Distributed Deep Learning at the Edge with IoT Data

Alka Bhushan, Govind Lahoti, Umesh Bellur
IIT Bombay
Mumbai, Maharashtra, India
abhushan,govind14,umesh@cse.iitb.ac.in

Vinayak Naik
IIIT Delhi
New Delhi, Delhi, India
naik@iiitd.ac.in

ABSTRACT

The proliferation of sensing devices has driven an explosion in the amount of data being generated close to the Edge. Sensing data is usually employed to train a learning model, such as a ANN or DNN situated in the Cloud, where all the data is shipped. The Cloud ships back the trained model to the Edge centers, which serve user requests based on this model. Edge data centers do not usually possess the compute power to keep up with high data velocities for training purposes. However, the cost of data transfer coupled with latency considerations of shipping vast amounts of sensed data over large distances makes us ask the question - why can't we train the learning model at the Edge in a *distributed* manner instead of in the Cloud? This work seeks to explore the specific question of the distribution configuration that best suits a given application. As a part of this, we have developed a distributed learning simulator that can run a given DNN in a data distribution fashion rolling up learned parameter values up a hierarchy of parameter servers that merge parameters received from the lower levels. The root of this hierarchy has the latest model, which is then pushed lazily back down the tree to the Edge servers. We discuss the use of this simulation environment with an air quality sensing and prediction application over a geographic area organized as a grid of small cells some of which are populated with air quality and other sensors. We show how different distributed setups affect the accuracy and cost of the prediction exercise.

CCS CONCEPTS

• **Computing methodologies → Distributed computing methodologies**;

KEYWORDS

Edge Computing, Deep Neural Network, Air Quality Information

ACM Reference Format:
Alka Bhushan, Govind Lahoti, Umesh Bellur and Vinayak Naik. 2019. Distributed Deep Learning at the Edge with IoT Data. In *Proceedings of International Conference on Distributed Computing and Networking (ICDCN'19)*. ACM, New York, NY, USA, Article 4, 5 pages. https://doi.org/10.475/123_4

1 INTRODUCTION

The last decade has witnessed a tremendous surge in the amount of data derived from sensing devices deployed everywhere (smart phones, smart cameras etc.). Sensors are physically close to Edge data centers which house a relatively small amount of compute power. Deriving value from the sensed data usually means that it is used to train a learning model, such as a ANN or DNN situated in the Cloud (with a much larger amount of compute power compared with the Edge) where all the data is shipped. The Cloud then ships back the trained model to the Edge centers, which serves user requests based on this model. This traditional approach incurs a significant cost of data transfer and high latency.

However, the computational power available at the Edge can be used in performing reasonably complex operations on-site instead of sending the data to the central Server/Cloud for training the model. Note that while a single Edge center may not be able to support handling very large models or a high rate of data, a set of Edge centers could cooperatively accomplish what the Cloud does. This results in preserving privacy to users, reducing latency and communication bandwidth, and increasing robustness when connectivity to the Cloud is poor. This trending paradigm is referred by the term 'Edge Computing', where a part of the work happens right at the Edge of the network which connects the physical world to the Cloud seamlessly [9, 10]. An Edge computing application uses the power of Edge devices in preprocessing and filtering the data, and learning complex analytical models in a distributed setting [4, 7, 10, 11].

A distributed DNN over the Edge has been proposed in [10]. The model is trained for the classification problem and built on a hierarchy where each level of the hierarchy contains the model of higher granularity. The model is suitable for image classification where an image is sent to higher level only if it is not classified with sufficient confidence by the model available at a lower level. These existing approaches have shown that Edge hierarchy plays an important role in data distributed learning models. However, setting up an Edge hierarchy is itself a challenging task in terms of mapping sensors to Edge devices and deciding how many levels and how many Edge devices are appropriate for a given application. Thus, *it would be extremely useful to have a simulation environment to simulate the performance and cost of different distributed learning strategies for a given application.*. With the help of this, a user can compare different strategies and can select the right strategy based on the available budget and other application requirements, such as accuracy.

