Hi

CS 744: PYTORCH

Shivaram Venkataraman

Fall 2021

ADMINISTRIVIA

Assignment 2 out! Due Oct 13th early AM! 12th evening

Bid on topics, submit group (1 sentences) – Oct 11 → One week!

Title confirmed – Oct 14

Project Proposal (2 pages) – Oct 25

Introduction

Related Work

Timeline (with eval plan)

WRITING AN INTRODUCTION

Spark

1-2 paras: what is the problem you are solving

How to support interactive data analytics

why is it important (need citations)

1-2 paras: How other people solve and why they fall short

Related work and what is the downside

1-2 paras: How do you plan on solving it and why your approach is better → Your techniques

1 para: Anticipated results or what experiments you will use

RELATED WORK, EVAL PLAN

0.5 page

Group related work into 2 or 3 buckets (1-2 para per bucket)

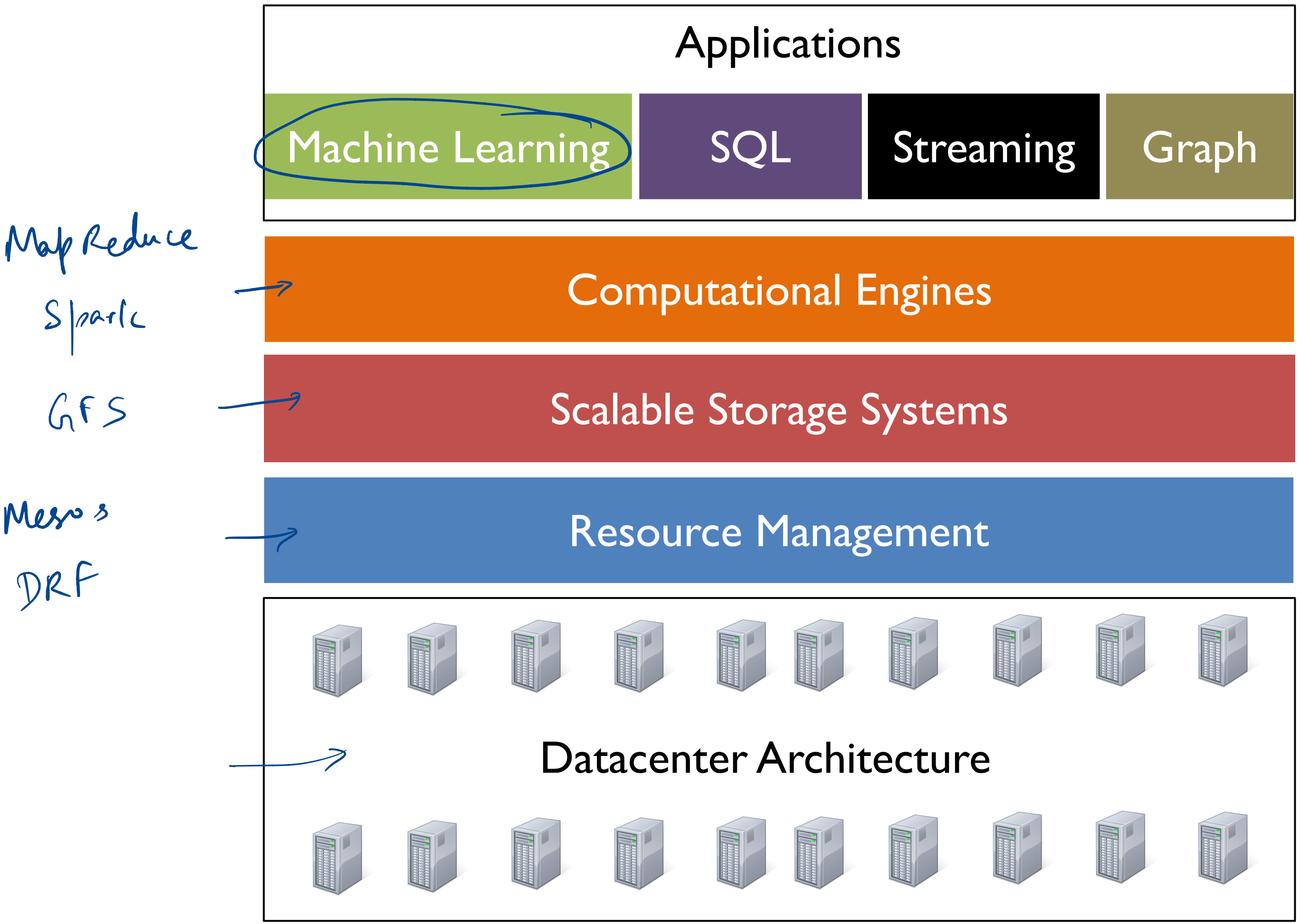
Explain what the papers / projects do → survey of existing work

Why are they different / insufficient

Eval Plan

Describe what datasets, hardware you will use

Available: Cloudlab, Google Cloud (~$150), Jetson TX2 etc.


