DLion: Decentralized Distributed Deep Learning in Micro-Clouds

Rankyung Hong
University of Minnesota

Abhishek Chandra
University of Minnesota

Abstract

Deep learning is a popular technique for building models from large quantities of input data for applications in many domains. With the proliferation of edge devices such as sensor and mobile devices, large volumes of data are generated at rapid pace all over the world. Migrating large amounts of data into centralized data center(s) over WAN environments is often infeasible due to cost, performance or privacy reasons. Moreover, there is an increasing need for incremental or online deep learning over newly generated data in real-time. These trends require rethinking of the traditional training approach to deep learning. To handle the computation on distributed input data, *micro-clouds*—small-scale clouds deployed near edge devices in many different locations—provide an attractive alternative for data locality reasons. However, existing distributed deep learning systems do not support training in micro-clouds, due to the unique characteristics and challenges in this environment.

In this paper, we examine the key challenges of deep learning in micro-clouds: computation and network resource heterogeneity at inter- and intra micro-cloud levels and their scale. We present *DLion*, a decentralized distributed deep learning system for such environments. It employs techniques specifically designed to address the above challenges to reduce training time, enhance model accuracy, and provide system scalability. We have implemented a prototype of *DLion* in TensorFlow and our preliminary experiments show promising results towards achieving accurate and efficient distributed deep learning in micro-clouds.

1 Introduction

Micro-clouds [4, 11–13, 42, 44, 47] are an emerging type of infrastructure to support the exponentially growing large amounts of data generated by edge devices such as surveillance cameras [21, 34], mobile phones [33, 35, 58], or various sensors [32, 42, 53]. Individual micro-clouds consist of a small or medium number of servers. Instead of storing and maintaining the huge amounts of data in a few monolithic public clouds, users and organizations can use micro-clouds across multiple locations, deployed close to edge devices for providing faster services with minimum latency.

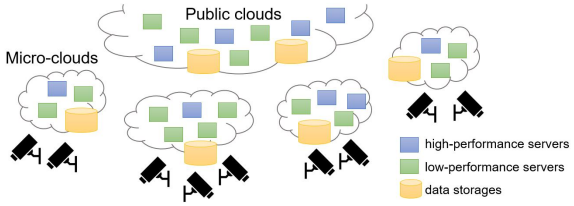


Figure 1: Distributed deep learning (DL) in micro-clouds at multiple locations.

Deep learning (DL) is a popular technique to build models from large quantities of input data for applications in many domains [3, 7, 14, 26, 48, 56]. Traditionally, DL models are trained on large quantities of data assembled in cluster or data center environments. With recent advances in deep learning techniques, continuously generated data could be used for online-learning or incremental-learning [5, 27, 36, 37, 39, 41, 55]. Thus, instead of being a one-time training solution for a fixed set of training data, DL models could keep evolving using data continuously generated from a large number of edge devices across the globe. However, migrating such large amounts of data into centralized cloud(s) over WAN environments for training is likely to be prohibitive due to cost, performance or privacy reasons. For instance, such data is hard to move because of WAN bandwidth constraints, or because it could contain a lot of personal information such as pictures or videos generated by user devices or recorded using surveillance cameras as shown in Figure 1. The need for geo-distributed data analysis has also been shown for many other analytics tasks [17, 20, 25, 38, 49, 50].

An attractive alternative is to *carry out distributed deep learning across the micro-clouds*, since they often provide (limited) computation and storage capabilities. Recently, there has been growing interest in using the edge for DL *inference* [2, 22, 30, 52], where models trained in the cloud are deployed at the edge for faster inference. In this paper, we argue for the use of micro-cloud environments for DL *training*

to efficiently build DL models in-situ and to support online and incremental learning.

There are three major challenges of distributed deep learning in micro-clouds.

1. Compute resource heterogeneity. Different micro-clouds can have different number of servers equipped with different performance hardware. In addition, servers in the micro-clouds can be shared by other applications, so the available compute capacity may dynamically change.

2. Network resource heterogeneity. Servers in a micro-cloud communicate with each other over LAN, whereas servers in different micro-clouds are connected via WAN. Network capacities in LANs may vary due to network resource contention with other applications, while bandwidths in WANs are much more scare and fluctuating than in LANs.

3. Scale. A micro-cloud covers much smaller area than a public cloud, but the number of micro-clouds is much bigger than the number of data centers in a multi-DC public cloud. The micro-clouds are also likely to be much more geographically distributed, leading to heavy communication over WANs.

