Department of Intelligent Systems Engineering

ScaDLES: Scalable Deep Learning over Streaming data at the Edge

Sahil Tyagi and Martin Swany

Luddy School of Informatics, Computing and Engineering

Indiana University Bloomington, USA

SECTION 1

Introduction

- Smart devices growing exponentially in recent years
- Data captured across different modalities: image, audio, text, etc.

- From privacy and network bandwidth perspective, data should be stored on device only
- The problem is exacerbated on the edge due to **limited storage** and **high streaming rates**!
- Thus, a lot of data is stored transiently that could otherwise be useful as training data for various deep learning tasks

SECTION 2

Challenges in DL training on streaming data

Heterogeneity in device streaming rates

- Distributed synchronous SGD equation: $$w_{t+1} = w_t - \eta \frac{1}{N} \sum_{n=1}^{n=N} \frac{1}{|b_i|} \sum_{i \in b_i} \frac{\partial}{\partial w_t} \mathcal{L}(x_{(i,n)}, w_t)$$

- Each device trains with mini-batch ***b***, but training on streaming data can have different wait times due to different streaming rates

- With streaming rate ***p***, device ***i*** needs to wait ***(b/p)*** seconds!

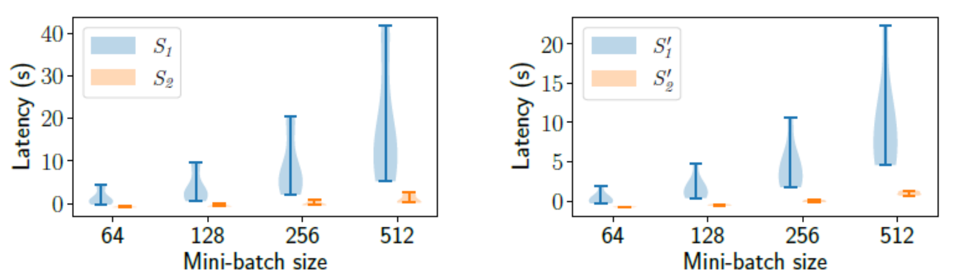
- Variances among device streaming rates becomes a source of heterogeneity; **device with lowest streaming rate thus acts as a straggler**!

- Heterogeneity can be inter or intra-device as well

- We sample device streaming rates from uniform and normal distributions

Distribution	Sample set	Mean	Std. Dev.
Uniform	S_1	38	24
	S_2	300	112
Normal	S_1'	64	24
	S_2'	256	28

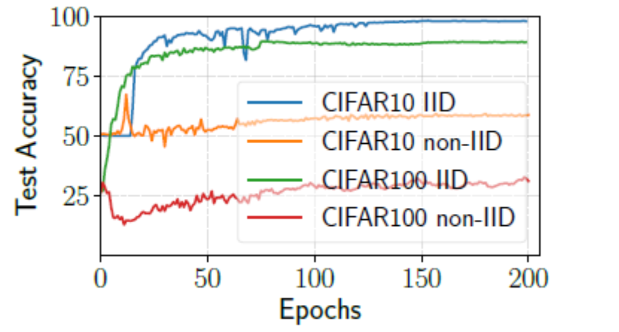
- Sets ***S1*** and ***S1'*** have smaller mean and variance, while ***S2*** and ***S2'*** have higher mean and standard deviation (thus, high stream-rates)



(a) Latency in S_1 and S_2 (b) Latency in S_1' and S_2'

Data skewness and unbalanced data

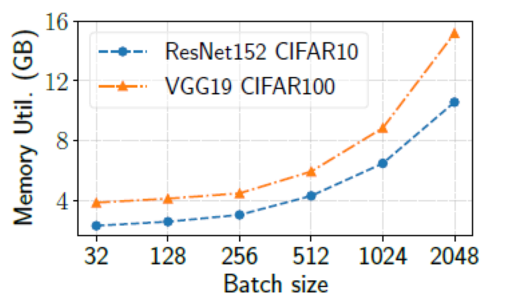
- Data on individual devices can be skewed in volume, properties, or both!
- Skewness occurs when distribution of device-local data varies from overall dist.
- Moving data away to improve IID-ness raises privacy concerns
- ResNet152 and VGG19 trained on CIFAR10/100 on IID vs non-IID data