Applications
Machine Learning
SQL
Streaming
Graph
Computational Engines
Scalable Storage Systems
Resource Management
Datacenter Architecture
MapReduce
Spark
GFS
Mesos
DRF

EMPIRICAL RISK MINIMIZATION

Train a ML model

Supervised learning

data, labels ?

→ avoids overfitting

$$\min_{w \in \mathbb{R}^d} \sum_{i=1}^{N} f(w, z_i) + P(w)$$

Regularization

Function

Model

Data (Examples)

$f(x, w) = \hat{z}_i$

Minimize $\|\hat{z}_i - z_i\|$

$x_i \rightarrow z_i$

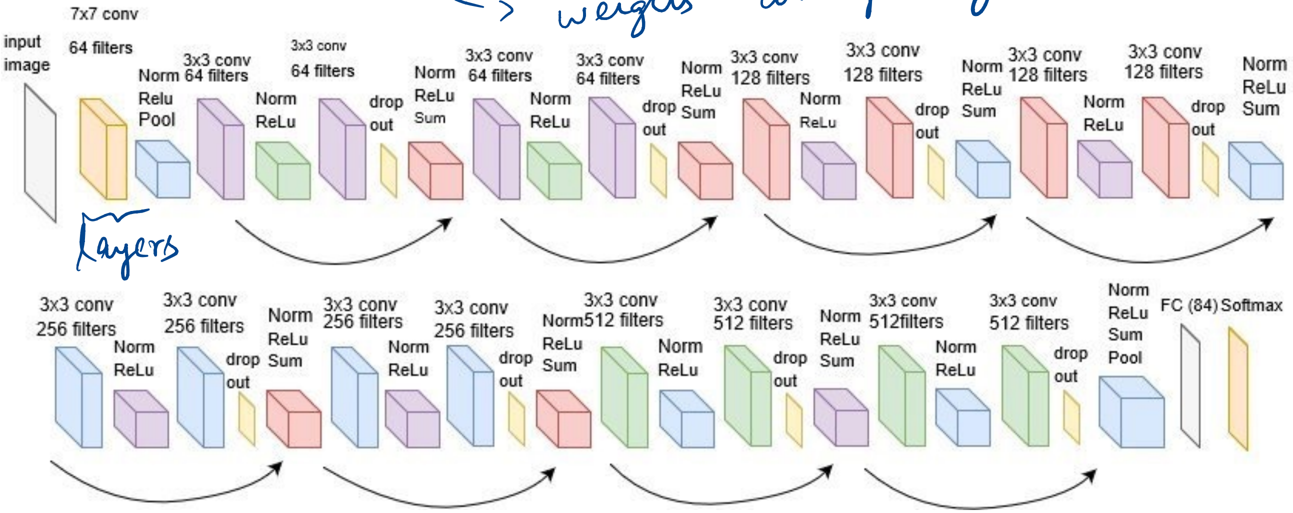
data label

DEEP LEARNING

$l_2(l_1(x_i, \underline{w_1}), \underline{w_2})$ . . . . .

weights corresponding to each layer

ResNet18



layers

- Convolution
- ReLU
- MaxPool
- Fully Connected
- SoftMax

. . .

STOCHASTIC GRADIENT DESCENT

update weights

$$\boxed{w^{(k+1)} = w^{(k)} - \underbrace{\alpha_k}_{\text{learning rate}} \underbrace{\nabla f(w^{(k)})}_{\text{gradient}}}$$

Initialize w ← w_0

For many iterations:

Loss = Forward pass

Gradient = backward pass

Update model

End

$\downarrow$

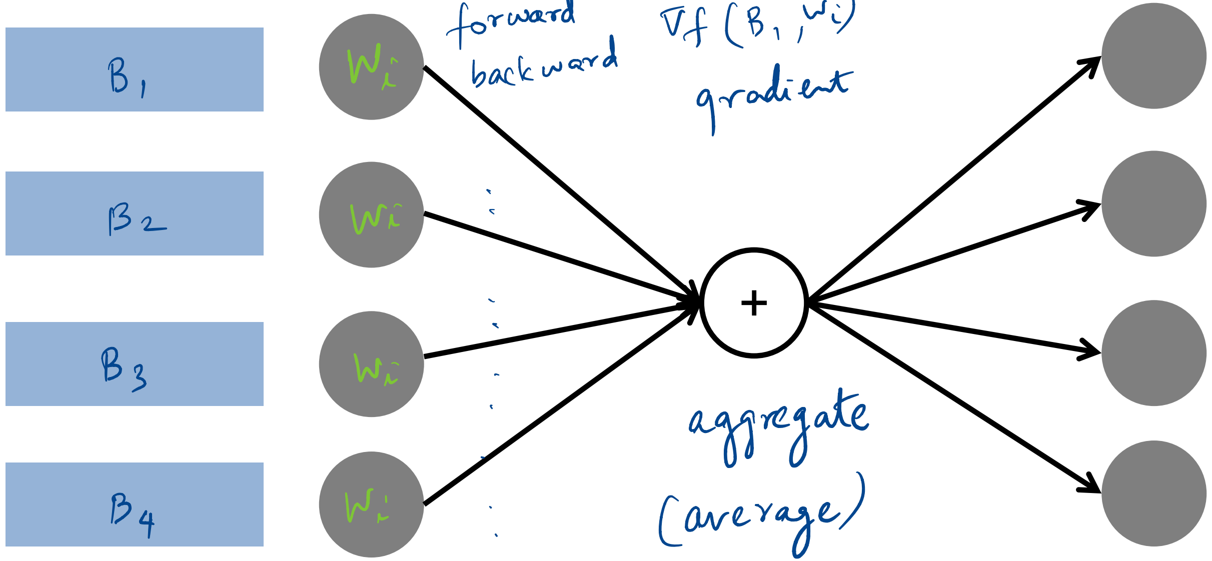
w_{i+1} given w_i and $\nabla f(w_i)$

$f(w_i) \rightarrow$ how close is $f(w_i)$ to labels

$\nabla f(w_i) \rightarrow$ chain rule to compute this

DATA PARALLEL MODEL TRAINING

4 machine



B

forward
backward

$\nabla f(B_1, W_i)$
gradient

aggregate
(average)