Existing distributed deep learning systems, however, do not fully address these challenges. General purpose distributed DL systems like TensorFlow [1], MXNet [6] or CNTK [43] do not consider the heterogeneity or scale, resulting in much longer training time due to network bottleneck issue as cluster size increases. Recent research [19, 31, 54] has addressed the network bottleneck issue by reducing the amounts of data transmitted over the network. As a result, models can be trained faster, potentially at the cost of loss of model accuracy as cluster size increases. Other recent research [18, 23, 24] tackles scalability issue by communicating with a small number of peers, but it does not consider network or compute resource heterogeneity. Thus, none of the existing systems comprehensively considers all the challenges of micro-cloud environments: compute and network heterogeneity and scale.

In this paper, we present *DLion*, a decentralized distributed deep learning system that is designed for deep learning in large-scale heterogeneous environments such as micro-clouds. The goals of the system are to reduce training time, improve model accuracy, and handle system scalability. It employs techniques specifically designed to address the above challenges, including compute capacity-aware batching, network-aware data exchange, and selective data propagation (§ 3). We have implemented a prototype of *DLion* on top of TensorFlow and present our preliminary results in § 4.

2 Background and Related Work

Deep Learning. We consider supervised learning using mini-batch stochastic gradient descent (SGD) [40] to minimize the loss value of the function f over the training dataset x (eq. 1).

$$\text{Learning:} \quad \min_{x \in R^n} f(x;w) = \frac{1}{m}\sum_{i=1}^{m} f_i(x;w_t) \tag{1}$$

A deep learning model consists of a set of parameters called *weights*, and operators. The meaning of training a DL model is to find the best values for the weights, which lead to the smallest loss value.

$$\text{Gradient Calculation:} \quad g_t = \frac{1}{m}\sum_{i=1}^{m} \bigtriangledown_w f_i(x;w_t) \tag{2}$$

$$\text{Weight Update:} \quad w_{t+1} = w_t - \eta g_t \tag{3}$$

The weights are tuned by iterations of *gradient* g_t calculation (eq. 2) and *weight* w_t update (eq. 3) over minibatches. A *minibatch* is composed of m training data samples from the training data x and *batch size* indicates the size of a minibatch. An *iteration* indicates a cycle of gradient calculation and weight update over a minibatch. An *epoch* indicates a set of iterations trained over one pass of the whole training data. Batch size and learning rate η are tunable model parameters.

Distributed Deep Learning. Weight update follows eq. 4 in distributed deep learning systems.

$$w_{t+1} = w_t - \eta \frac{1}{k}\sum_{j=1}^{k}\frac{1}{m}\sum_{i=1}^{m} \bigtriangledown_w f_i(x;w_t) \tag{4}$$

k workers calculate their own gradients locally based on a minibatch size of m in parallel. Weights are updated based on the average of the k gradients, where the total batch size of the model is $m * k$.

Distributed Deep Learning Systems. Distributed deep learning systems allow users to train their DL models using a cluster of multiple machines where training data are distributed. General purpose DL systems [8] like TensorFlow [1], MXNet [6] or CNTK [43] utilize central components called *parameter servers* (PS) for weight updates in a *centralized* manner. However, in such a centralized architecture, PSs can be communication bottleneck. *Decentralized* distributed DL systems [18, 23, 24, 31] such as Ako [54] synchronize models without PSs. Workers exchange their local gradients with each other, and update their local weights based on the collected gradients. The workload imposed on PSs can be offloaded to all the workers. *Hybrid* distributed DL systems such as Gaia [19] employ the decentralized architecture to exchange gradients between PSs over WANs while learning in a centralized manner in LANs. While some of the existing systems such as Gaia and Ako have addressed network bottleneck issue by exchanging small amount of gradients or sending full gradients to a subset of peers, none of them comprehensively consider all the challenges in a micro-cloud environment.

3 DLion

We propose *DLion*, a decentralized distributed deep learning system for learning in micro-clouds. Figure 2 shows the system architecture of *DLion*. The philosophy of decentralized architecture fits well to the heterogeneous environments of micro-clouds. In *DLion*, there are no centralized components such as parameter servers. Workers within a micro-cloud are

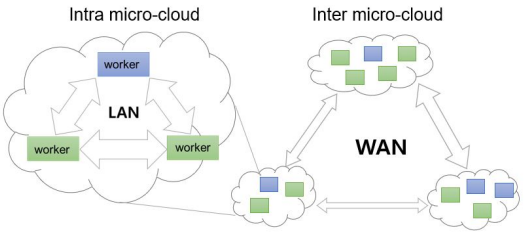


Figure 2: Decentralized Dlion system architecture. Workers in a micro-cloud communicate in LANs, whereas workers in different micro-clouds use limited heterogeneous network bandwidth like WANs. Individual micro-clouds have different number of servers and computation capacity.

