Implementing Deep Learning and Inferencing on Fog and Edge Computing Systems

Swarnava Dey (Author)
Embedded Systems and Robotics
TCS Research & Innovation
Kolkata, India
Email: swarnava.dey@tcs.com

Arijit Mukherjee (Author)
Embedded Systems and Robotics
TCS Research & Innovation
Kolkata, India
Email: mukherjee.arijit@tcs.com

***Abstract*—The case for leveraging the computing resources of smart devices at the *edge* of network was conceptualized almost *nine* years back. Since then several concepts like Cloudlets, Fog etc. were instrumental in realizing computing at network edge, in physical proximity to the data sources for building more responsive, scalable and available Cloud based services. An essential component in smartphone applications, Internet of Things(IoT), field robotics etc. is the ability to analyze large amount of data with reasonable latency. Deep Learning is fast becoming a *de facto* choice for performing this data analytics owing to its ability to reduce human interventions in such workflows. Major deterrent of providing Deep Learning based Cloud services are Cloud outages and relatively high latency. In the current article the role of Fog Computing in addressing these issues is discussed, current state of standardization in Fog / Edge Computing is reviewed and the importance of optimum resource provisioning for running Edge-Analytics is highlighted. A detailed design and evaluation of the distribution and parallelization aspects of an Edge based Deep Learning framework using off-the-shelf components is presented along with strategies for optimum resource provisioning in constrained edge devices based on experiments with system resource (CPU, GPU & RAM) consumptions of a Deep Convolutional Neural Network.**

I. Introduction

In today's *Digital World* the dumb rule based network end-devices are giving way to intelligent, semi-autonomous devices that provide tailor-made, real-time services. These service endpoints include smartphones, wearable devices, autonomous vehicles, robots & drones and other embedded systems used in domains such as healthcare, city services, engineering, finance, entertainment and several others. As these sensor fitted devices, sensors and human beings are generating high amount of contextual data, the options for application of artificial intelligence (AI) for rendering more intelligent, customized and effective services are increasing. Though these data-driven, machine inferred services promise to bring in a paradigm shift in several different application areas, the major challenge remains in managing the *big data* generated by data sources and applying AI for learning/inferencing in a distributed fashion for near accurate and near real-time response. Due to fast changing nature of services and requirement of fast time-to-market, there is a shift of focus towards automated generation of *features* for machine learning via *Deep Learning* (DL) [3], [5], from traditional hand engineering of *machine learning features*. In a typical supervised DL application, input data with desired output is fed to a complex *neural network* (NN) with processing and non-linear transition happening at several layers to adjust the NN to form a model, which can be utilized later to predict/classify/encode new sets of data. Cloud based sensor data analytics frameworks are often challenged by low latency requirement of the applications and intermittent network connectivity due to high mobility of the devices. To handle these issues resource rich devices within local access network can be utilized and this was first established in [1], where service software was offloaded for execution on a *Cloudlet* virtual machine. Fog Computing [2] proposed in 2012 envisaged deployment of services within local access network, augmenting Cloud based deployments. As an ongoing activity, Multi-access Edge Computing (MEC) [6] initiative from ETSI is standardizing the process of deploying applications and services in applications like video analytics, Internet of Things (IoT) etc. in the Radio Access Network (RAN) edge. With the availability state-of-the-art distributed DL frameworks like TensorFlow [7] and scalable Cloud/Fog based infrastructure it is possible to rapidly develop intelligent, data-driven services. In this work we address this issue of running DL based analytics in a *Cloud-Edge* setup. We focus on two primary requirements for successful deployment of such services: a) analyzing resource requirement of applications in terms of processing speed, memory for optimum resource utilization and b) utilizing the workload information for optimized partitioning of load between networked resources in order to minimize deployment cost or response time or some other parameter important for the problem at hand.We perform fine grained analyses of the resource requirement for offloaded execution of a representative application that implements DL analytics on streaming data, considering NN size (depth, width, number of layers), data throughput, NN hyperparameters (feature extraction filter size, batch size) etc. with respect to execution time, CPU, GPU and memory requirement for different levels of accuracy of prediction/classification. We present a set of benchmarking methods and results and hope that these will be helpful in provisioning optimum resources at network edge to design effective DL analytics frameworks capable of handling large volume of streaming data. We also discuss the distribution and parallelization strategies using

978-1-5386-3227-7/18/$31.00 ©2018 IEEE

using off-the-shelf components for running an edge based Deep Convolutional Neural Network (DCNN).