to get $\nabla f(B, W_i)$

↳ update the model

Make sure updated model is on all machines

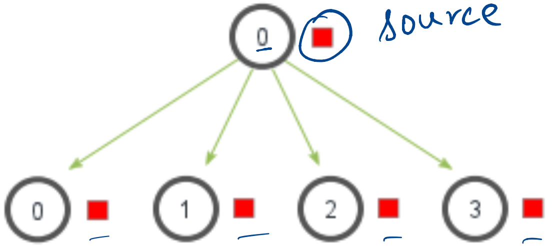
Parameter or Model averaging

COLLECTIVE COMMUNICATION

↳ late 1980s

Broadcast, Scatter

MPI_Bcast

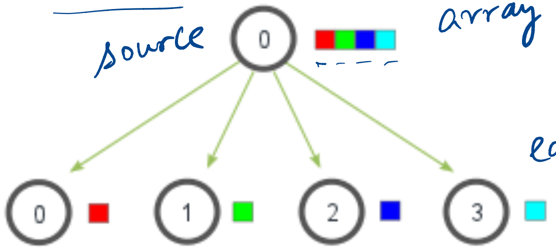


source

destinations

MPI_Scatter

source

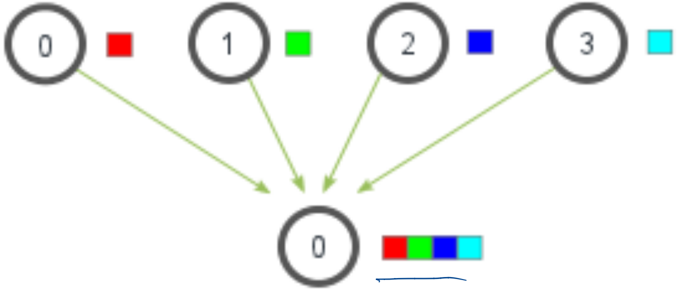


array

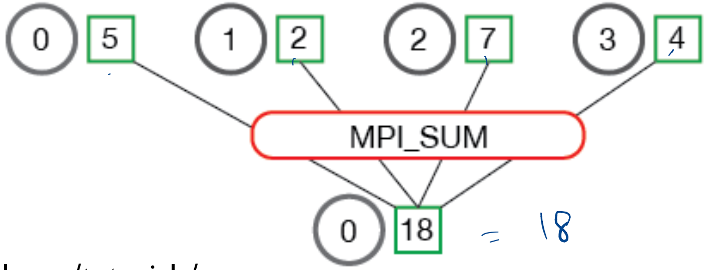
each dest gets part of array

Gather, Reduce

MPI_Gather



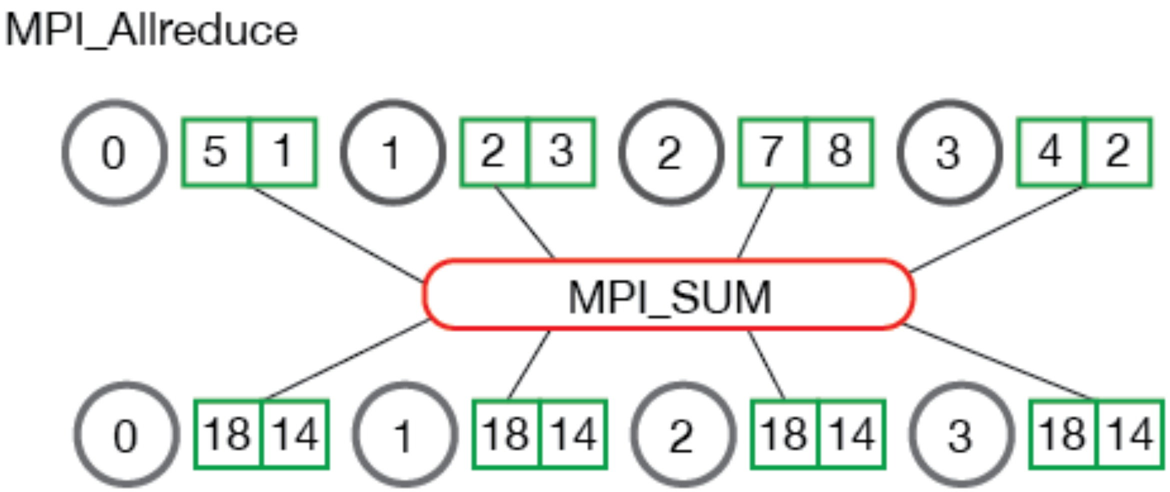
MPI_Reduce



= 18

From https://mpitutorial.com/tutorials/

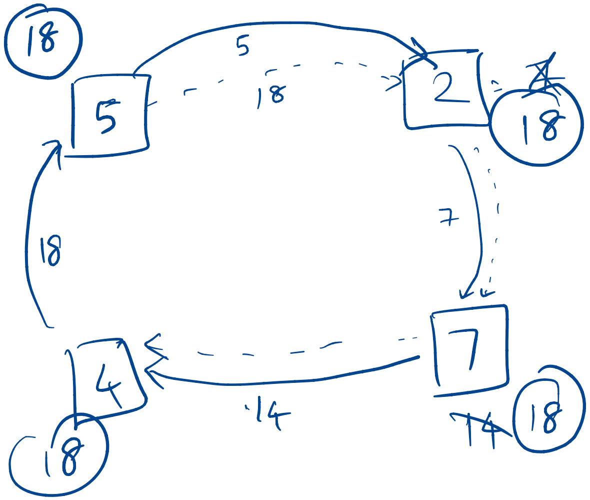
ALL REDUCE USING A RING



reduce, but output of reduce is present on all the machines

down side = latency is high

Ring reduce



A single machine has constant number of messages = better than reduce + broadcast

From https://mpitutorial.com/tutorials/

DISTRIBUTED DATA PARALLEL API

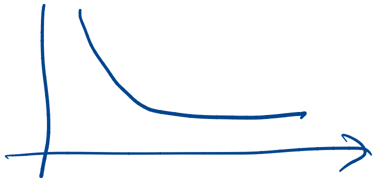
```
# setup model and optimizer
net = nn.Linear(10, 10)
net = par.DistributedDataParallel(net)
opt = optim.SGD(net.parameters(), lr=0.01)

# run forward pass
inp = torch.randn(20, 10)
exp = torch.randn(20, 10)
out = net(inp)

# run backward pass
nn.MSELoss()(out, exp).backward()

# update parameters
opt.step()
```