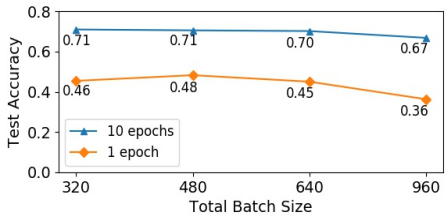


Figure 3: Existence of batch size threshold indicating accuracy plunge between total batch size 640 and 960; 4-worker cluster training with different batch sizes for 10 epochs; The prediction of the threshold can be done with an epoch training.

connected over LANs, and those in different micro-clouds communicate over WANs. There are three major goals in *DLion*, which are reducing training time, improving model accuracy, and providing system scalability for deep learning in micro-clouds. In the rest of the section, we describe the techniques to deal with compute heterogeneity (§ 3.1), network heterogeneity (§ 3.2) and scalability (§ 3.3) encountered in micro-clouds environments. Details of the experimental setup for our exploratory and preliminary experiments are specified in section 4.

3.1 Compute Capacity-aware Batching

We first describe how *DLion* makes use of heterogeneous compute resources in micro-clouds to reduce training time and adaptively increase model accuracy through consideration of available compute resources.

The idea of compute capacity-aware batching is to have different batch sizes assigned to workers based on their computation capacities. For faster learning, *DLion* maximizes data parallelism by assigning a batch size m_j to worker j proportional to its compute capacity C_j, so its time to process a minibatch, $T_j = \frac{m_j}{C_j}$, is close to an expected global unit processing time T_{unit}, thus balancing the load proportionally across workers. This approach could result in very large batch sizes being assigned to fast workers. However, it has been shown that accuracy in large-batch training can degrade drastically beyond a certain batch size [28]. We observed this phenomenon in our experiment shown in Figure 3 where the accuracy drops between total batch size of 640 and 960. To avoid this accuracy degradation, *DLion* makes sure the total batch size across workers $\sum_{j=1}^{k} m_j$ does not exceed a certain threshold T_{bs} beyond which accuracy can drop. *DLion* measures the computation power of each worker through pre-profiling before training, and selects different batch sizes for individual workers accordingly. The new weight update equation based on compute capacity-aware batching (eq. 5) adds two additional constraints as follow:

$$w_{t+1} = w_t - \eta \frac{1}{k} \sum_{j=1}^{k} \frac{1}{m_j} \sum_{i=1}^{m_j} \nabla_w f_i(x; w_t) \quad (5)$$

subject to $|T_j - T_{unit}| \leq \varepsilon$ and $\sum_{j=1}^{k} m_j \leq T_{bs}$.

Table 1: Effect of adaptive learning rate and batch size

Adaptive LB techniques	Accuracy	Training Time
Baseline (SmallLR + StaticBS)	0.6956	2343
LargeLR + StaticBS	0.6256	2404
WarmUpLR + StaticBS	0.7052	2401
WarmUpLR + SpeedUpBS	0.7090	2186
WarmUpLR + IncreaseBS	0.7212	5216
WarmUpLR + SpeedUpBS + DecayLR	**0.7220**	**2183**

Adaptive model parameter tuning. On top of compute capacity-aware batching, *DLion* applies techniques related to large-batch training for better model optimization such as:
warm-up learning rate [15,57] increasing learning rate from η to $\eta * k$ (# workers) early in the learning phase (`WarmUpLR`)
increasing batch size [10, 28] late in learning (`SpeedUpBS`) or throughout learning (`IncreaseBS`)
decaying learning rate [46] late in learning (`DecayLR`).

DLion adaptively adjusts the DL model parameters, learning rate and batch size. There is no comprehensive and integrated existing solution for applying these techniques. Our system automatically applies them by determining when, how, and what to apply the techniques with comprehensive consideration of heterogeneous computation capacities of workers, batch size threshold T_{bs} and learning progress. For example, we make sure the total batch size is not greater than the threshold T_{bs} while increasing `IncreaseBS` or speeding up batch size `SpeedUpBS`. The threshold T_{bs} is approximately 10 percents of training data size [46].

We performed exploratory experiments to see the effect of the techniques. We trained a model stated in § 4 with different combinations of the techniques for 10 epochs. Table 1 shows that the best combination `WarmUpLR + SpeedUpBS + DecayLR` results in faster training time and higher accuracy compared to other baselines that either do not use these techniques, or use them individually or pairwise combinations.

3.2 Network-aware Data Exchange

In this section, we describe how *DLion* deals with heterogeneous network bandwidth resources in micro-clouds to improve model accuracy and achieve faster training times.