II. Background: Edge Analytics

Deep Learning (DL) [5] is a method of machine learning (ML) for classification/regression/encoding. Due to brevity of space we redirect the reader to articles that elaborate basic concepts [3], [4] of DL and specifically CNN [9] for understanding the benchmark results presented in the later sections. In recent past several frameworks and applications are developed for realizing DL analytics in mobile devices. TensorFlow [7], an open source distributed DL middleware from Google has got its mobile implementation [8]. Application of DL in mobile sensing in general and activity recognition through smartwatch is presented in [10] and [11] respectively. In contrast to small scale, standalone applications on mobile devices, a large scale analytics platform requires: a) *Intermediate Resource Standardization*: Mobile devices as well as intermediate hierarchical compute and storage devices within local network for data manageability and low latency, b) *Provisioning of Intermediate Resources*: expected compute, buffering and memory load expected in the intermediate servers such that the DL analytics platform designer can decide the local processing proportion, given there is a mechanism to partition the load and c) *Offloaded Execution*: consider the capability of external network and capacity of resources connected to external network along with the resources in the local network for effectively partitioning a DL analytics that minimizes latency or cost or any other target associated with the problem at hand.

A. Resource Standardization for Edge Computing

European Telecommunications Standards Institute (ETSI) [12] is standardizing a set of services (Multi-access Edge Computing (MEC) [6]), that allows different third party vendors to deploy services at the Radio Access Network (RAN) edge server. These servers, to be hosted in LTE eNodeB , multi-RAT access points etc. are connected to the mobile devices via local radio networks and promise to provide ultra low latency. The major benefits expected out of this initiative, apart from the advantage of compute and storage in local network, are as follows:

- Trust: Data generated from personal mobile phones, private or government deployed sensors are often confidential. This is often a deterrent for offloading such data for processing in the wild. As an example many organizations are using Amazon's EC2 and S3 for computing and storing in the Cloud as privacy and safety (data loss prevention) etc. of the data is guaranteed.
- Monetization: With standardized revenue models for MEC resources, the third party vendors and analytics designers would be able to monetize their services.

The current state of MEC initiative [6] looks promising with several specifications available, that might play an important role in shaping up edge computing. The key take away for edge based data analytics are:

- In specification *GS MEC-IEG 004: Mobile-Edge Computing (MEC); Service Scenarios* at *section-4.7* an IoT gateway service scenario is depicted where the aggregation and distribution services are implemented. The services include device message analytics, analytic results - feedback/actuation, storage including database, access control and provisioning of devices
- In specification *GS MEC 002: Mobile Edge Computing (MEC); Technical Requirements* standardizes the connectivity (sec 6.2.2), storage (6.2.3) aspects that is required for low latency execution of Big Data analytics with buffering. Particularly interesting the ability of the mobile device to directly communicate with each other (6.2.2 - [Connectivity-02]), that may enable peer to peer offloading.
- In *GS MEC-IEG 004* Traffic routing (sec 6.2.4) specifies the ability of the end devices to modify the network bandwidth, network usage priorities that may be effective in handling *Variety* of Big Data from different variety of data sources and actuating devices.
- In section A.4 of *GS MEC-IEG 004* detailed requirements and architecture of a Big data use case with massive sensor data pre-processing and data processing at Smart City scale are discussed. Based on this an architecture shown in Fig. 1 can be envisaged.

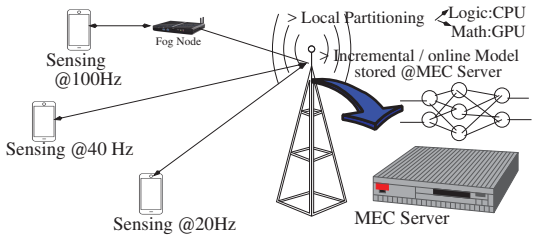

Fig. 1. A Deep Learning based Big Data analytics architecture running on MEC

Similar to the MEC initiative, OpenFog Consortium Architecture Working Group proposed a specification and implementation architecture [13] for leveraging the resources like routers, gateways, smartphones in local access network. The framework is similar to *Volunteer Computing* [14], [15] and is far more open in terms of *vendor lock-in* and usage cost. Fog Computing researchers need to come up with secure mechanisms of running code in peer devices, viable schemes for incentivization for people to up their private devices and several other innovations highlighted consicely in [16].

