column of Z is an eigenvector of A so that we can cluster matrix together based on the rows of Z. Then, we apply appropriate data structure to each cluster based on their sparsity for computing time reduction. We name this clustering procedure as modified spectral co-clustering (MSCC). A rationale of the proposed MSCC is depicted in Fig. 5(a) and 5(b). After applying MSCC, k dense clusters are generated and the corresponding input neurons are transmitted to the assigned worker nodes for parallel executions.

3) Fine-Grain Cross Partition (FGCP): Spectral co-clustering focuses on reducing only the external connectivity between the clusters without considering the internal cluster density. To solve this problem, we propose FGCP to partition

Algorithm 2: Weight Partition for Sparse FL (MSCC & FGCP)

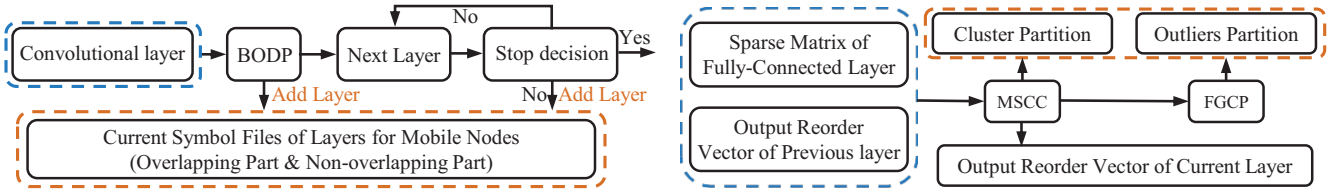
Input: Weight matrix A of size $r \times c$ ($r >= c$), total worker nodes k
Initialization: $W_{[i]} = 0, i = 0, 1, ...k$ and $ET_{[i]} = Initial\_ET(i)$, $i = 1, 2, ...k$, which denotes the transmission overhead of worker nodes
1: **do**
2: Normalize A to A_{norm}: $A_{norm} = R^{-\frac{1}{2}} A C^{-\frac{1}{2}}$, where R is diagonal matrix with entry i equal to $\sum_j A_{ij}$, C is diagonal matrix with entry j equal to $\sum_i A_{ij}$
3: Apply SVD to A_{norm}: $A_{norm} = U \Sigma V$, the columns of U,V are derived as left and right singular vectors
4: Compute matrix $Z = \begin{bmatrix} R^{-\frac{1}{2}} U \\ C^{-\frac{1}{2}} V \end{bmatrix}$, the size of U, V are given by $L = \lceil \log_2 k \rceil$, hence $U = [u_2, u_3 ... u_{L+1}]$, $V = [v_2, v_3 ... v_{L+1}]$
5: Cluster the rows in Z by K-means into clusters $C_{[1]} ... C_{[k]}$, the first r results in each cluster C represents the clustering of rows in A while the next c designates the clustering of columns
6: **if** $sparsity(C_{[i]}) > Thld_{[i]}(size(C_{[i]}))$, $i = 1, 2, ...k$
7: $W_{[i]}$ = array structure of $C_{[i]}$
8: $ET_{[i]} = GET_{[i]}(row(C_{[i]}), column(C_{[i]}))$
9: **else**
10: $W_{[i]}$ = linked list structure of $C_{[i]}$
11: $ET_{[i]} = SPT_{[i]}(non\_zeros(C_{[i]}))$
12: Get outliers for each node: $O_{[i]} = A_{sub[i]} - C_{[i]}$, $A_{sub[i]}$ is the sub-matrix of A with the same row index as C_i, $i = 1, 2, ...k$
13: Apply steps 6-11 to $O_{[i]}$ and update $W_{[i]}$ and $ET_{[i]}$
14: **do**
15: Transfer the column with minimal nonzeros of $O_{[s]}$ from $W_{[s]}$ to $W_{[0]}$, where $s = \text{argmax}_i\, f(i) := ET_{[i]}$
16: Recalculate $ET_{[i]}$ based on $W_{[i]}$, $i = 0, s$
17: **while** $ET_{[0]} < Max(ET_{[i]}), i = 1, 2, ...k$
18: **end**
Output: Workload partition $W_{[i]}$ $i = 0, 1, ...k$, where $i = 0$ represents GO

the remaining outliers in the weight matrix after MSCC to balance the workloads between the GO and the worker nodes. The basic idea here is to identify the sets of the weights with minimal number of nonzero elements and keep them computed on the GO rather than sending to the worker nodes to avoid the high cost introduced by the long data delivery time .