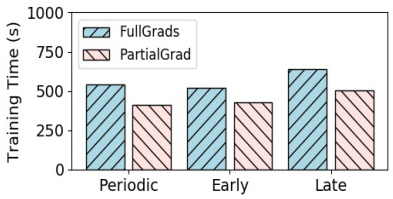


Figure 4: When-to-do weight exchange (WE) for model synchronization; Weights are exchanged every 10 iterations during whole training (`Periodic`), first 2 epochs (`Early`), and last 2 epochs (`Late`). WE early in training achieves comparable result with periodic WE. Partial gradient exchange (`PartialGrads`) reduces training time for all cases. A model is trained until it reaches 60% accuracy.

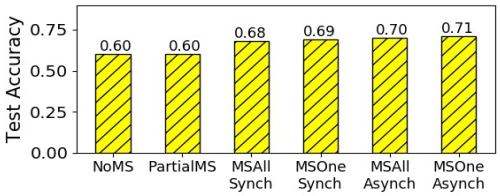


Figure 5: What-to-do, whom-to-do, and how-to-do model synchronization. Partial weight exchange does not help to improve accuracy. Rather, full weight exchange is much more effective in model optimization. A model is trained with partial gradient exchange and periodic WE for 30 minutes.

The distributed DL model synchronization happens by exchanging data between workers during the training phase from-time-to-time. There are two types of data, gradients and weights, that workers can exchange for model synchronization. Existing decentralized systems like Ako and Gaia exchange only gradients, which can result in longer training time and accuracy drop as cluster size increases. Recent work [51] has proposed a periodic weight exchange algorithm to compensate for drop in accuracy.

DLion employs this direct model synchronization across workers though weight exchange (WE) in addition to gradient exchange (GE). The key idea of network-aware data exchange is to adjust data size by controlling the quality of data and considering the available network bandwidth of individual workers in real-time. We explore several decisions to understand factors contributing to the data dissemination, as follows. **When-to-do**: when to exchange data, e.g., more frequently early or late in training, or periodically; **what-to-do**: whether to exchange whole or partial data; **whom-to-do**: whether to exchange data with all workers or a subset of workers; and **how-to-do**: whether to exchange data synchronously or asynchronously.

DLion uses a system parameter to control the contribution to the model update when adjusting data size, especially for gradient exchange (GE). *DLion* increases the contribution threshold to reduce the size of partial gradients, but still to convey the important information by partial gradients. When workers need to send gradients over WANs, the threshold is set higher, compared to sending them over LANs.

Figure 4 and Figure 5 show exploratory results for the four decisions regarding weight and gradient exchanges. Figure 4 shows that frequent WE early in learning has a comparable performance with periodic WE with high frequency. Also, partial GE helps to reduce the training time, compared to full GE. *DLion* uses these insights to allocate more resources for model synchronization at the beginning of training phase, and concentrate more on gradient exchange later in the training phase. In addition, Figure 5 shows the results regarding what-to-do, whom-to-do, and how-to-do. Model synchronization by WE (`MS`) benefits every case except for partial WE (`PartialMS`). In addition, exchanging weights only to the worst worker having the highest loss value (`One`) and asynchronous WE (`Asynch`) leads to higher accuracy as it was able to more iteration for a given training time because of small amount data exchange.

3.3 Selective Data Propagation

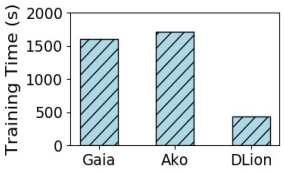
We next discuss how *DLion* can handle system scalability when learning over large number of micro-clouds over WANs. We explicitly consider the compute and network heterogeneity in our scalability approach as well.

In *DLion*, at each iteration, a subset of micro-clouds (senders) send their gradients to a subset of micro-clouds (receivers), instead of an all-to-all communication, where each micro-cloud would broadcast gradients to all micro-clouds. We use a probabilistic model to select senders and receivers based on their compute capacities. Since micro-clouds with higher capacities are more likely to generate more informative gradients over larger minibatches, they have a higher probability of being selected as senders. Similarly, micro-clouds with lower capacities are more likely to generate less informative gradients over smaller minibatches, so they are likely selected as receivers.

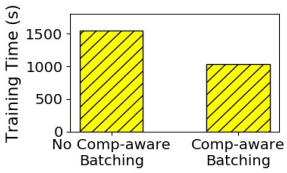
After the sender and receiver selection, we employ our network-aware data exchange techniques (§ 3.2) to reduce the data size. Moreover, the workload of the gradients delivery from a sender micro-cloud to a receiver micro-cloud is offloaded to all the workers in each location, to avoid overloading only a single worker at each location. Finally, we are also considering gossipping algorithms to more efficiently disseminate the data through the network.

4 Evaluation

Implementation. We are implementing a prototype of *DLion* on TensorFlow. Each worker trains DL models by using TensorFlow. Messages are delivered via Redis, an in-memory data store, as a message broker for workers in *DLion*. Redis provides persistence that is necessary for dynamic cluster configurations such as fault tolerance, worker join or leave in the future. Currently, TensorFlow provides a static cluster configuration where it is hard to deal with the cluster dynamics.