one line of code change

rest of code resembles single machine code

GRADIENT BUCKETING

Total number = 1B



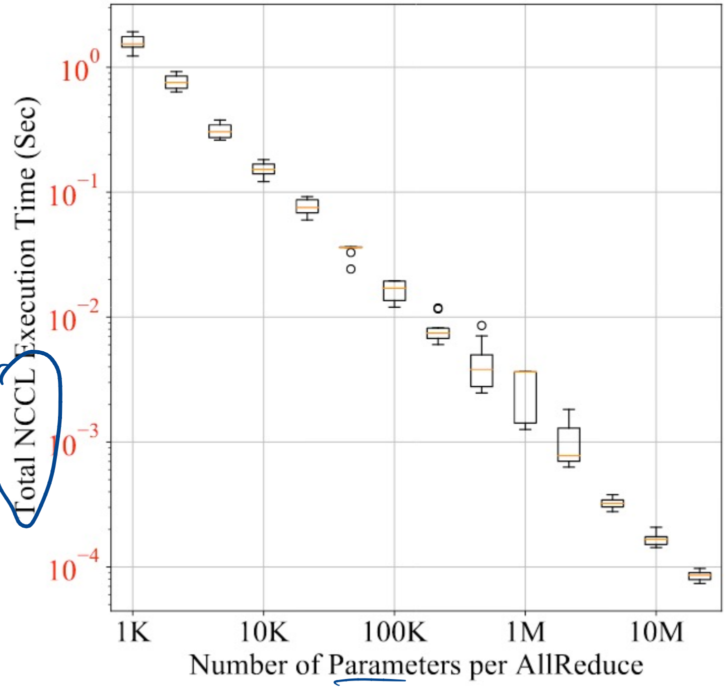
Why do we need gradient bucketing?

↳ When do we do aggregation

↳ larger aggregations → more efficient

Perf. model

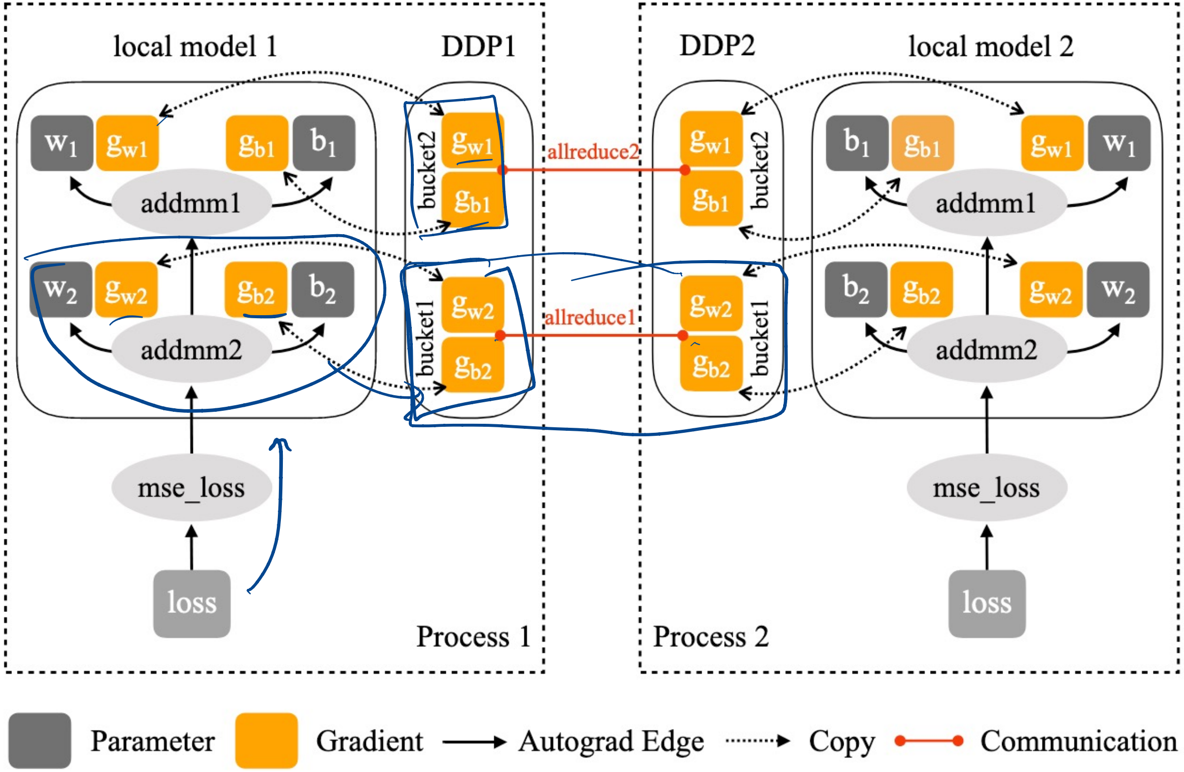
$$\text{Time}_{comm} = \underbrace{\alpha T_{lat}}_{\text{fixed overhead}} + \beta \underbrace{T_{bw}}_{\text{scales with data size}}$$



larger →

GRADIENT BUCKETING + ALL REDUCE

Overlap
- Comm bucket 1 with computation bucket 2



- Bucket fixed size 25 MB
- Put grads into bucket until it fills up

Trigger bucket AllReduce to happen in the background

GRADIENT ACCUMULATION

doesn't do AllReduce
in background

```
ddp = DistributedDataParallel(net)
with ddp.no_sync():
    for inp, exp in zip(inputs, expected_outputs):
        # no synchronization, accumulate grads
        loss_fn(ddp(inp), exp).backward()
# synchronize grads
loss_fn(ddp(another_inp), another_exp).backward()
opt.step()
```

this is when all sync happens.