B. Deep Learning in Edge: Current State of Art

In recent past several research efforts [17], [18], [20], [19] address the issue of data analytics, especially using DL on Fog, Edge and Cloud setup. In [17], distributed DL is implemented on devices, Edge servers and Cloud servers. The two main building blocks in this framework are: a) the ability to *exit* before the input data completes a full *forward pass* through all the layers of a neural network distributed over heterogeneous

nodes and b) the use of Binarized Neural Networks (BNNs) to reduce the memory and compute load on resource constrained end devices.The *early exit* is based on BranchyNet, designed earlier by the authors, that allows early exit points to be placed in Deep neural network layers, allowing *accuracy-resource* trade-off . The experiments for object detection are conducted on a multi-view multi-camera dataset with a combination of convolutional and fully connected deep network, where the aggregators at Edge and Cloud level hierarchically manage the learning process leading to communication load reduction by a factor of 20. In the current work we use standard DL Networks (in contrast to BNNs) that is commonly used in majority of applications and benchmark the resource requirement in a layer wise fashion for ensuring optimum resource allocation in the processing nodes. *ECHO* [18], a full feature, comprehensively evaluated platform for realizing data analytics on a distributed hybrid *Edge-Fog-Cloud* setup. ECHO provides handles for dataflow management across resources through a *dataflow programming model* that allows streams, files and data batches to be processed. It provides services such as resource discovery, deployment and state monitoring of virtualized applications and interfaces to data analytics runtime engines. In [19], a distributed analytics platform involving Fog devices is developed by using open-source frameworks: TensorFlow (DL programming model), Docker (virtualization tool for Raspberry Pi based Fog devices) and Kubernetes (for cluster management and orchestration). The benchmarking results of running an object detection application and a sensor reading classification application are reported highlighting the increase in data processing throughput and decrease in CPU load due to inclusion of Fog devices. Such frameworks demonstrate the availability of tools and methods for realizing Fog vision [2] for complex algorithms like DL. A major issue that none of the earlier efforts address is optimum resource provisioning in the heterogeneous cluster nodes. With MEC and Fog Computing standardizing the landscape of intermediate resources in Big Data analytics, the challenge remains in understanding the resource requirement for running DL algorithms in those resources. Such understanding is key to allocate resources in an optimized and distributed fashion and avoid failure in sustaining the analytics pipeline in the face of high velocity and volume of data.

III. Server Profile at Network Edge

As a first step towards optimum resource provisioning, we profile the workload of executing DL algorithms on different representative resource configurations with different combinations CPU, GPU and RAM, presented in table I. With growing interest on large scale activity inferencing in *Smart City* use cases, we choose tri-axial accelerometer data for Activities of Daily Living (ADL) [23] form UCI [22] as representative workload. Docker [21] containers running distributed TensorFlow [7] are hosted in server class machines, connected by wireless LAN. Inferencing and training are performed on the representative datasets streamed from client machines.

TABLE I
A table of container configurations representing edge servers. Each CPU is of type Intel(R) Core(TM) i7-7500U CPU @ 2.70GHz and each GPU core is having 1.62 GHz GPU clock rate, 533 Mhz memory clock rate and a memory bus width of 64-bit. RAM is specified in gigabytes (GB)

Variant	CPU	GPU	RAM	Represents
Type-1	1.5	0	2.5	smartphone
Type-2	3	0	4	Robots, Gateways
Type-3	4.5	4	5.5	edge server
Type-4	6	8	7	edge server
Type-5	9	18	10	edge server
Type-6	12	24	14	edge server

We observe that tuning of different *hyperparameters* for DCNN has significant effect on the resource usage. We use the profiling results for varying training data, convolution kernel sizes, streaming window sizes, training batch sizes for determining if it is possible to run such algorithms in a particular type of resource and find the bound on execution time, given a particular resource configuration. Using a DCNN with *two convolution*, *two pooling* and *one fully connected* layers, we present a few sample benchmarking results in this section. The *throughput* shown in the benchmarks are tri-axial accelerometer samples (containing *three* data points) per second with The average number of samples comprising different activities ranging from 250 to 1000 data points per activity. *Accuracy* values in the benchmarks represent how often predictions matches labels averaged by the number of samples as specified in TensorFlow [7] reference on *tf.metrics.accuracy*.