(a) Training time comparison. (60% accuracy)



(b) Effect of compute capacity-aware batching. (70% accuracy)

Figure 6: Benefit of handling compute heterogeneity.

Experimental setup. We compared *DLion* with Gaia and Ako implemented in our prototype, and emulated micro-cloud environments using 4 local servers by using the linux `tc` and `stress` commands to throttle network bandwidth, and impose load on servers, respectively. Network links are set to 1Gbps for LANs, and 100 Mbps for WANs. High-performance servers have 24 cpus and low-performance servers have 8 cpus per server. All servers have 60GB available memory and run on Ubuntu 16.04 installed with TensorFlow 1.4.1. A worker runs in a server. Dataset CIFAR10 [29] and a test model (2conv+2fc, model size is 17MB) are used for the purpose of preliminary experiments. The test model with CIFAR10 is converged after 10-epoch training, and takes around 30 minutes in LAN, so we use those two as training termination conditions. We use training time and accuracy as metrics to measure system performance and model optimization.

4.1 Heterogeneous Computation Resources

We evaluate the usefulness of the compute capacity-aware batching feature and compare *DLion* with existing distributed deep learning systems, Gaia and Ako, in compute resource heterogeneous environments. We set up a cluster where there are three low-performance workers and one high-performance worker, and network resources are homogeneous. Figure 6 shows the training time until reaching a target accuracy for the various systems. We use two different accuracy targets (60% accuracy for Figure 6a and 70% for Figure 6b) as Gaia and Ako were unable to reach 70% accuracy in our experiments. For Gaia, workers exchange significant partial gradients, and for Ako, workers exchange partitioned partial gradients at each iteration. Figure 6a shows the comparison results where *DLion* is able to reach the target accuracy 73% and 74% faster than Gaia and Ako, respectively. This is because both existing systems are unable to take advantage of compute heterogeneity and do not have any features to improve model accuracy like direct model synchronization through weight exchange and adaptive parameter tuning. As shown in Figure 6b, we see that compute capacity-aware batching technique helps to reduce the training time by 34% by fully utilizing the heterogeneous compute resource of the cluster. The high-performance worker was able to train a larger minibatch than the low-performance workers at each iteration. Thus, using compute capacity-aware batching can learn more information faster than without using it.

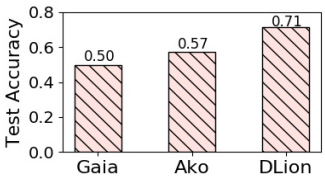


Figure 7: Model accuracy comparison to show the advantage of handling heterogeneous network bandwidth resources.

4.2 Heterogeneous Network Resources

We compare *DLion* using all four techniques with Gaia and Ako in network bandwidth heterogeneous environments. We set up two micro-clouds with three workers in a micro-cloud and a worker in another micro-cloud. All 4 workers have homogeneous compute resources. For Gaia, workers in a micro-cloud train in a centralized manner by exchanging full gradients with a PS. PSs in each micro-clouds exchange significant partial gradients according to its algorithm. For Ako, all workers exchange partitioned gradients determined based on the smallest bandwidth between micro-cloud. The test model is trained for 30 minutes for each of the three systems.

Figure 7 shows that *DLion* can achieve the highest accuracy during the same amount of training time, 42% and 25% higher than Gaia and Ako, respectively. Gaia does not take into account the available network bandwidths when determining the size of gradients. It waits until the significant gradients are delivered over WANs, resulting in more time to finish an iteration. On the other hand, Ako considers the smallest network bandwidth in calculating the size of partial gradients. As a result, it performs more iterations for a given amount of time than Gaia leading to higher accuracy. However, there is inefficiency in network resource utilization of three workers in LAN. If we selected the largest bandwidth for Ako, the accuracy would be lower due to network bottleneck issue in a WAN link between two micro-clouds.

5 Conclusion

There has been increasing need of data analytics based on deep learning in micro-clouds. However existing distributed deep learning systems do not handle the characteristics of micro-clouds environments such as compute capacity and network bandwidth heterogeneity as well as system scalability. In this paper, we have presented *DLion*, a decentralized deep learning system for fast learning and high accuracy in such environments. We have conducted preliminary experiments to show the effect of *DLion* techniques handling the compute and network heterogeneity in micro-clouds. Also, we have discussed design considerations for scalability solution. Our preliminary results are promising, and show the benefit of explicitly addressing the challenges exposed by micro-cloud environments.

Acknowledgment. This work is supported in part by NSF grant III-1422802, CNS-1619254 and CNS-1717834.