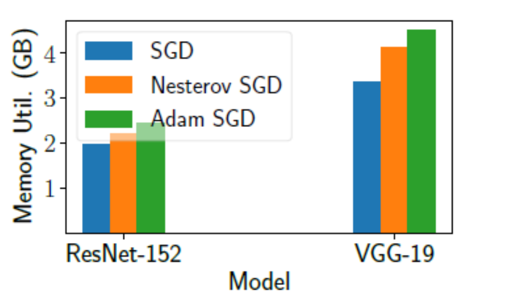
(a) Training with IID and non-IID data

Limited Memory

- GPU/TPU memory is much lower than system memory
- DL training requires storing weights, gradients, activation maps and training batches
- Memory util. increases with batch-size in a near-exponential fashion!
- Memory util. also varies with SGD variant



(b) Memory util. vs. batch-size



(a) Memory varies with SGD variant

Limited Storage

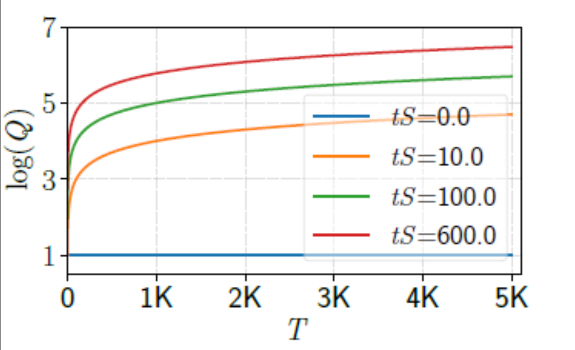
- Difficult to train models on streaming data at ***line-rate;*** *thus, data accumulates when stream rate > processing rate*

- Accumulated buffer size increases over time on account of residual samples from previous timesteps

$$Q_i = (t_i \cdot S_i - b_i) \cdot T + S^{(i)} \quad \forall \quad t_i \cdot S^{(i)} \geq b_i$$

- Assuming high stream-rates and considerable iteration times, buffer size approximates to:

$$Q_i = (T \cdot t_i \cdot S^{(i)} + S^{(i)}) \quad \text{if} \quad (t_i \cdot S^{(i)}) \gg b_i$$



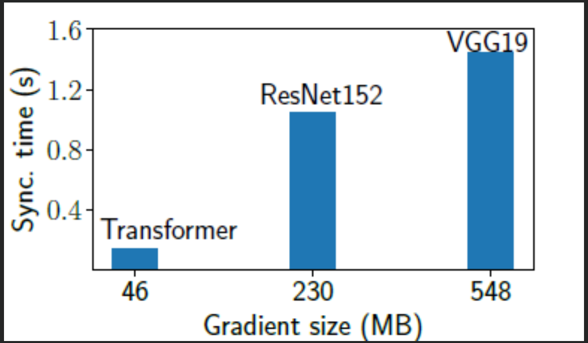
(b) Queue size grows over time

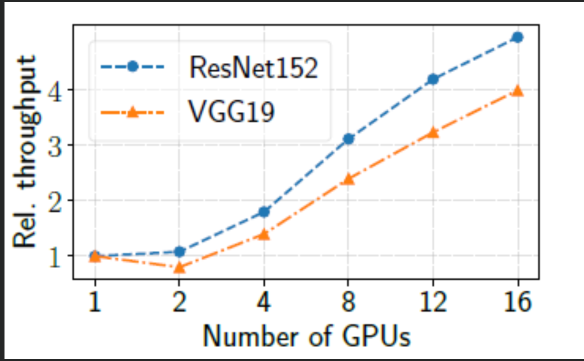
Communication overhead

- Accelerators like GPUs/TPUs bring down computation time

- Gradient synchronization time is considerable in AllReduce due to large model size and limited bandwidth

- Distributed DL scaling still limited by significant gradient sync time; ***adding D devices doesn't increase t/put by D!***





SECTION 3

ScaDLES

Heterogeneous streams

- To eliminate wait times on low-inflow devices, set worker batch-size proportional to its streaming rate