In this effort, we consider air quality information application as the AUT. Air Quality in many large metros across the world is an area of great concern. Hence we are motivated to serve predictions of the concentration of PM 2.5, PM 10, and Air Quality Index (AQI), in the region of interest. Existing studies and models have shown that air quality varies non-linearly in space and changes over time differently in different places [5, 8, 14, 15].

ICDCN'19, January 2019, Bangalore, India
© 2016 Copyright held by the owner/author(s).
ACM ISBN 123-4567-24-567/08/06.
https://doi.org/10.475/123_4

The ability to predict AQI over a specific region and for a specific point in time in the future can play an important role in planning day-to-day activities such as traffic routing/diversion and finding least polluted pedestrian paths. Among various prediction models present in the literature [5, 8, 14], the model based on Deep Neural Network (DNN) given in [8] predicts the AQI values for next 48 hours. The advent of portable AQI sensors [6, 12], has made it feasible to deploy AQI monitoring stations at a higher scale. With high sampling data rates and increase in the number of AQI sensors, a large amount of training data will be available to learn the deep neural network model for AQI prediction. The AQI DNN incorporates spatial and temporal characteristics in the optimization function, which increases the memory usage and run time significantly than a traditional DNN model. We postulate that traditional Edge centers with a few cores of computing power will be insufficient to keep up with the data rate of the entire city's sensors and hence the need for data distributed learning.

This work seeks to explore the specific question of the distribution configuration that best suits a given application. As a part of this, we have developed a distributed learning simulator that can run a given DNN in a data distribution fashion rolling up learned parameter values up a hierarchy of parameter servers that merge parameters received from the lower levels. The root of this hierarchy has the latest model, which is then pushed lazily back down the tree to the Edge servers. We demonstrate the use of this simulation environment with an air quality sensing and prediction application over a geographic area organized as a grid of small cells. We show how different distributed setups affect the accuracy and cost of the prediction exercise.

The rest of the paper is organized as follows: Distribution hierarchical approach is described in Section 2 and simulator is presented in Section 3. Section 4 presents the AQI prediction model. The experiments and results are given in Section 5. Finally, the paper is concluded with the future work in Section 6.

2 DISTRIBUTED AND HIERARCHICAL APPROACH FOR DEEP NEURAL NETWORK

There are two approaches proposed in the literature for distributed training of a deep neural network [3]: model parallelism and data parallelism. The former typically splits each layer of the DNN to run on its own server while the latter trains different instances of the model with subsets of the data and then merges the learnings from the instances using parameter servers. Model parallelism is suited for very large models with a many input features and since the AQI model does not possess this characteristic, we focus on the data parallelism approach where training data is distributed on multiple machines and assume that the complete model fits in any Edge device present in the computing framework.

We organize the set of Edge nodes into a hierarchy where the set of Edge nodes are heterogeneous in terms of compute capability. In our framework, we assume that each Edge device has the model and the training occurs at a few Edge devices acting as worker nodes and other Edge devices acting as parameter servers to refine the model.

2.0.1 Approach. In our framework, we assume that worker nodes are available at the lowest level of hierarchy and are connected to the sensors. Sensors are mapped to Edge node such that sensors mapped to a single Edge node are geographically close to each other. Workers are the actual computational units, which train their model using the training data points, while parameter servers act as relay between the workers to communicate. Each worker has its own copy of the entire model. Workers send their model updates (learned values of the parameters) to the designated parameter server, which merges all the updates and sends the accumulation to the workers upon request. When each parameter server merges multiple model's parameters, it propagates the updates up the tree to higher level parameter servers. The root of the tree is where this process stops. The organization is shown in Figure 1.

2.1 Training the hierarchy of nodes

Each worker node trains its own copy of the model and sends the gradients acquired from each epoch to its parent - an epoch defines the number of sensor events consumed by the model for each training cycle. The parent node updates its model using the gradients obtained from the child nodes by, either simply adding gradients to the weights, averaging the gradients obtained from each child, or adding the weighted average of the gradients where weights can be decided dynamically based on the accuracy of the model improved by gradients obtained from each child. It then sends the gradients to its parent and so on. In the downward direction, every node obtains the latest copy of the model from its parent on demand. We use the Downpour Stochastic Gradient Descent algorithm given in [3] for sharing the models/gradients up and down the levels in the tree.