Use a larger batch than machines can accomodate

IMPLEMENTATION

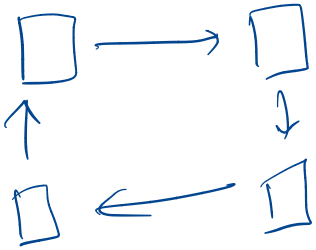
Bucket_cap_mb = Profiling network topology + libraries

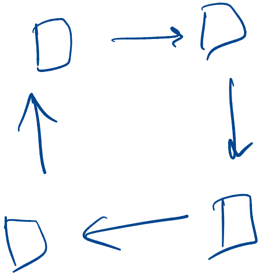
25MB

Parameter-to-bucket mapping → walk back from the end and add layers to buckets

Round-robin ProcessGroups

PCIe





NVLink

Data vs. Compute vs. Communication

Imagenet ~ 100s of GB → 100s of epoch

~ Days

Web search logs ~ 100s of TB

~ hours

SUMMARY

Pytorch: Framework for deep learning

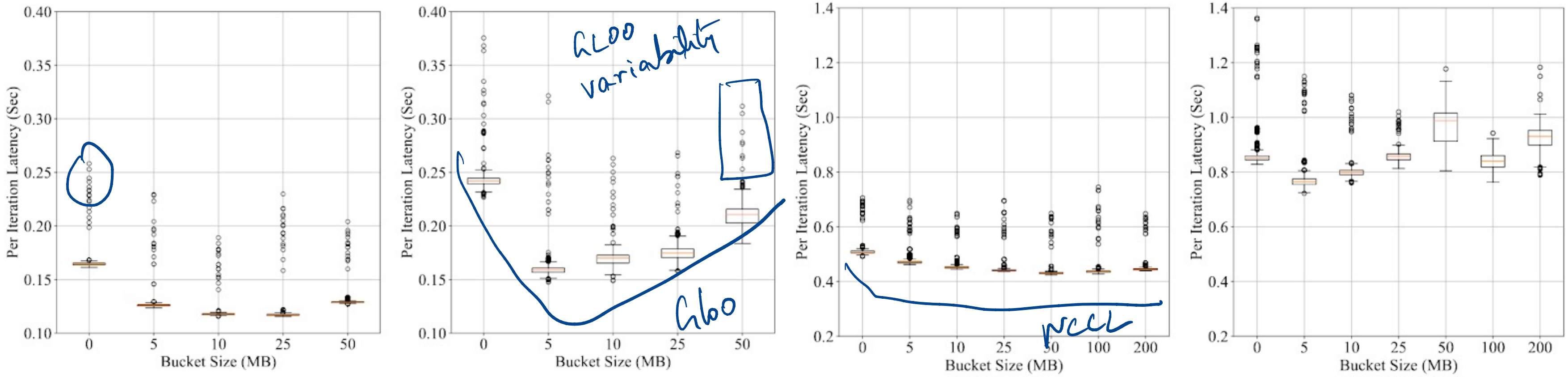