- Due to variable computation on each device, we perform weighted mean

$$r_t^{(i)} = \frac{S_t^{(i)}}{\sum_{j=1}^{n} S_t^{(j)}} \quad : \quad \sum_{j=1}^{n} r_t^{(j)} = 1.0$$

$$\tilde{g}_t = \sum_{j=1}^{n} r_t^{(j)} \cdot g_t^{(j)}$$

$$w_{t+1} = w_t - \eta_{scaled} \cdot \tilde{g}_t$$

- To limit extreme batch sizes in high-streams and degrade generalization, scale the learning rate as well

$$\eta_{scaled} = \gamma_{scaled} \cdot \eta \quad : \quad \gamma_{scaled} = \frac{\sum_{j=1}^{n} S_j}{B}$$

Dealing with limited memory and storage

- Accumulated buffer size can grow quickly due to continuous data streams and considerable iteration times
- By default, data streaming-in is queued until processed successfully: ***Stream Persistence***
- But buffer size grows as O(S*T) after T iterations
- In ***Stream Truncation***, we discard residual samples and hold enough data corresponding to device stream-rate; storage requirement is always O(S) in that case

Dealing with unbalanced and non-IID data

- Training on skewed data degrades model quality as per-device labels are not representative of the overall data distribution
- We add ***randomized data-injection*** to improve data distribution
- Here a fraction of random devices share partial training samples with other devices in the cluster; subset of devices α share fraction β of its streaming data; together, $(\alpha,\ \beta)$ determine what set of devices share how much of their training samples with other devices in distributed training
- Involves trade-off between model quality and privacy risk

Dealing with high communication cost

- Communication overhead is lowered either with low-frequency, high-volume (e.g., FedAvg) or high-frequency, low-volume strategies (e.g., compression)

- ScaDLES applies an adaptive compression technique over **Top-*k*** compression

- Compressed gradients are communicated if variance between compressed and original gradients falls below threshold δ; otherwise, original tensors are sent for weighted AllReduce

$$\mathrm{send}(\mathrm{Top}k(g)) \;\; \text{if} \;\; \frac{||g|^2 - |\mathrm{Top}k(g)|^2|}{|g|^2} \leq \delta \;\; \text{else} \;\; \mathrm{send}(g)$$

SECTION 4

Evaluation

Cluster setup

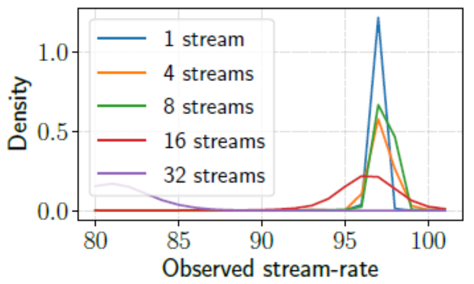
- We simulate streams with Kafka by sampling stream-rates from uniform and normal distributions
- Each training device is spawned as a docker container with 4vCPUs, 12GB system memory and 1 NVIDIA K80 GPU
- Containers communicate on a docker swarm network on 5Gbps network interface

CLUSTER SETUP AND EVALUATION ON IID AND NON-IID DATA

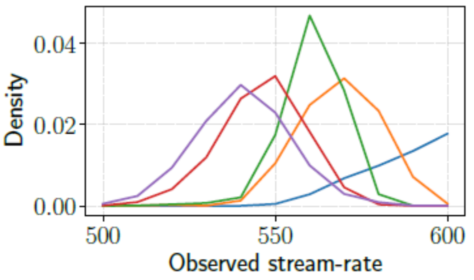
Model	Size	Data	Devices	Label/device
ResNet152	$60.2M$	IID Cifar10	16	10
		nonIID Cifar10	10	1
VGG19	$143.7M$	IID Cifar100	16	100
		nonIID Cifar100	25	4

Simulating streaming data

- A docker container with 16vCPUs and 32GB memory runs Apache Kafka broker and producers
- Container configured with 8 network and 4 IO threads, with 1 partition per topic
- Total topics = total participating devices
- Effective streaming rate could be improved for 600 samples/s by increasing n/w threads and partitions per topic