For the sparse outlier matrix shown in Fig. 5(c), for example, since the number of its columns is smaller than its rows, FGCP initially assigns the elements on the same rows where the cluster $C_{[i]}$ (obtained in MSCC) resides to node i. Then FGCP iteratively finds the worker node i with the maximum $ET_{[i]}$ and offloads the initially assigned weights on the same column in the outlier matrix with the minimal number of non-zero elements from the worker node i to the GO. In addition, FGCP needs to consider the discrepancy of execution time between the GO and the worker nodes during the offloading process, especially the data delivery time on the network between the worker node x and the GO, which can be conceptually formulated by:

$$Initial\_ET(x) = \frac{(\sum_{i=0}^{x} C_{column[i]} + \sum_{i=x}^{k} C_{row[i]})}{TPT}, \quad (2)$$

where TPT is the mobile network throughput; $\sum_{i=0}^{x} C_{column[i]}$ and $\sum_{i=x}^{k} C_{row[i]}$ describe the total non-overlapping data size of the input and output neurons to be transmitted during the execution. Obviously, the more



(a) Processing flow of convolutional-related layers.

(b) Processing flow of sparse FLs.

Fig. 6. Two processing flows in model processor of MoDNN.

sparse the outlier matrix is, the more elements can be possibly offloaded to the GO to balance the workload.

E. Processing Flow of DNN

Fig. 6 depicts the processing flows of CLs and FLs in MoDNN and how the two parts are integrated. Given a trained DNN, the model processor scans each layer and identify their type. If a CL is detected, the layer's input will be partitioned by BODP into small pieces, which are then combined with the subsequent non-overlapping layer structures e.g., ReLu layers, pooling layers, normalization layers, etc. for computation. If a sparse fully-connected layer is detected, MSCC and FGCP will be applied in sequence to assign the workloads to the worker nodes in clusters and the workloads for outliers, respectively, in order to achieve the minimum total execution time.

IV. Experiments

A. Environment Setup and Testbench Selection

The implementation of MoDNN is based on MXNet, which is a deep learning framework developed by the Distributed Machine Learning Community (DMLC) team for desktop platform using C++. We modified and recompiled the MXNet libraries so that it can support Android systems with ARM architecture through JAVA Native Interface (JNI) [17]. We adopt a pre-trained DNN model from ImageNet database – VGG-16 [1], as the testbench in our experiments. VGG is a popular and clear-in-structure Convolutional Neural Network (CNN) model that includes all mainstream layer types so that the significance of each component of MoDNN can be distinctly evaluated. In our experiments, VGG are executed locally or distributed to different numbers of worker nodes by MoDNN; the adopted mobile devices are LG Nexus 5 running Android 4.4.2 with a 2.28 GHz processor and 2GB RAM. The experiment setup is depicted in Fig. 7.

Fig. 8 presents the characterized results of $SPT_{[i]}(n)$ and $GET_{[i]}(r,c)$, which are two important parameters used in the partition schemes of MoDNN. Linked list and array structures are used in characterizing $SPT_{[i]}(n)$ and $GET_{[i]}(r,c)$, respectively. Here x-axis denotes the amount of computations, i.e., n non-zeros for $SPT_{[i]}(n)$ and $r \times c$ matrix for $GET_{[i]}(r,c)$, respectively. The results show that the calculation time of the worker nodes is proportional to the calculation number. For the same workload, SpMV is much slower than GEMV. Hence, we set $Thld_{[i]}(r,c)$ to 15.8% in our scheme. The measured average WLAN wakeup time and transmission throughput are $54.7ms$ and $43.8Mbps$, respectively.

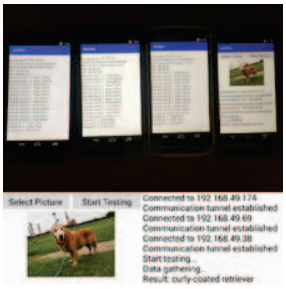


Fig. 7. Experimental setup.

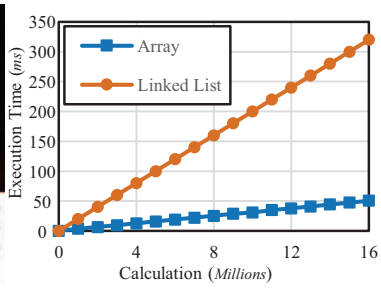


Fig. 8. Performance character. of Nexus 5.