6 Discussions and Future Work

In this paper, we presented promising preliminary results of *DLion* based on compute capacity-aware batching, adaptive model parameter tuning, network-aware data exchange features. We are actively developing details of our *DLion*, and plan to open-source the system. While this motivates the research on deep learning in large-scale heterogeneous environments such as micro-clouds, there are several discussion points and open issues:

Large-scale system evaluation. Scale is one of the major challenges in micro-clouds. We plan to incorporate our proposed scalability solutions (§ 3.3) into *DLion*. The experiments in this paper have been performed in a small-scale local cluster with a small size of deep learning model and dataset. We plan to conduct more extensive and thorough experiments in large-scale environments where there are many micro-clouds to evaluate the scalability of *DLion*. We will also use multiple large-scale deep learning models such as VGG16 [45] and ResNet50 [16] and datasets like ImageNet [9]. We expect *DLion* to outperform existing systems with larger models and datasets consuming larger computation and network resources because *DLion* carefully factors heterogeneous resources, the sizes of models and training data in training them.

Training data migration. It is possible that data distribution across the micro-clouds may be skewed and may not match their compute capabilities. For instance, it may happen that micro-clouds with small computation resources have larger volumes of training data compared to other micro-clouds with large computation resources. This may require migrating training data across micro-clouds for better load balancing. We will investigate the tradeoff of input data migration vs. load balancing in future work. For instance, one approach could be to balance the size of training data for individual micro-clouds depending on their relative computation capacities by migrating small portions of training data to the closest large-capacity micro-clouds.

Dynamic environments. We would like to cover dynamic environments where compute and network resources are dynamically changing over time to show the adaptability of *DLion*. Our system and model parameters need to be adjusted based on the changing cluster resources. We plan to exploit how the system dynamically adjusts the parameters based on environmental changes. In addition, we will put more efforts in the assessment of the prediction accuracy based on profiling and the effectiveness of runtime parameter adjustment.

Fault tolerance and cluster dynamics Workers may join, leave, or fail during training, or micro-clouds may get disconnected. To support such scenarios, we plan to explore a fault tolerance and cluster dynamics features in our system. We have already separated the message passing module from the training core module based on TensorFlow to implement the feature. We will continue to study on this to support worker failure recovery and cluster dynamics in *DLion*.

Edge devices We assume workers as machines always powered on. If various edges devices like mobile devices connected to micro-clouds are equipped with powerful computation components, we can consider the trade-off between available power (energy) of the devices and system performance in terms of training time and test accuracy.

References

[1] M. Abadi et al. Tensorflow: a system for large-scale machine learning. In *OSDI*, volume 16, pages 265–283, 2016.

[2] Muhammad Ali, Ashiq Anjum, M Usman Yaseen, A Reza Zamani, Daniel Balouek-Thomert, Omer Rana, and Manish Parashar. Edge enhanced deep learning system for large-scale video stream analytics. In *2018 IEEE 2nd International Conference on Fog and Edge Computing (ICFEC)*, pages 1–10. IEEE, 2018.

[3] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[4] Kashif Bilal, Osman Khalid, Aiman Erbad, and Samee U Khan. Potentials, trends, and prospects in edge technologies: Fog, cloudlet, mobile edge, and micro data centers. *Computer Networks*, 130:94–120, 2018.

[5] Francisco M Castro, Manuel J Marín-Jiménez, Nicolás Guil, Cordelia Schmid, and Karteek Alahari. End-to-end incremental learning. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 233–248, 2018.

[6] Tianqi Chen, Mu Li, Yutian Li, Min Lin, Naiyan Wang, Minjie Wang, Tianjun Xiao, Bing Xu, Chiyuan Zhang, and Zheng Zhang. Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems. *arXiv preprint arXiv:1512.01274*, 2015.

[7] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

[8] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Andrew Senior, Paul Tucker, Ke Yang, Quoc V Le, et al. Large scale distributed deep networks. In *Advances in neural information processing systems*, pages 1223–1231, 2012.

[9] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. 2009.

[10] Aditya Devarakonda, Maxim Naumov, and Michael Garland. Adabatch: Adaptive batch sizes for training deep neural networks. *arXiv preprint arXiv:1712.02029*, 2017.

[11] Yehia Elkhatib, Barry Porter, Heverson B Ribeiro, Mohamed Faten Zhani, Junaid Qadir, and Etienne Rivière. On using micro-clouds to deliver the fog. *IEEE Internet Computing*, 21(2):8–15, 2017.

[12] Keke Gai, Meikang Qiu, Hui Zhao, Lixin Tao, and Ziliang Zong. Dynamic energy-aware cloudlet-based mobile cloud computing model for green computing. *Journal of Network and Computer Applications*, 59:46–54, 2016.