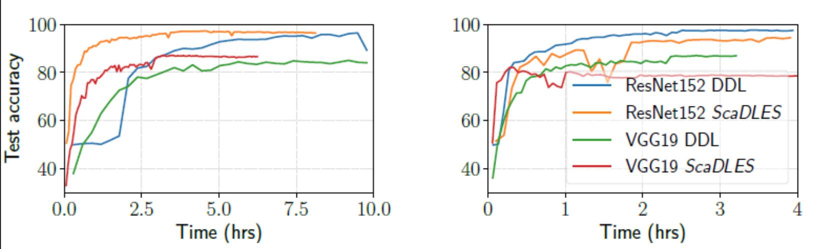
(a) 100 samples/sec



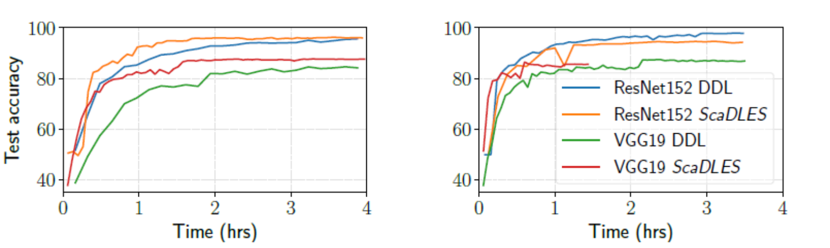
(b) 600 samples/sec

Weighted aggregation in heterogeneous streams

- Comparing ScaDLES with conventional distributed training with per-device mini-batch 64
- ScaDLES converges 3.3x and 1.9x faster under S1; DDL has more accuracy under S2 due to large batches in ScaDLES (4.5K vs. 1K)
- S1’: ScaDLES converges 3.6x and 4x faster



(a) S_1 distribution (b) S_2 distribution



(c) S_1' distribution (d) S_2' distribution

Managing limited memory and storage

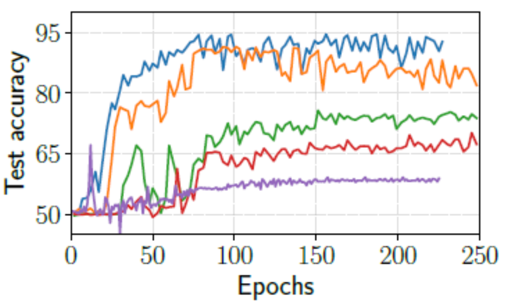
- With *stream persistence*, ScaDLES occupies up to 3.5x less space with S1, 641x less space with S2, 5x with S1' and 42x less space with S2'
- We look at the number of accumulated samples with *persistence* and *truncation* policies
- Each sample is a 32x32 image of size 3Kb

BUFFER-SIZE REDUCTION WITH TRUNCATION POLICY

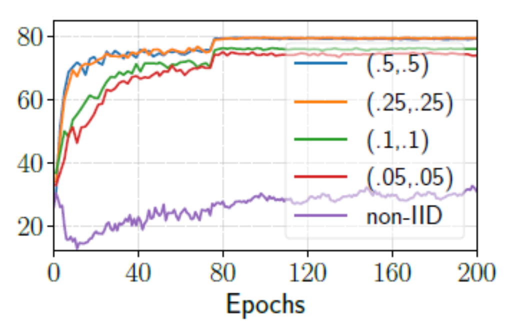
Dist.	Model	Persistence	Truncation	Reduction
S_1	ResNet152	2.9×10^5	129	2238×
	VGG19	1×10^5	118	848×
S_2	ResNet152	4.36×10^6	633	6889×
	VGG19	4×10^6	523	7830×
S_1'	ResNet152	6.2×10^5	143	4340×
	VGG19	3.7×10^5	129	2861×
S_2'	ResNet152	3.6×10^6	384	9429×
	VGG19	2.5×10^6	360	6956×

Data-injection for non-IID data

- We evaluate four sets of $(\alpha,\ \beta)$ parameters:
  - (0.5, 0.5)
  - (0.25, 0.25)
  - (0.1, 0.1)
  - (0.05, 0.05)