DistributedDataParallel API

Gradient bucketing, AllReduce

Overlap computation and communication

DISCUSSION

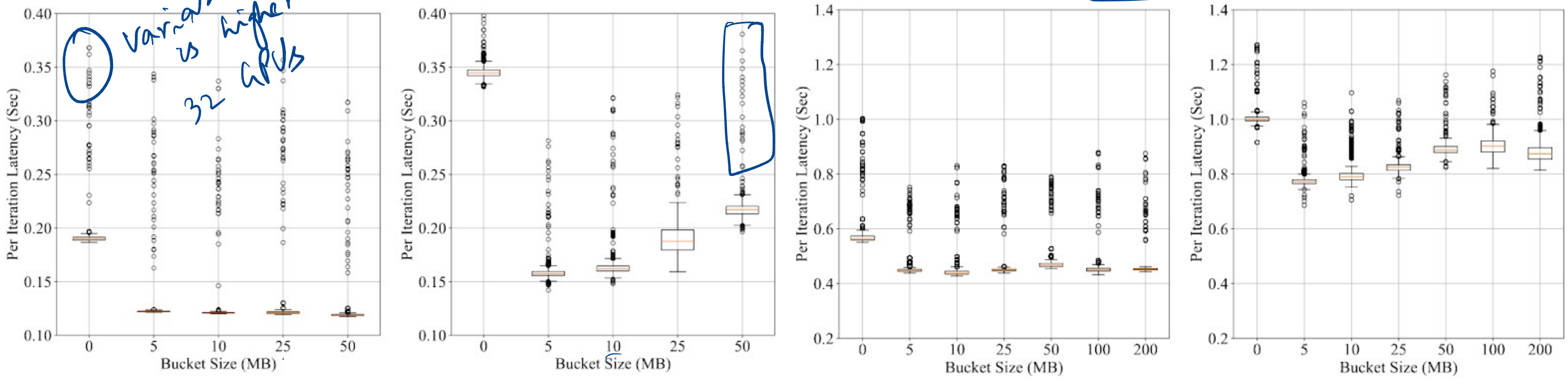
https://forms.gle/YnZC8PKQy1CDFJRf9



(a) ResNet50 on NCCL (b) ResNet50 on Gloo (c) BERT on NCCL (d) BERT on Gloo

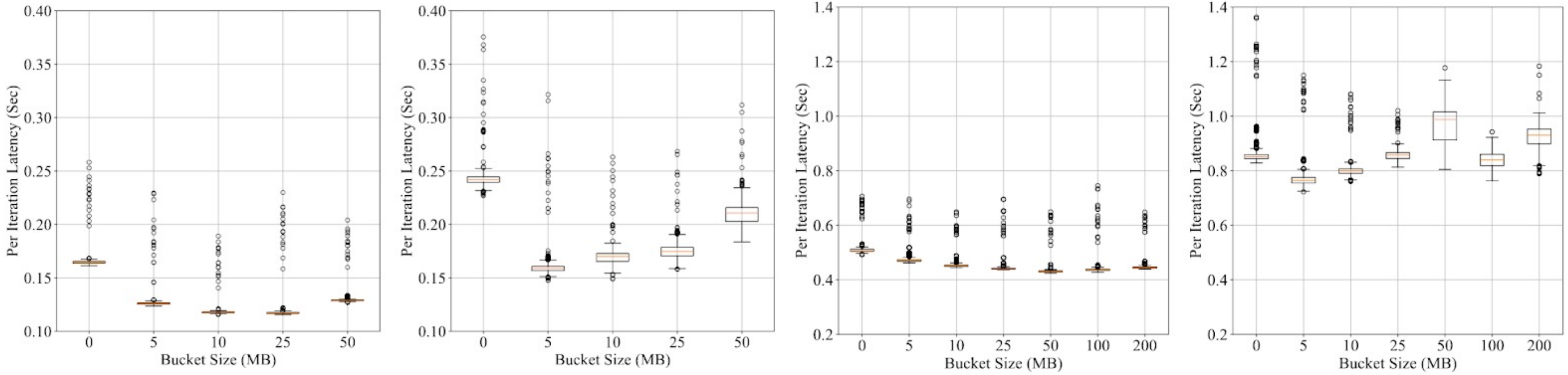
Figure 7: Per Iteration Latency vs Bucket Size on 16 GPUs

Bucket size is steady!



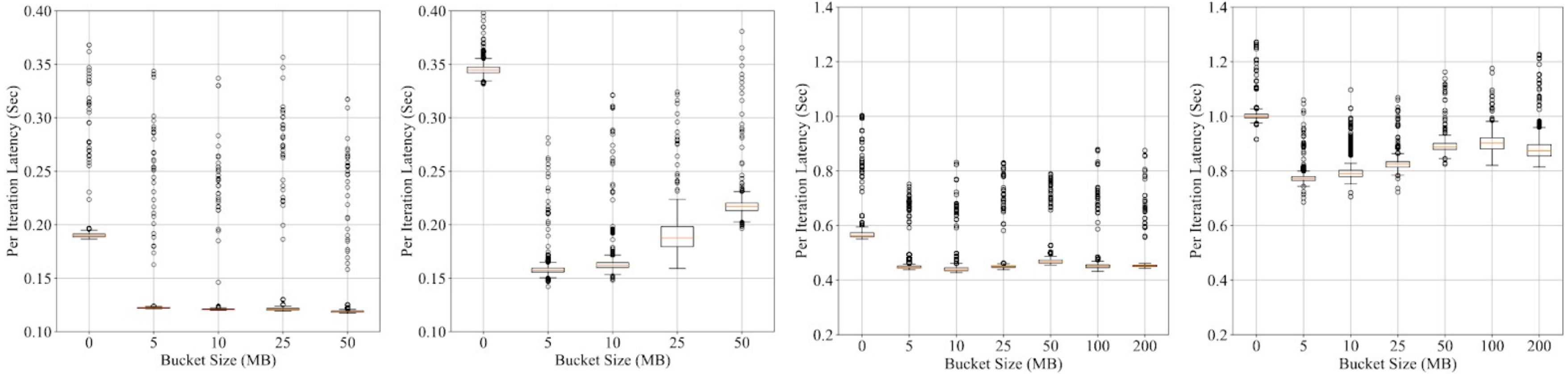
(a) ResNet50 on NCCL (b) ResNet50 on Gloo (c) BERT on NCCL (d) BERT on Gloo

