Large-Scale Distributed Sentiment Analysis With RNNs

Group 8: Jianzhun Du, Rong Liu, Matteo Zhang, Yan Zhao

Introduction

- **Project Goal**: to perform sentiment analysis with Recurrent Neural Networks in order to uncover whether a piece of text has positive or negative sentiment
- **The Need for Big Data**: We handle 92.45 GB of 142.8 million reviews
- **The Need for HPC**: it takes 18 hours for the RNN model to run 10 epochs and achieve 80% test accuracy on a single instance of AWS p2.xlarge (Tesla K80).

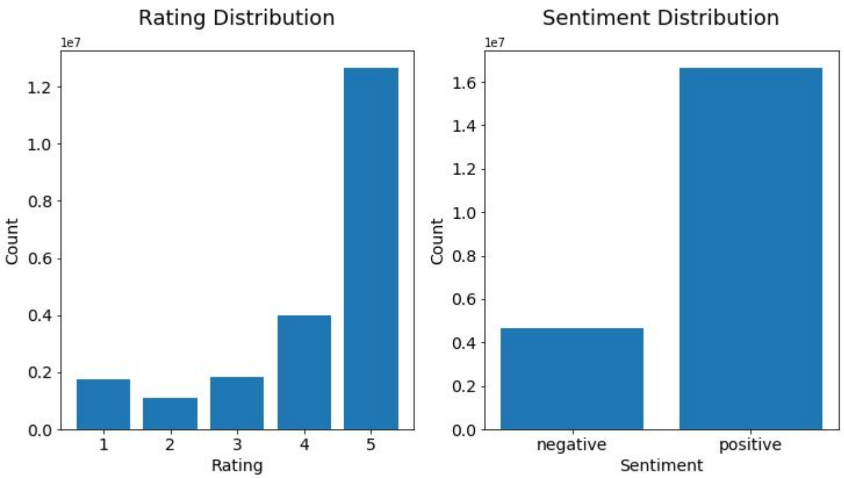
Solution Overview

- **Our Solution**
  - Data Preprocessing: MapReduce
  - Data Storage: HDF5
  - RNN Acceleration: Distributed Workload with Large Minibatch

- **Comparison with Existing Work**
  - Goyal et al. uses similar technique to train ResNet and minimize training time. We parallelize RNN and focus on parallelization tradeoffs instead

I. Data Processing - Data Description

- We borrow raw Amazon product review data, which consists of product reviews along with the ratings, and other information, spanning from May 1996 to July 2014
- We take the ratings as indicators to the underlying sentiments



I. Data Processing - Serial Version

- Extract relevant information
- Remove duplicates of text and keep the mode of the ratings
- Map ratings to binary sentiment indicators
- Map words to numbers
- Truncate or pad text sequences to achieve fixed length

I. Data Processing - Parallelization

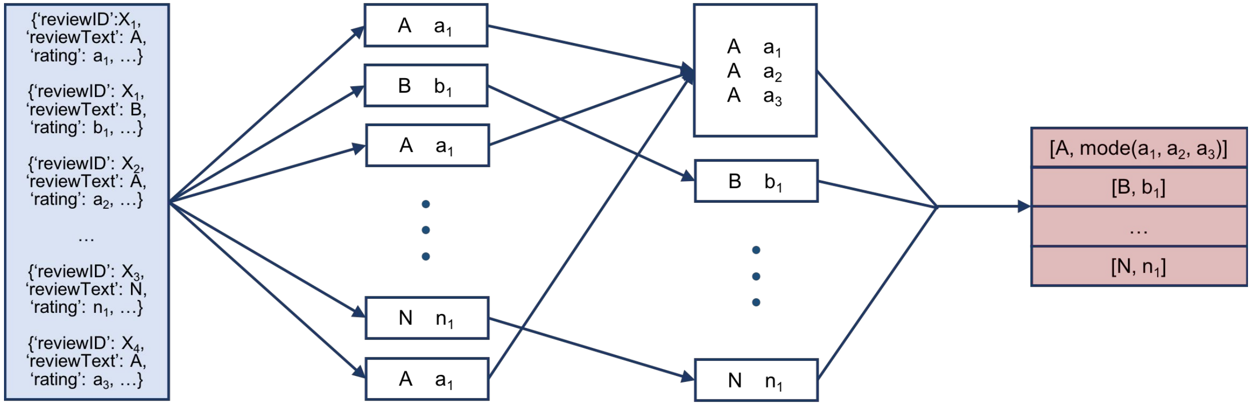
- Extract relevant information

Mapper

- Remove duplicates of text and keep the mode of the ratings
- Map ratings to binary sentiment indicators
- Map words to numbers
- Truncate or pad text sequences to achieve fixed length