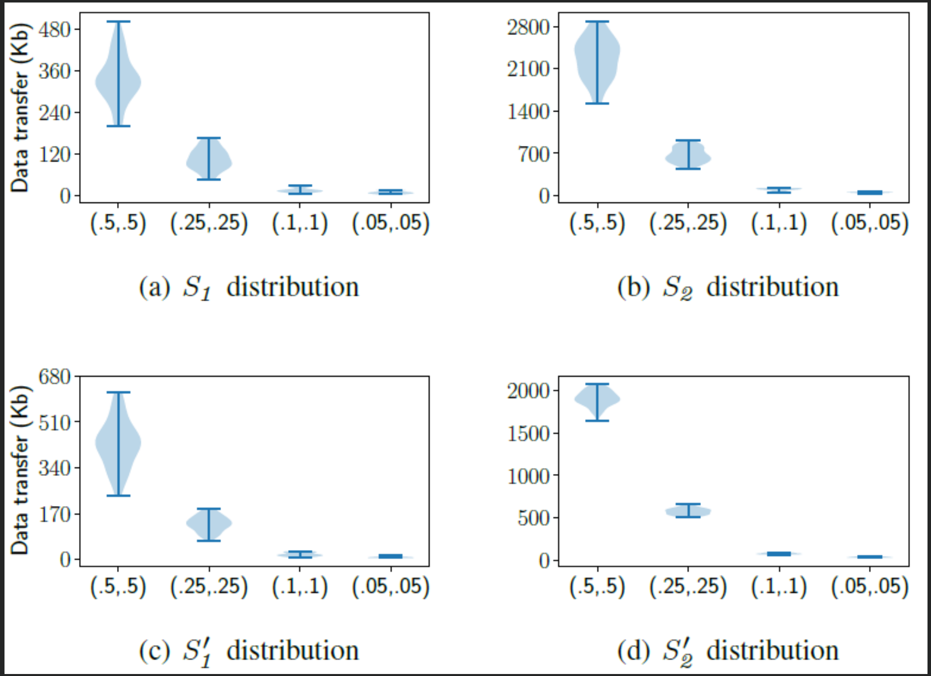


(a) ResNet152



(b) VGG19

Overhead of data-injection strategy



Adaptive compression

- *Compression ratio (CR)* measures the degree of compression; 0.1=10x, 0.01 = 100x
- Using the adaptive compression rule, *CNC* measures the fraction of training iterations using compression to the total iterations

$$\text{CNC ratio} = \frac{T_{compressed}}{T_{compressed} + T_{uncompressed}}$$

COMMUNICATION REDUCTION IN ADAPTIVE COMPRESSION

Model	CR	δ	CNC	Accuracy	Floats sent
ResNet152	0.1	0.1	0.29	97.55%	4.43×10^{11}
		0.2	0.99	96.81%	0.56×10^{11}
		0.3	1.0	98.41%	0.4×10^{11}
		0.4	1.0	98.57%	0.4×10^{11}
	0.01	0.1	0	97.39%	6.02×10^{11}
		0.2	0.17	97.47%	4.99×10^{11}
		0.3	0.43	96.72%	2.56×10^{11}
		0.4	0.99	94.97%	6.32×10^{8}
VGG19	0.1	0.1	0	85.45%	1.3×10^{12}
		0.2	0.08	84.74%	1.19×10^{12}
		0.3	1.0	81.91%	1.3×10^{10}
		0.04	1.0	81.78%	1.3×10^{10}
	0.01	0.1	0	84.68%	1.3×10^{12}
		0.2	0	83.98%	1.3×10^{12}
		0.3	0	83.94%	1.3×10^{12}
		0.4	0.004	84.39%	1.29×10^{12}

Overall performance in ScaDLES

- Comparing ScaDLES with typical DDL w.r.t final accuracy, buffer size reduction and overall speedup

Model	Dist.	Acc. drop	Buffer red. (GB)	Speedup
ResNet152	S_1	-0.06%	0.6	**1.89×**
	S_2	-0.32%	5.9	**1.15×**
	S'_1	-0.13%	0.8	**3.29×**
	S'_2	-0.21%	4.03	**1.42×**
VGG19	S_1	-1.93%	0.26	**1.56×**
	S_2	-4.18%	3.91	**2.83×**
	S'_1	-2.03%	0.35	**2.06×**
	S'_2	-1.59%	2.58	**2.13×**

SECTION 5

Conclusion

- Distributed training over streaming data is challenges by both parallel and systems heterogeneity.
- ScaDLES uses weighted aggregation, stream truncation, randomized data-injection and adaptive compression to accelerate distributed training over streaming data at the edge
- In the best case, ScaDLES converges 3x faster than conventional DDP training while occupying 33% lesser buffer space.
- In the worst case, ScaDLES results in up to 4.18% lesser final accuracy in highly heterogeneous streams due to generalization drop in large-batch training