Table II shows the accuracy of classification and average inferencing throughput using full ADL dataset. However in most DL applications, it is expected that only a limited set of labeled training data will be available and some *pre-trained* model will be incrementally trained. From Fig. 2 it can be observed that there is a linearly increasing relation of accuracy with data size for the larger resources of type 3-6. However the smaller resources representing mobile phones, robots etc. have problem in loading the model (500MB) for a sample network (*three* convolution and *two* fully connected layers) and for those the accuracy improvement is shown as *zero* in the plot 2. Thus model retraining is recommended with the edge servers with the minimum system resources we designate, for network end devices such models need to be loaded offline. The requirement of using different window sizes on input data is

TABLE II
A table showing the accuracy of classification and average inferencing throughput on the same dataset when the full training set is used for model building

Processor	Inference Throughput	Prediction Accuracy
Type-1	5779	0.252
Type-2	12112	0.486
Type-3	20939	0.738
Type-4	13962	0.648
Type-5	21590	0.741
Type-6	21881	0.780

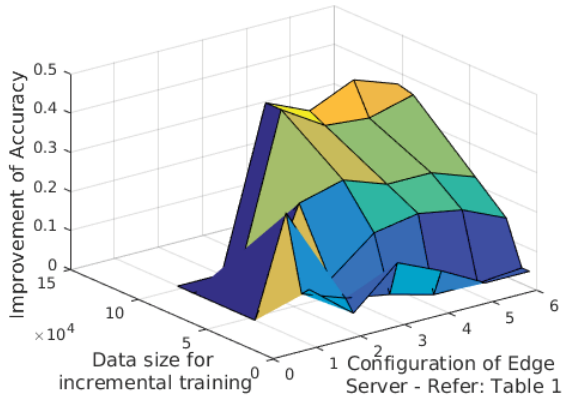

Fig. 2. Demonstration of the effect of labeled training data size on classification accuracy improvement in a incremental learning setup on a pre-trained model when executed on different Edge server configurations

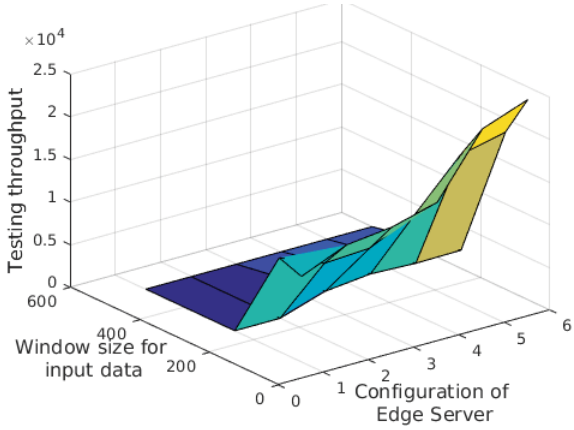

Fig. 4. Demonstration of the effect of using different window sizes to segment / buffer input data on inferencing throughput when executed on different Edge server configurations

governed by the buffering capability at the server and network. As shown in Fig. 3 and 4 window sizes more than 256 are unusable due to high computation load. Though smaller window sizes betters the throughput, maximum accuracy is achieved at 128 and 256 sized windows. We perform tuning