[13] Nelson Mimura Gonzalez, Walter Akio Goya, Rosangela de Fatima Pereira, Karen Langona, Erico Augusto Silva, Tereza Cristina Melo de Brito Carvalho, Charles Christian Miers, Jan-Erik Mångs, and Azimeh Sefidcon. Fog computing: Data analytics and cloud distributed processing on the network edges. In *2016 35th International Conference of the Chilean Computer Science Society (SCCC)*, pages 1–9. IEEE, 2016.

[14] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[15] Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017.

[16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[17] Benjamin Heintz, Abhishek Chandra, and Ramesh K Sitaraman. Optimizing grouped aggregation in geo-distributed streaming analytics. In *Proceedings of the 24th International Symposium on High-Performance Parallel and Distributed Computing*, pages 133–144. ACM, 2015.

[18] Li-Yung Ho, Jan-Jan Wu, and Pangfeng Liu. Adaptive communication for distributed deep learning on commodity gpu cluster. In *2018 18th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, pages 283–290. IEEE, 2018.

[19] K. Hsieh et al. Gaia: Geo-distributed machine learning approaching lan speeds. In *NSDI*, pages 629–647, 2017.

[20] Anand Padmanabha Iyer, Aurojit Panda, Mosharaf Chowdhury, Aditya Akella, Scott Shenker, and Ion Stoica. Monarch: gaining command on geo-distributed graph analytics. In *10th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 18)*, 2018.

[21] Samvit Jain, Ganesh Ananthanarayanan, Junchen Jiang, Yuanchao Shu, and Joseph Gonzalez. Scaling video analytics systems to large camera deployments. In *Proceedings of the 20th International Workshop on Mobile Computing Systems and Applications*, pages 9–14. ACM, 2019.

[22] Hyuk-Jin Jeong, Hyeon-Jae Lee, Chang Hyun Shin, and Soo-Mook Moon. Ionn: Incremental offloading of neural network computations from mobile devices to edge servers. In *Proceedings of the ACM Symposium on Cloud Computing*, pages 401–411. ACM, 2018.

[23] Zhanhong Jiang, Aditya Balu, Chinmay Hegde, and Soumik Sarkar. Collaborative deep learning in fixed topology networks. In *Advances in Neural Information Processing Systems*, pages 5904–5914, 2017.

[24] Peter H Jin, Qiaochu Yuan, Forrest Iandola, and Kurt Keutzer. How to scale distributed deep learning? *arXiv preprint arXiv:1611.04581*, 2016.

[25] Albert Jonathan, Abhishek Chandra, and Jon Weissman. Multi-query optimization in wide-area streaming analytics. In *Proceedings of the ACM Symposium on Cloud Computing*, pages 412–425. ACM, 2018.

[26] Andrej Karpathy, George Toderici, Sanketh Shetty, Thomas Leung, Rahul Sukthankar, and Li Fei-Fei. Large-scale video classification with convolutional neural networks. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 1725–1732, 2014.

[27] Ronald Kemker and Christopher Kanan. Fearnet: Brain-inspired model for incremental learning. *arXiv preprint arXiv:1711.10563*, 2017.

[28] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima. *arXiv preprint arXiv:1609.04836*, 2016.

[29] A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

[30] He Li, Kaoru Ota, and Mianxiong Dong. Learning iot in edge: deep learning for the internet of things with edge computing. *IEEE Network*, 32(1):96–101, 2018.

[31] Xiangru Lian, Ce Zhang, Huan Zhang, Cho-Jui Hsieh, Wei Zhang, and Ji Liu. Can decentralized algorithms outperform centralized algorithms? a case study for decentralized parallel stochastic gradient descent. In *Advances in Neural Information Processing Systems*, pages 5330–5340, 2017.

[32] Seng W Loke. Supporting ubiquitous sensor-cloudlets and context-cloudlets: Programming compositions of context-aware systems for mobile users. *Future Generation Computer Systems*, 28(4):619–632, 2012.

[33] A Tawalbeh Lo'ai, Rashid Mehmood, Elhadj Benkhlifa, and Houbing Song. Mobile cloud computing model and big data analysis for healthcare applications. *IEEE Access*, 4:6171–6180, 2016.

[34] Yao Lu, Aakanksha Chowdhery, and Srikanth Kandula. Optasia: A relational platform for efficient large-scale video analytics. In *Proceedings of the Seventh ACM Symposium on Cloud Computing*, pages 57–70. ACM, 2016.

[35] Xiao Ma, Chuang Lin, Xudong Xiang, and Congjie Chen. Game-theoretic analysis of computation offloading for cloudlet-based mobile cloud computing. In *Proceedings of the 18th ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, pages 271–278. ACM, 2015.