3 SIMULATOR

Since Tensorflow [1, 2] limits the user's control over the data/model sharing schemes in distributed training of neural networks, we have built a simulator that is capable of measuring the performance of a distribution strategy for training a hierarchy of nodes. The simulator is coded in Python 3 and executes on multiple machines. Each process simulates a worker node that reads the sensor data and trains the actual DNN model (for AQI prediction in our case) for an epoch of sensor values. Before and after each epoch, workers test the accuracy of their model with a set of test queries present at each node. Workers communicate with designated parameter servers using sockets for pulling and pushing the model. Workers push the model parameters to their parent at the end of an epoch (an epoch's duration is cutomizable). All communication is asynchronous and each parameter server reevaluates its model on receiving the parameters from any of its children by testing the accuracy of the model before and after merging the updates of the parameters. After reevaluation, it pushes the model up to its parent should it have one in the hierarchy. Each parameter server pulls the model from its parent after updating the model for a predefined number of times. The simulation tool then computes the cost of the training run (in terms of network bytes transferred and computational power used in all nodes) and the performance of the training run (the number of seconds used for training the model on each sensor input). The simulator also reports back on the accuracy of the prediction for

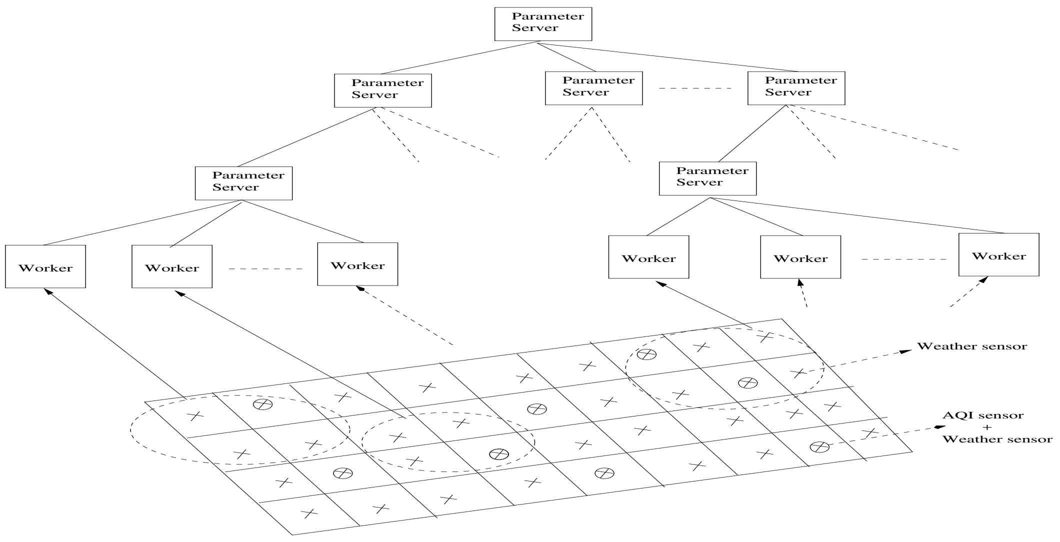

Figure 1: Edge Hierarchy

the test set performed at each worker and parameter server node at various points of the run.

3.1 Configuration

Configuration of the Edge network (Edge centers and the connectivity between them) and a pointer to the input data on each worker node are given to the Simulator in a yaml configuration file. The yaml file contains all the customizable variables of the simulation environment, such as the delays between nodes (network latencies), time interval for pulling, and pushing the model, and lists all the nodes present in the framework. For each node, it provides an unique id, its IP address (e.g. localhost), the port on which the RPC server corresponding to that node is to be run and the file that contains training data. The ID of the parent node (if present) is also specified. The delays and time intervals for pulling and pushing the model can also be customized.