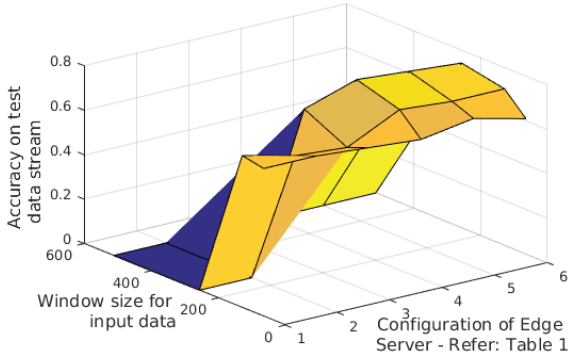

Fig. 3. Demonstration of the effect of using different window sizes to segment / buffer input data on accuracy when executed on different Edge server configurations

of epoch size, DCNN kernel size and minibatch size on similar lines and note the resource requirements. We observe that a) increasing the number of epoch increases the accuracy on training data but reduces the inferencing accuracy after a threshold, b) using larger *minibatch* sizes reduces accuracy and for many different datasets 64 and 128 sized batches provided best accuracy and minimum memory load and c) for DCNN kernels or filter masks, that act as feature detectors [9] the accuracy improves at certain kernel size configurations and drops for others.The reason for getting good accuracy for only few selected kernel sizes, shown in Fig 5 can be due to the nature of the particular dataset that we used. The average number of samples comprising different activities ranged from 250 to 1000. As ADL sizes varied widely, we had to carefully pack individual ADL types and segment into windows for building training and testing datasets. It is possible that some of the kernel sizes were suitable for learning features for some activity classes, providing an average increase of accuracy. From resource perspective, kernel size tuning was possible only in case of resource type 4, 5 and 6. Through these

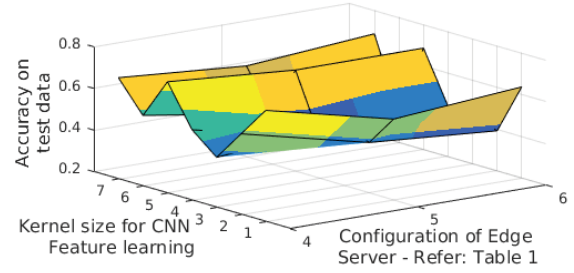

Fig. 5. Demonstration of the effect of tuning the kernel size in a CNN instance with 2 convolutional layers on training and testing accuracy when executed on different Edge server configurations

experiments we get layerwise (operation wise) execution time bounds, CPU / GPU and memory load, which acts as building block optimal load distribution.

IV. Optimized Offloading at Edge and Cloud

To realize the goal of optimal load distribution among Edge, Cloud and end device type resources we consider a scenario where Fog / MEC level server requires further offloading to Cloud servers for execution of the Deep Learning analytics application. Fig. 6 depicts such a scenario where processing capability at servers, network latency, compute cost and data transfer cost are considered for optimum load management. We develop agents running on the edge devices, where one edge device acts as a master and other devices can join to advertise their resource configuration, cost. The master agent, shown in Fig. 7 distributes tasks to other agents and returns result to the client system. The minimum unit of distributed execution for the current version is a single Layer Operation (LO), achieved by using the *with tf.device* directive as shown in Fig. 8. In future we would like to partition the computation graph in a more granular way so that each layer processing

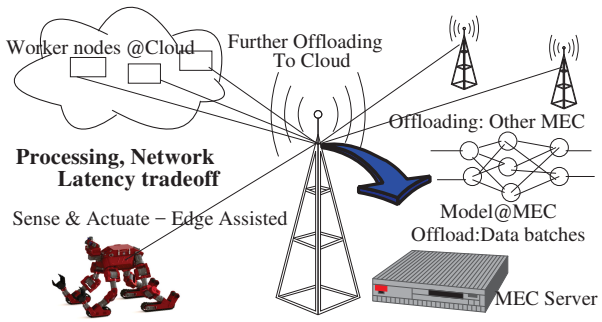

Fig. 6. An *Edge-Cloud* Computation offloading architecture for Deep Learning

can be split for parallel processing. For optimal workload assignment we use a simple capacity based partitioning scheme.