B. Data Delivery Time Evaluation of BODP

The bars in Fig. 9 show the computing times of 13 CLs in VGG-16 during testing phase, excluding the data delivery time. The results of running locally and on 2, 3, and 4 worker nodes in MoDNN are depicted. For comparison purpose, the results of using conventional 2D-grids partition scheme for 4 worker nodes is also included in the figure. When the number of the worker nodes increases, the execution time of each CL keeps reducing, proving the effectiveness of MoDNN in parallel computing. The results of BODP with 4 worker nodes and 2D-grids partition are very close, implying little impact of the input shapes of the CLs on the computing time. As also illustrated by the dot lines in Fig. 9, compared to 2D-grids partition, BODP also slightly increases the average data transmission size of each CL from 41048 $bytes$ to 59856 $bytes$ and hence, increases the average transmission time from $7.15ms$ to $10.43ms$. Nonetheless, when taking into account that the total wakeup time contribute to approximately 30% of the total data delivery time in each data sharing procedure, BODP still achieves shorter total data delivery time than 2D-grids partition for 4 worker nodes as less transmission channels need be established.

C. Transmission Size Evaluation of MSCC & FGCP

In order to evaluate the effectiveness of MSCC and FGCP on large-scale, sparse FLs, the FLs in VGG-16 are sparsified

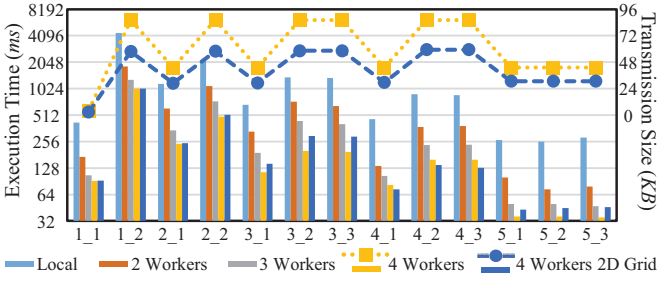


Fig. 9. Computing time and transmission time of CLs in VGG-16.

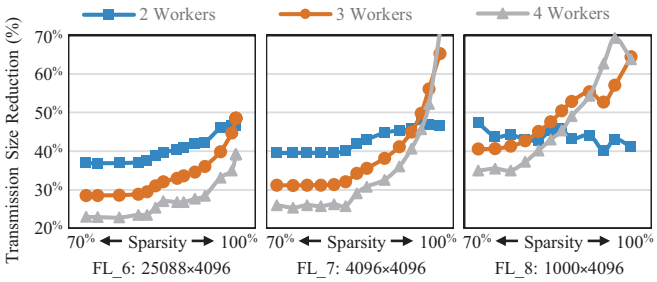


Fig. 10. Transmission size decrease ratio of MoDNN to baseline.

by L_1-norm group lasso with a predefined discarding threshold to control the sparsity [4]. We define the evaluation baseline as the one-dimensional partition that divides the weight matrix along its longer side without any overlaps between the partitioned parts. The experiments are performed on the layer sparsity varying from 70% to 96% and the results of the transmission size reduction are normalized by the one of the baseline, as shown in Fig. 10.

According to the results, MSCC and FGCP effectively reduce the transmission size by at least 22.6% compared with the baseline implementation. In most cases, the transmission size reduction ratio keeps increasing with the layer sparsity, e.g., reaches as high as 49.3%, 69.2%, and 69% for FC_6, FC_7, and FC_8, respectively at different numbers of worker nodes. One exception occurs in the FC_8 with 2 worker nodes, which shows a decrease in the transmission size reduction ratio when the layer sparsity increases. It is because of the residual unbalance in the clustering due to the limited solution space of the small-scale layers ($e.g.$ 1000×4096 for FC_8). Nonetheless, following the increase of the number of the worker nodes, the effectiveness of MSCC and FGCP also increases, demonstrating a good design scalability.