3.2 Simulator Functions

Our simulator does the following:

- Reads in the configuration file passed to it as command line argument.
- After reading the specifications from the configuration file, the master spawns the (worker and parameter server) nodes on separate threads. Each node further spawns a separate thread for its RPC server. We use a built-in `SimpleXMLRPCServer` library for this purpose. RPC is used for all the inter-node communication. Functionalities of nodes, like pushing/pulling the gradients/model, has been implemented using the RPC functions.
- Each leaf node begins reading from the data file (representing a sensor data source) provided to it. Each node trains their model using the Downpour SGD algorithm given in [3].
- Each node pushes its gradients to the parent node and pulls the model from the parent node at intervals specified in the configuration file, where intervals are the number of times model has been reevaluated at the node.
- Delays are inserted in the RPC call to simulate the network latencies between the nodes.
- Each node logs all the events taking place on it. These logs are pushed to the master controlling the simulation and can be processed after the simulation run is over to obtain the analytics.

4 AQI MODEL [8]

The problem is defined in [8] as follows: A given geographic area is divided into a grid of cells $[g_1, g_2, g_3, \ldots, g_n]$ of equal size, where each cell contains sensors to collect weather information. AQI sensors are placed at few grid cells only. The data obtained from cells containing AQI sensors are called as labeled data and the data from remaining cells containing only weather data is called as unlabeled data. To predict the AQI values of all the cells (geographic areas), both the weather data of the cell as well as sensor readings of either that cell or its nearest neighbors is used. Spatio-temporal semi-supervised learning is added in the output layer of the model to achieve useful information from the unlabeled data. This Spatio-temporal learning requires access to training data points from the nearest cells in the spatial neighborhood and previous data points in the temporal neighborhood. The model considers feature selection and prediction, both in different layers of deep neural network. In this work, we focus on prediction only and thus ignore the feature selection layer. The deep neural network model without feature selection is shown in Figure 2.

5 EXPERIMENTS

We tested the AQI prediction accuracy obtained from the simulator using the air quality data obtained from air quality monitoring stations in Beijing [13, 14]. There were 36 monitoring stations and the data was observed for a year (2013-2-8 to 2014-2-8). At each timestamp, Air Quality Information contains PM2.5, PM10,

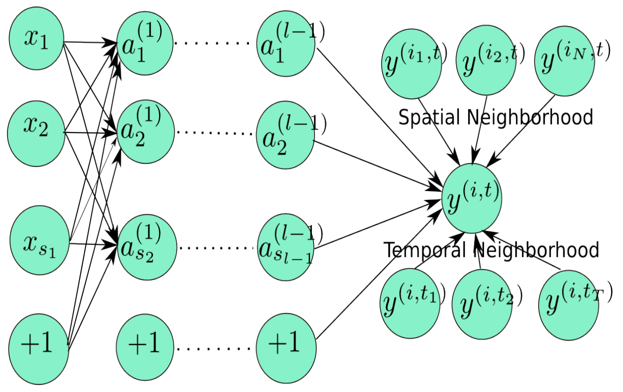

Figure 2: AQI Prediction Model

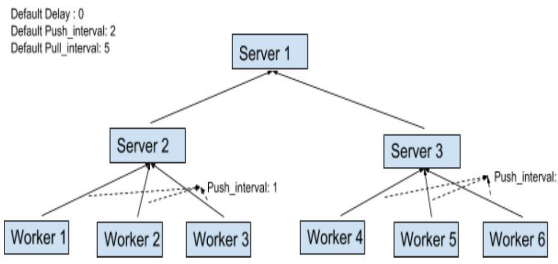

Figure 3: Edge Hierarchy for Network 1 on AQI Data

and NO2 values. The meteorological data contains temperature, pressure, humidity, wind speed, and weather. The null values from the data are removed by adding respective mean values. We use the same features for training data as used in [8] except weather forecasting values are not used. We randomly chose 7 sensors from 36 sensors, which contains AQI values. For the remaining sensors, AQI values were ignored and we used the data as unlabeled data. Our AQI DNN contains 276 input features and 48 output values. We obtained a total of 168911 data points from which 158645 points were used as training data points and remaining 10266 were used for testing the model. Note that test data points are from the sensors, whose values were not included in the training data. We cluster the sensors into 6 groups and for each group, we store the training data points of the corresponding sensors. Each group of sensors is also mapped to a worker node in running the AQI model.