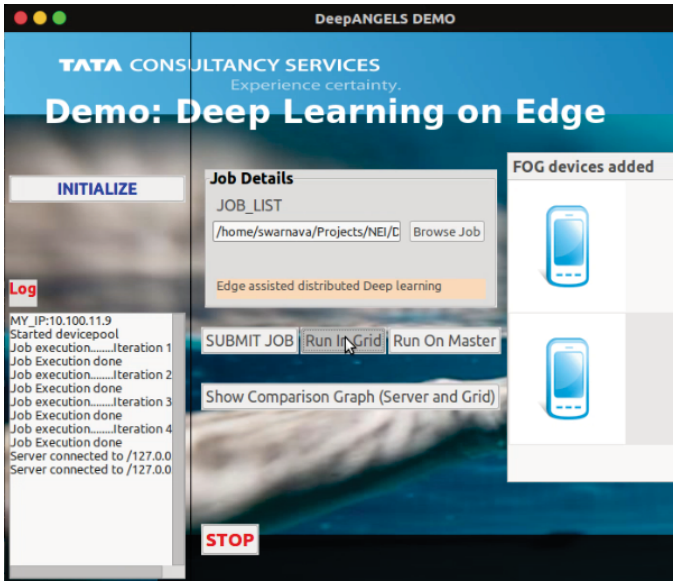

Fig. 7. An *Edge-Cloud* Computation offloading framework for Deep Learning

A. Numerical Model for Task Partitioning

Given a TensorFlow code with a set of LOs *Ops* = l_1 l_2 ... l_n and a set of grid computing elements (Edge/Cloud) *G* = g_1 g_2 ... g_m, we would like to find an optimal allocation that minimizes the learning time and cost of computing. Let $P_i^{l_j}$ denote the processing time per grid element g_i for LO l_j, $N_i^{l_j}$ denote the network transfer time required for model parameter and result exchange by the TF middleware between grid element g_i and the master edge device for LO l_j and $C_i^{l_j}$ denote the computing cost for i^{th} grid element g_i for j^{th} LO l_j. For simplification of the current version of the model we make *two* modifications as follows:

- As each layer might have different execution times (e.g. convolutional layer vs fully conected layer), we partition the computation graph to create offloadable tasks containing one or more LO, such that each composite task unit has similar execution time. Let the modified definition for a composite offloadable task is $\widetilde{l_i}$. We get this information from the benchmarking presented in the earlier section. However this is a workaround and in future we plan to formulate the constraint optimization so that individual layer can be offloaded.
- We observed that a LO can be completed in an edge device even with memory limitations (swap enabled), albeit taking more execution time. For the current model we thus consider only processing time for partitioning.

- Running "In Graph partitioning" through TensorFlow

```
cluster = tf.train.ClusterSpec({"worker": ["10.100.11.18:2222",
"10.100.11.18:2223 ... and so on
```

- Starting workers in different Docker instances in any PC, RaspberryPi, Android (TBD)

```
docker run -it --cpus="6.5" --memory="7.5g" --net=host --name tw1 tf-dist-w

Initialize GrpcChannelCache for job worker ->
{0 -> 10.100.11.18:2222, 1 -> 10.100.11.18:2223} 2017-11-23

05:30:29.191703: I tensorflow/core/distributed_runtime

/rpc/grpc_server_lib.cc:316] Started server with
target: grpc://10.100.11.18:2222
```

- Distributing Computation Graph by prior estimation of the resources load in a computation block and automated insertion of Python directive in TensorFlow code:

```
with tf.device("/job:worker/task:1"):
      three convolutional layers with their channel counts, and a
      fully connected layer (tha last layer has 10 softmax neurons)
      K = 4  # first convolutional layer output depth
      L = 8  # second convolutional layer output depth
      Y1 = tf.nn.relu(tf.nn.conv2d(X, W1, strides=[1, stride, stride, 1], p
```

Fig. 8. Using TensorFlow to assign different LOs to different Docker containers

Based on the above discussion the operation assignment variable is defined as:

$$x_{ij} = \begin{cases} 1 & \text{if } \widetilde{l_j} \text{ is assigned to } g_i \\ 0 & \text{otherwise} \end{cases}$$