Reducer

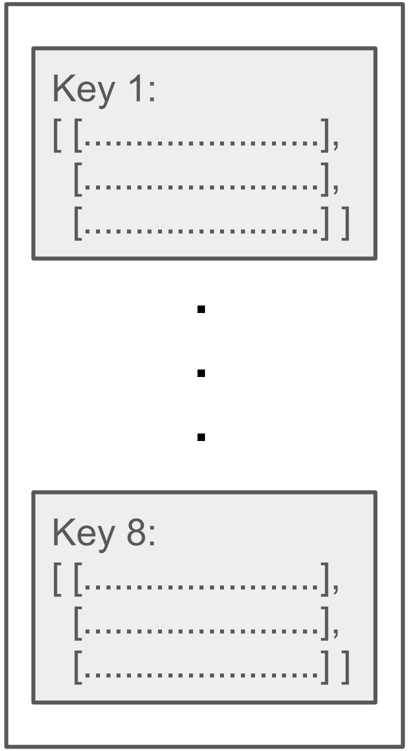
AWS EMR Cluster
of 8 m4.xlarge nodes



I. Data Processing - Data Storage

- Structure:
  - HDF5 File
  - 8 chunked datasets, each containing 2650000 rows of data

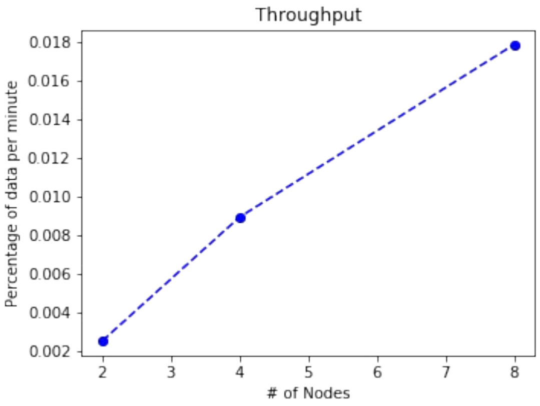
- Benefit:
  - During the training process, after the data loader specifies the dataset key and index number, only that chunk of data is loaded into the memory
  - Balances memory usage and communication overhead

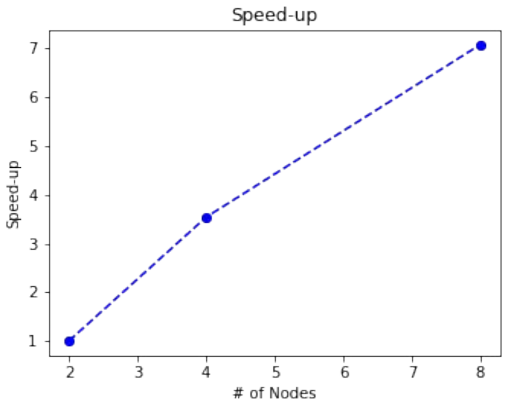


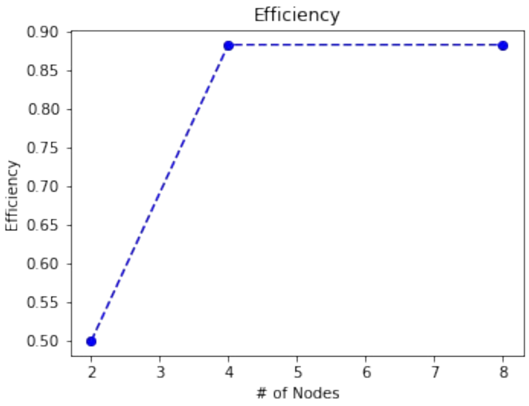
I. Data Processing - Performance

- **Strong Scaling:** Nearly linear speedup
- **Weak Scaling:** unable to investigate

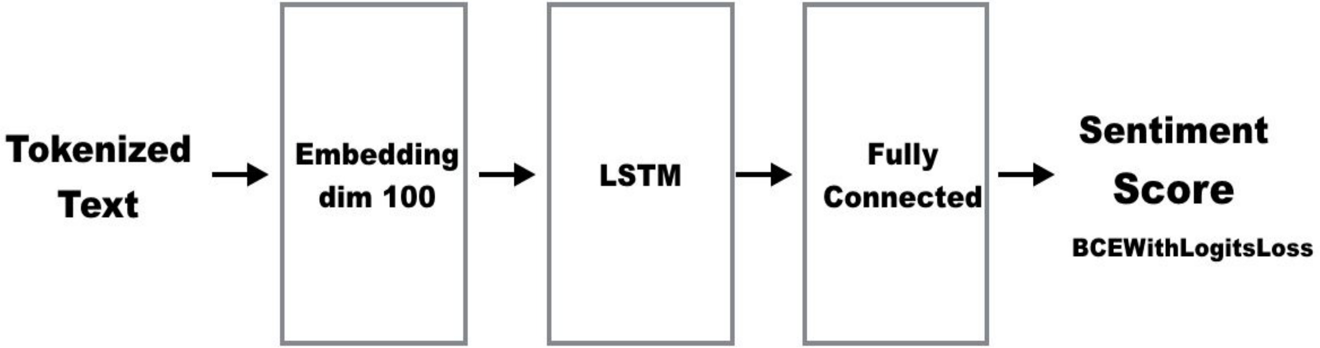
% of Dataset	Runtime (2 Nodes)	Runtime (4 Nodes)	Runtime (8 Nodes)
25%	99 min	28 min	14 min
50%	Failed	51 min	30 min
100%	Failed	Failed	55 min