Figure 8: Per Iteration Latency vs Bucket Size on 32 GPUs



(a) ResNet50 on NCCL (b) ResNet50 on Gloo (c) BERT on NCCL (d) BERT on Gloo

Figure 7: Per Iteration Latency vs Bucket Size on 16 GPUs



(a) ResNet50 on NCCL (b) ResNet50 on Gloo (c) BERT on NCCL (d) BERT on Gloo

Figure 8: Per Iteration Latency vs Bucket Size on 32 GPUs

What could be some challenges in implementing similar optimizations for AllReduce in Apache Spark?

- Spark job could have many stages
- Reduction tree + Broadcast
  - ↳ slower than this paper?
- How mutable models can be handled

Spark: Many more tasks than cores

Assuming that all tasks are running at same time

NEXT STEPS

Next class: PipeDream

Assignment 2 is out!

Project Proposal

- Preferences, Groups by Oct 11
- 2 pager by Oct 25

BREAKDOWN

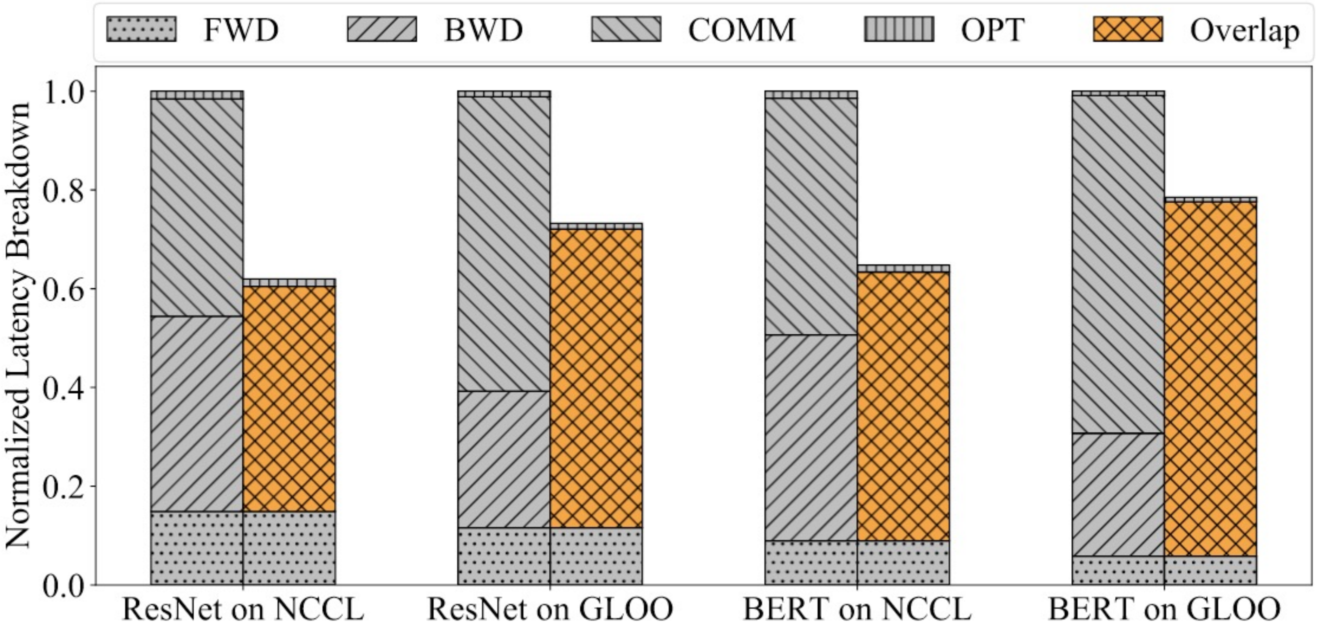


Figure 6: Per Iteration Latency Breakdown