Formally, the master edge node solves the following optimization problem for finding x_{ij}:

$$min(\alpha * max((P_i + N_i) * \sum_{j=1}^{n} x_{ij}) + \beta * C_i^{l_j} * \sum_{j=1}^{n} x_{ij})$$
$$\forall i : 1 \leq i \leq m$$

such that:

$$\sum_{j=1}^{n} x_{ij} = 1 ... \forall i : 1 \leq i \leq m \quad (1)$$

$$max((P_i + N_i) \leq \tau ... \forall i : 1 \leq i \leq n \quad (2)$$

The objective function minimizes the makespan of all the jobs (longest execution time among the jobs) and the cost to offload. Relative importance to the constituent parameters is use-case specific and is denoted by α and β. The first constraint for optimization stipulates that each LO is assigned to exactly one edge. The second constraint dictates that the makespan for DL algorithm execution must be less than the deadline τ specified by the client that assigned job to the master edge server. For

complex DL implementations with large number of layers, overhead of optimization is less compared to the benefits achieved in terms of makespan improvement and cost savings. Fig. 9 demonstrates the makespan improvement achieved by using our capacity based partitioning strategy in comparison to an uniform load distribution among the participating edge nodes.

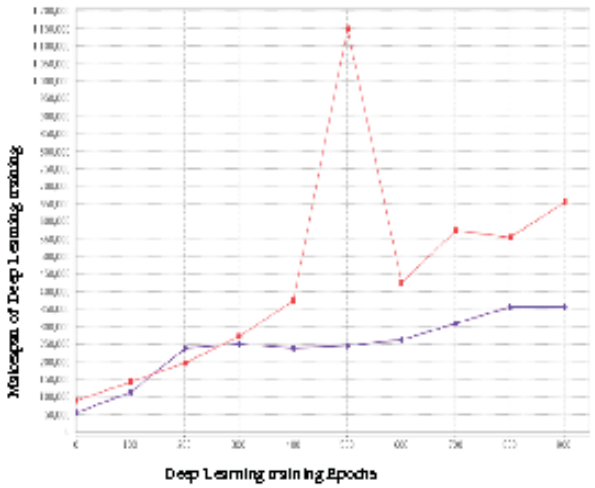

Fig. 9. A comparison of the makespans (in *miseconds*) of naive uniform workload distribution *in red* and optimized distribution *in blue* for a Deep Learning job

V. Conclusion

In this article we have presented a set of methods, results and insights that can help in building Edge augmented data analytics systems capable of handling high volume data with minimum human intervention. We highlighted the need for resources at hierarchical, intermediate levels for effective analytics through Deep Learning in an *Edge-Cloud* setup. With reference to existing standards in Fog and Edge Computing, we underlined the importance of being able to allocate optimum resources at network edge for smooth and cost effective functioning of deep learning based analytics applications. Along with the results in terms of effect in accuracy and system resource requirement due to hyperparameter tuning, we implement and evaluate an optimized computation offloading framework for distributed Deep Learning on resource constrained device grid. Building a framework for automatic tuning of Deep Learning algorithms for best accuracy under specified resource constraints including energy consumption at end devices and improving the DL partitioning strategy will be fitting future work for us.

References

[1] M. Satyanarayanan, P. Bahl, R. Caceres and N. Davies, *The Case for VM-Based Cloudlets in Mobile Computing,* in IEEE Pervasive Computing, vol. 8, no. 4, pp. 14-23, Oct.-Dec. 2009. doi:10.1109/MPRV.2009.82

[2] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. 2012. *Fog computing and its role in the internet of things.* In Proceedings of the first edition of the MCC workshop on Mobile cloud computing (MCC '12). ACM, New York, NY, USA, 13-16. DOI=http://dx.doi.org/10.1145/2342509.2342513

[3] Bengio Y: *Learning Deep Architectures for AI*. Now Publishers Inc., Hanover, MA, USA; 2009.

[4] Geoffrey E. Hinton , Simon Osindero , and Yee-Whye Teh. *A Fast Learning Algorithm for Deep Belief Nets* Neural Computation 2006 18:7, 1527-1554

[5] Yann LeCun, Yoshua Bengio, Geoffrey Hinton *Deep learning*, *Nature*, Vol. 521, No. 7553. 27 May 2015, pp. 436-444, doi:10.1038/nature14539.

[6] *Multi-access Edge Computing*, ETSI, http://www.etsi.org/technologies-clusters/technologies/multi-access-edge-computing.