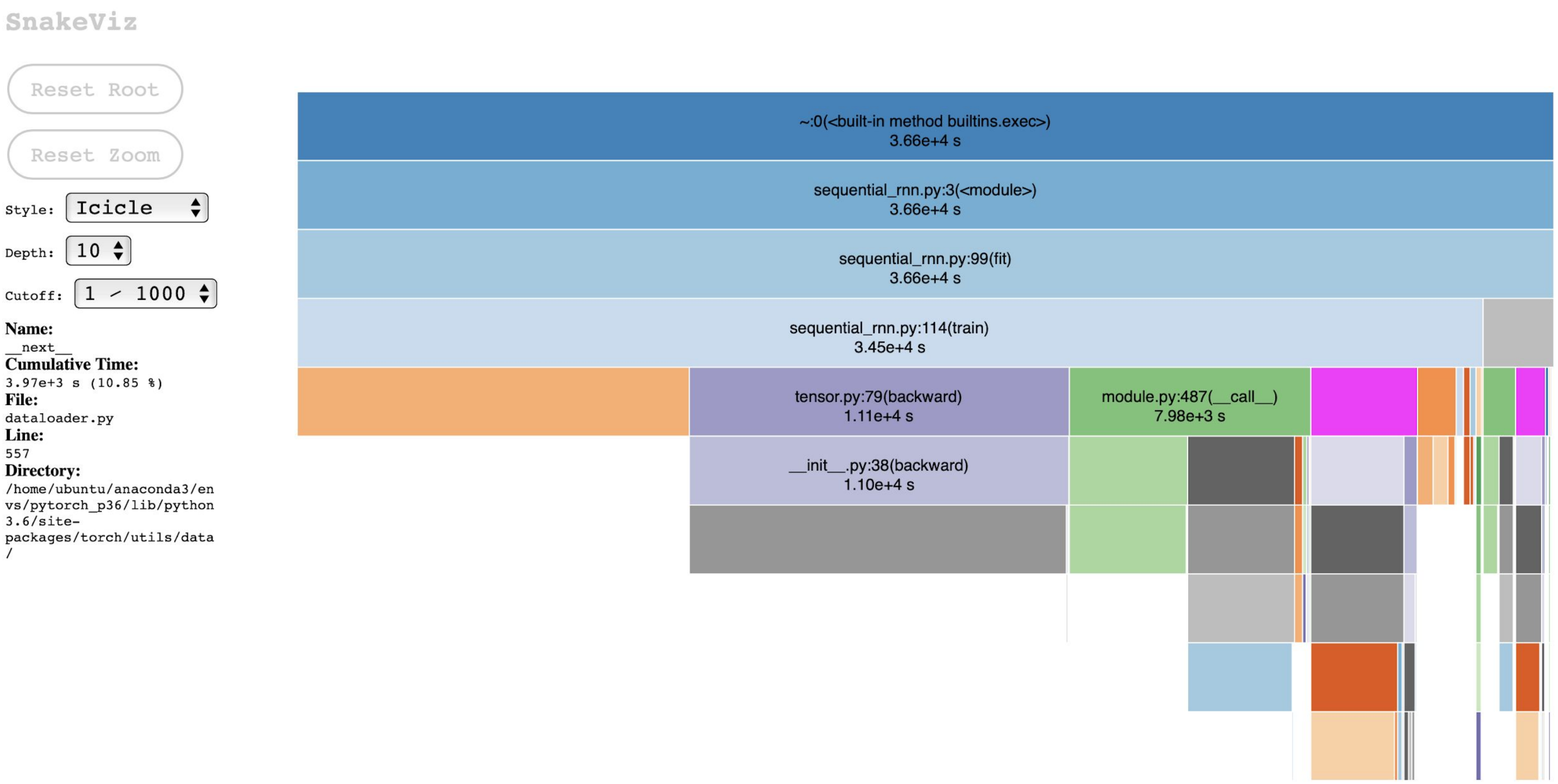


II. RNN+SGD - Serial Version

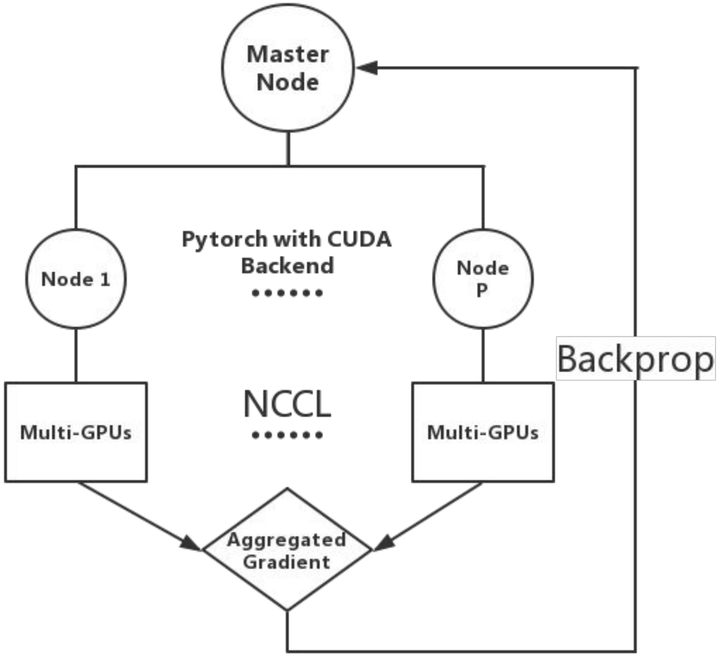


- Compared torch (number input) to torchtext (with text input). Chose torch for lower preprocessing overheads
- Experimented with multiple loss functions to combat class imbalance: weighted cross entropy, MSE, NLL loss, BCEwithLogit Loss, etc.
- Tried differ sentence length to maintain decent performance while keeping training time reasonable. Our current length at 100 words covers about 88% of samples in a subset data.

II. RNN+SGD - Serial Version Profiling



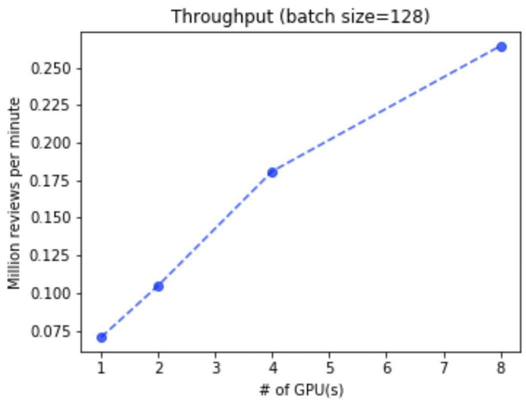