[36] Anusha Nagabandi, Chelsea Finn, and Sergey Levine. Deep online learning via meta-learning: Continual adaptation for model-based rl. *arXiv preprint arXiv:1812.07671*, 2018.

[37] Mahardhika Pratama, Andri Ashfahani, Yew Soon Ong, Savitha Ramasamy, and Edwin Lughofer. Autonomous deep learning: Incremental learning of denoising autoencoder for evolving data streams. *arXiv preprint arXiv:1809.09081*, 2018.

[38] Qifan Pu, Ganesh Ananthanarayanan, Peter Bodik, Srikanth Kandula, Aditya Akella, Paramvir Bahl, and Ion Stoica. Low latency geo-distributed data analytics. *ACM SIGCOMM Computer Communication Review*, 45(4):421–434, 2015.

[39] Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, Georg Sperl, and Christoph H Lampert. icarl: Incremental classifier and representation learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2001–2010, 2017.

[40] Herbert Robbins and Sutton Monro. A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407, 1951.

[41] Doyen Sahoo, Quang Pham, Jing Lu, and Steven CH Hoi. Online deep learning: Learning deep neural networks on the fly. *arXiv preprint arXiv:1711.03705*, 2017.

[42] Mahadev Satyanarayanan, Zhuo Chen, Kiryong Ha, Wenlu Hu, Wolfgang Richter, and Padmanabhan Pillai. Cloudlets: at the leading edge of mobile-cloud convergence. In *6th International Conference on Mobile Computing, Applications and Services*, pages 1–9. IEEE, 2014.

[43] Frank Seide and Amit Agarwal. Cntk: Microsoft's open-source deep-learning toolkit. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 2135–2135. ACM, 2016.

[44] Usman Shaukat, Ejaz Ahmed, Zahid Anwar, and Feng Xia. Cloudlet deployment in local wireless networks: Motivation, architectures, applications, and open challenges. *Journal of Network and Computer Applications*, 62:18–40, 2016.

[45] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2016.

[46] Samuel L Smith, Pieter-Jan Kindermans, Chris Ying, and Quoc V Le. Don't decay the learning rate, increase the batch size. *arXiv preprint arXiv:1711.00489*, 2017.

[47] Tolga Soyata, Rajani Muraleedharan, Colin Funai, Minseok Kwon, and Wendi Heinzelman. Cloud-vision: Real-time face recognition using a mobile-cloudlet-cloud acceleration architecture. In *2012 IEEE symposium on computers and communications (ISCC)*, pages 000059–000066. IEEE, 2012.

[48] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[49] Raajay Viswanathan, Ganesh Ananthanarayanan, and Aditya Akella. Clarinet: Wan-aware optimization for analytics queries. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI16)*, pages 435–450, 2016.

[50] Ashish Vulimiri, Carlo Curino, Philip Brighten Godfrey, Thomas Jungblut, Konstantinos Karanasos, Jitendra Padhye, and George Varghese. Wanalytics: Geo-distributed analytics for a data intensive world. In *Proceedings of the 2015 ACM SIGMOD international conference on management of data*, pages 1087–1092. ACM, 2015.

[51] Jianyu Wang and Gauri Joshi. Adaptive communication strategies to achieve the best error-runtime trade-off in local-update sgd. *arXiv preprint arXiv:1810.08313*, 2018.

[52] Junjue Wang, Ziqiang Feng, Zhuo Chen, Shilpa George, Mihir Bala, Padmanabhan Pillai, Shao-Wen Yang, and Mahadev Satyanarayanan. Bandwidth-efficient live video analytics for drones via edge computing. In *2018 IEEE/ACM Symposium on Edge Computing (SEC)*, pages 159–173. IEEE, 2018.

[53] Shiqiang Wang, Tiffany Tuor, Theodoros Salonidis, Kin K Leung, Christian Makaya, Ting He, and Kevin Chan. When edge meets learning: Adaptive control for resource-constrained distributed machine learning. In *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*, pages 63–71. IEEE, 2018.

[54] P. Watcharapichat et al. Ako: Decentralised deep learning with partial gradient exchange. In *Proceedings of the Seventh ACM Symposium on Cloud Computing*, pages 84–97. ACM, 2016.

[55] Tianjun Xiao, Jiaxing Zhang, Kuiyuan Yang, Yuxin Peng, and Zheng Zhang. Error-driven incremental learning in deep convolutional neural network for large-scale image classification. In *Proceedings of the 22nd ACM international conference on Multimedia*, pages 177–186. ACM, 2014.

[56] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *International conference on machine learning*, pages 2048–2057, 2015.

[57] Matthew D Zeiler. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.

[58] Chaoyun Zhang, Paul Patras, and Hamed Haddadi. Deep learning in mobile and wireless networking: A survey. *arXiv preprint arXiv:1803.04311*, 2018.