[7] M. Abadi, A. Agarwal *et al.*, *Tensorflow: Large-scale machine learning on heterogeneous distributed systems*, arXiv:1603.04467, 2016.

[8] Building Mobile Apps with TensorFlow https://www.tensorflow.org/mobile/

[9] Ali Sharif Razavian, Hossein Azizpour, Josephine Sullivan, and Stefan Carlsson. 2014. *CNN Features Off-the-Shelf: An Astounding Baseline for Recognition.* In Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW '14). IEEE Computer Society, Washington, DC, USA, 512-519. DOI=http://dx.doi.org/10.1109/CVPRW.2014.131

[10] Nicholas D. Lane and Petko Georgiev. 2015. *Can Deep Learning Revolutionize Mobile Sensing?*. In Proceedings of the 16th International Workshop on Mobile Computing Systems and Applications (HotMobile '15). ACM, New York, NY, USA, 117-122. DOI=http://dx.doi.org/10.1145/2699343.2699349

[11] S. Bhattacharya and N. D. Lane, *From smart to deep: Robust activity recognition on smartwatches using deep learning*, 2016 IEEE International Conference on Pervasive Computing and Communication Workshops (PerCom Workshops), Sydney, NSW, 2016, pp. 1-6. doi: 10.1109/PERCOMW.2016.7457169

[12] ETSI - European Telecommunications Standards Institute www.etsi.org/

[13] OpenFog Consortium Architecture Working Group, *OpenFog Architecture Overview*,http://www.openfogconsortium.org/wp-content/uploads/OpenFog-Architecture-Overview-WP-2-2016.pdf, 2016.

[14] D. P. Anderson and G. Fedak, *The Computational and Storage Potential of Volunteer Computing*, Cluster Computing and the Grid, 2006. CCGRID 06. Sixth IEEE International Symposium on, Singapore, 2006, pp. 73-80. doi:10.1109/CCGRID.2006.101

[15] A. Mukherjee, H. S. Paul, S. Dey and A. Banerjee, *ANGELS for distributed analytics in IoT*, 2014 IEEE World Forum on Internet of Things (WF-IoT), Seoul, 2014, pp. 565-570. doi:10.1109/WF-IoT.2014.6803230

[16] M. Satyanarayanan, *The Emergence of Edge Computing,* in Computer, vol. 50, no. 1, pp. 30-39, Jan. 2017.doi:10.1109/MC.2017.9

[17] S. Teerapittayanon, B. McDanel and H. T. Kung, *Distributed Deep Neural Networks Over the Cloud, the Edge and End Devices,* 2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS), Atlanta, GA, 2017, pp. 328-339. doi:10.1109/ICDCS.2017.226

[18] Pushkara Ravindra, Aakash Khochare, Siva Prakash Reddy, Sarthak Sharma, Prateeksha Varshney, Yogesh Simmhan, *ECHO: An Adaptive Orchestration Platform for Hybrid Dataflows across Cloud and Edge*, arXiv:1707.00889, 2017.

[19] P. H. Tsai, H. J. Hong, A. C. Cheng and C. H. Hsu, *Distributed analytics in fog computing platforms using tensorflow and kubernetes,* 2017 19th Asia-Pacific Network Operations and Management Symposium (APNOMS), Seoul, Korea (South), 2017, pp. 145-150. doi: 10.1109/APNOMS.2017.8094194

[20] Bo Tang, Zhen Chen, Gerald Hefferman, Tao Wei, Haibo He, and Qing Yang. 2015. *A Hierarchical Distributed Fog Computing Architecture for Big Data Analysis in Smart Cities*, In Proceedings of the ASE BigData & SocialInformatics 2015 (ASE BD& SI '15). ACM, New York, NY, USA, Article 28, 6 pages. DOI: https://doi.org/10.1145/2818869.2818898.

[21] Docker, https://www.docker.com/

[22] Lichman, M. (2013). UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science.

[23] B. Bruno, *et al. Analysis of human behavior recognition algorithms based on acceleration data*. In ICRA 13, https://pdfs.semanticscholar.org/f614/b897b68a22ed9885bad5e6de3aa9b3d33213.pdf.