Our simulator configuration is as follows: We run the simulator for two different configurations, where the configuration given in Figure 3 is a three level hierarchy and another configuration given in Figure 4 is a two level hierarchy.

We run the simulator using two servers each contains 6 Intel (R) Xeon (R) quadcore CPUs and 25 GB RAM.

We tested both the configurations on our simulator and compared the results with the centralized framework, where all the data is sent to a central server representing the Cloud and the model is trained there alone. For the experiments, we used the values of model parameters given in Table 1. The gradients obtained for a child node are averaged at the parent node.

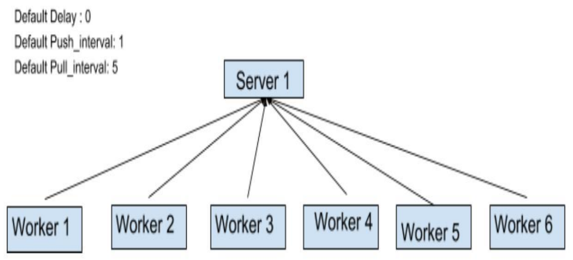

Figure 4: Edge Hierarchy for Network 2 on AQI Data

Table 1: Parameters of AQI Model

Parameter	Value
Spatial Neighborhood Parameter (α)	5
Temporal Neighborhood Parameter (β)	0.2
Regularization Parameter (λ)	2
Learning Rate η	.01

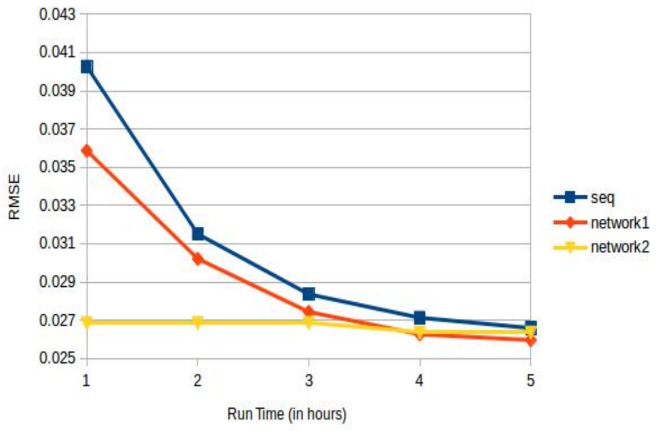

Figure 5: Varying Simulator Run Time

5.1 Varying Simulator run time

In this experiment, we run the simulator for different periods of time and measure the prediction accuracy of the model for PM2.5 values at the root node. The results are shown in Figure 5. From the results, we observe that the root mean square error (RMSE) is higher initially in Network 1 but improves beyond that of the Network 2 when the model is run for 5 hours. Further, we observe that the accuracy of the model of Network 1 and Network 2 is always better than that of the sequential (Cloud based) model. From this, we claim that we can always train the model sequentially in a way trained in the distributed framework. Thus, we can always achieve the same accuracy.

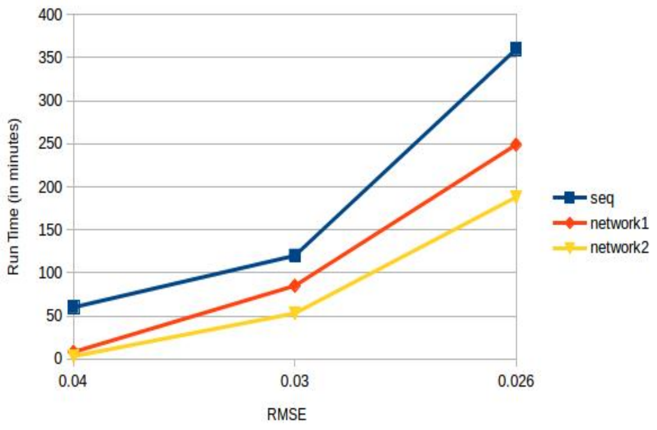

Figure 6: Varying RMSE values

5.2 Varying RMSE

In this experiment, we run the simulator until we achieve the desired RMSE value. The results are in Figure 6. From the results, we see that the distributed setups (both of them) are significantly faster than the sequential model. Further, we see that network 2 configuration is much faster that the network 1 configuration leading us to conclude that a two level hierarchy is not as performant as a single level one for the AQI Application.