D. Overall Evaluation of MoDNN

Table I summarizes the overall execution time to compute the whole VGG-16 model in DoDNN over different numbers of mobile devices. Following the increase of the number of the worker nodes, the overall execution time reduces significantly, demonstrating excellent computing parallelism: the purely computation time improves by 2.17-4.28× with 2 to 4 worker nodes. Table I also summarizes the data delivery time and the data transmission size of different scenarios, which indicates the extra cost introduced by the distributed computing mechanism of MoDNN. MoDNN also outperforms the conventional 2D-grids partition scheme by substantially reducing the data delivery time though the data transmission size is slightly increased.

TABLE I
OVERALL EVALUATION OF MODNN WITH 2-4 WORKER NODES.

	Execution Time (ms)	Data Delivery Time (ms)	Transmission Size (KB)
Local	15809	0	0
2 Workers	8509	1819	1196
3 Workers	6884	2563	2257
4 Workers	5208	2567	3336
4 Workers 2D-grids	6324	3073	2256

V. CONCLUSION

In this work, we propose MoDNN – a local distributed mobile computing system to enable parallel computation of DNN on mobile platforms. As convolutional layers and fully-connected layers are identified as the major DNN components that contribute to the total execution time, several advanced partition schemes, i.e., BODP, MSCC, and FGCP are proposed to well balance the workloads of each worker nodes and minimize the data delivery time. Experiments show that MoDNN can achieve better than linear performance speedup on DNN computations, demonstrating great potential of mobile platforms in DNN applications.

VI. ACKNOWLEDGMENT

This work was supported in part by NSF CCF-1615475, CNS-1422057, and AFRL FA8750-15-1-0176.

REFERENCES

[1] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *Computing Research Repository*, 2014.
[2] J. Hauswald, T. Manville, Q. Zheng, R. Dreslinski, C. Chakrabarti, and T. Mudge, "A hybrid approach to offloading mobile image classification," in *2014 IEEE International Conference on Acoustics, Speech and Signal Processing*. IEEE, 2014, pp. 8375–8379.
[3] M. Li, D. G. Andersen, J. W. Park, A. J. Smola, A. Ahmed, V. Josifovski, J. Long, E. J. Shekita, and B.-Y. Su, "Scaling distributed machine learning with the parameter server," in *11th USENIX Symposium on Operating Systems Design and Implementation*, 2014, pp. 583–598.
[4] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding," *Computing Research Repository*, vol. 2, 2015.
[5] W. Chen, J. T. Wilson, S. Tyree, K. Q. Weinberger, and Y. Chen, "Compressing neural networks with the hashing trick," *Computing Research Repository*, 2015.
[6] WiFi alliance: WiFi direct specifications. [Online]. Available: http://www.wi-fi.org/discover-wi-fi/wi-fi-direct
[7] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," *Communications of the ACM*, vol. 51, pp. 107–113, 2008.
[8] B. Panda, J. S. Herbach, S. Basu, and R. J. Bayardo, "Planet: massively parallel learning of tree ensembles with mapreduce," *Proceedings of the VLDB Endowment*, vol. 2, no. 2, pp. 1426–1437, 2009.
[9] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.
[10] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *Computing Research Repository*, 2015.
[11] W. Wen, C. R. Wu, X. Hu, B. Liu, T. Y. Ho, X. Li, and Y. Chen, "An eda framework for large scale hybrid neuromorphic computing systems," *ACM/IEEE Design Automation Conference*, pp. 1–6, 2015.
[12] A. Coates, B. Huval, T. Wang, D. Wu, B. Catanzaro, and N. Andrew, "Deep learning with cots hpc systems," *International Conference on Machine Learning*, vol. 20, no. 3, pp. 96–104, 2013.
[13] D. Camps-Mur, A. Garcia-Saavedra, and P. Serrano, "Device-to-device communications with Wi-Fi Direct: overview and experimentation," *IEEE Wireless Communications*, vol. 20, no. 3, pp. 96–104, 2013.
[14] M. R. Garey, D. S. Johnson, and L. Stockmeyer, "Some simplified np-complete graph problems," *Theoretical Computer Science*, vol. 1, no. 3, pp. 237–267, 1976.
[15] U. von Luxburg, "A tutorial on spectral clustering," *Statistics and Computing*, vol. 17, no. 4, pp. 395–416, 2007.
[16] I. S. Dhillon, "Co-clustering documents and words using bipartite spectral graph partitioning," *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.
[17] T. Chen, M. Li, Y. Li, M. Lin, N. Wang, M. Wang, T. Xiao, B. Xu, C. Zhang, and Z. Zhang, "Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems," *Computing Research Repository*, 2015.