6 CONCLUSION

In this work, we postulate that data distributed training of DNNs is a viable alternative to the sending all data to the Cloud. We presented a tool to simulate distributed training of a DNN with different distribution configurations. The simulator was tested for AQI prediction. This is preliminary work done in simulating an Edge computing environment for measuring the cost of the framework. The results showed that distributed configuration also gives the similar prediction accuracy to sequential model at the cost of faster run time. The work is on going and we are currently focused on building accurate cost models into the simulator as well as experimenting with other applications.

REFERENCES

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2015. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. (2015). https://www.tensorflow.org/ Software available from tensorflow.org.

[2] Martin Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2016. TensorFlow: A system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. 265–283. https://www.usenix.org/system/files/conference/osdi16/osdi16-abadi.pdf

[3] Jeffrey Dean, Greg S. Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Quoc V. Le, Mark Z. Mao, Marc'Aurelio Ranzato, Andrew Senior, Paul Tucker, Ke Yang, and Andrew Y. Ng. 2012. Large Scale Distributed Deep Networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1 (NIPS'12)*. 1223–1231.

[4] Seungyeop Han, Haichen Shen, Matthai Philipose, Sharad Agarwal, Alec Wolman, and Arvind Krishnamurthy. 2016. MCDNN: An Approximation-Based Execution Framework for Deep Stream Processing Under Resource Constraints. In *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys '16)*.

[5] Hsun-Ping Hsieh, Shou-De Lin, and Yu Zheng. 2015. Inferring Air Quality for Station Location Recommendation Based on Urban Big Data. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '15)*.

[6] V. Jain, M. Goel, M. Maity, V. Naik, and R. Ramjee. 2018. Scalable Measurement of Air Pollution using COTS IoT Devices. In *Proceedings of 10th International Conference on Communication Systems and Networks (COSMSNETS âĂŸ18)*.

[7] G. Kamath, P. Agnihotri, M. Valero, K. Sarker, and W. Z. Song. 2016. Pushing Analytics to the Edge. In *IEEE Global Communications Conference (GLOBECOM)*. Washington, DC, 1–6.

[8] Z. Qi, T. Wang, G. Song, W. Hu, X. Li, and Z. M. Zhang. 2018. Deep Air Learning: Interpolation, Prediction, and Feature Analysis of Fine-grained Air Quality. *IEEE Transactions on Knowledge and Data Engineering* (2018), 1–1. DOI: http://dx.doi.org/10.1109/TKDE.2018.2823740

[9] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu. 2016. Edge Computing: Vision and Challenges. *IEEE Internet of Things Journal* 3, 5 (2016), 637–646.

[10] S. Teerapittayanon, B. McDanel, and H. T. Kung. 2017. Distributed Deep Neural Networks Over the Cloud, the Edge and End Devices. In *IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*. Atlanta, GA, 328–339.

[11] S. Wang, M. Zafer, and K. K. Leung. 2017. Online Placement of Multi-Component Applications in Edge Computing Environments. *IEEE Access* 5 (2017), 2514–2533.

[12] yun Yu Jiang and Cheng-Te Li. 2016. Forecasting Geo-sensor Data with Participatory Sensing Based on Dropout Neural Network. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management (CIKM '16)*.

[13] Y. Zheng, X. Chen, Q. Jin, Y. Chen, X. Qu, X. Liu, E. Chang, W. Ma, Y. Rui, and W. Sun. 2014. *A Cloud-Based Knowledge Discovery System for Monitoring Fine-Grained Air Quality.* Technical Report MSR-TR-2014-40.

[14] Yu Zheng, Furui Liu, and Hsun-Ping Hsieh. 2013. U-Air: when urban air quality inference meets big data. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining (KDD'13)*.

[15] Yu Zheng, Xiuwen Yi, Ming Li, Ruiyuan Li, Zhangqing Shan, Eric Chang, and Tianrui Li. 2015. Forecasting Fine-Grained Air Quality Based on Big Data. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '15)*.