II. RNN+SGD - Parallelization

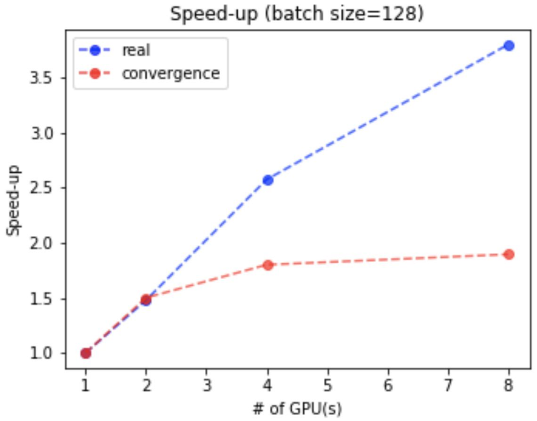


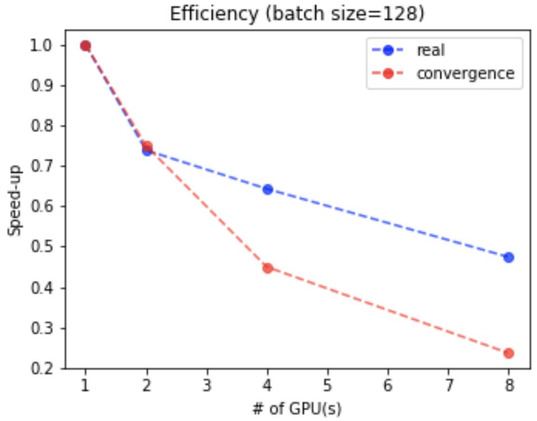
- Parallelize through PyTorch distributed parallel module with CUDA, using MPI-like interface with NCCL backend for inter-GPU communication.
- Single Program Multiple Data
- AWS EC2 - flexible GPUs setup
- Experiment on a different combination of:
  - p2.xlarge (1 NVIDIA Tesla K80 + 4 vCPUs)
  - g3.4xlarge (1 NVIDIA Tesla M60 + 16 vCPUs)
  - g3.16xlarge(4 NVIDIA Tesla M60 + 64 vCPUs)

II. RNN+SGD - Different number of GPUs

- Linear throughput scaling
- Log-linear speedup scaling
- Not strongly scalable

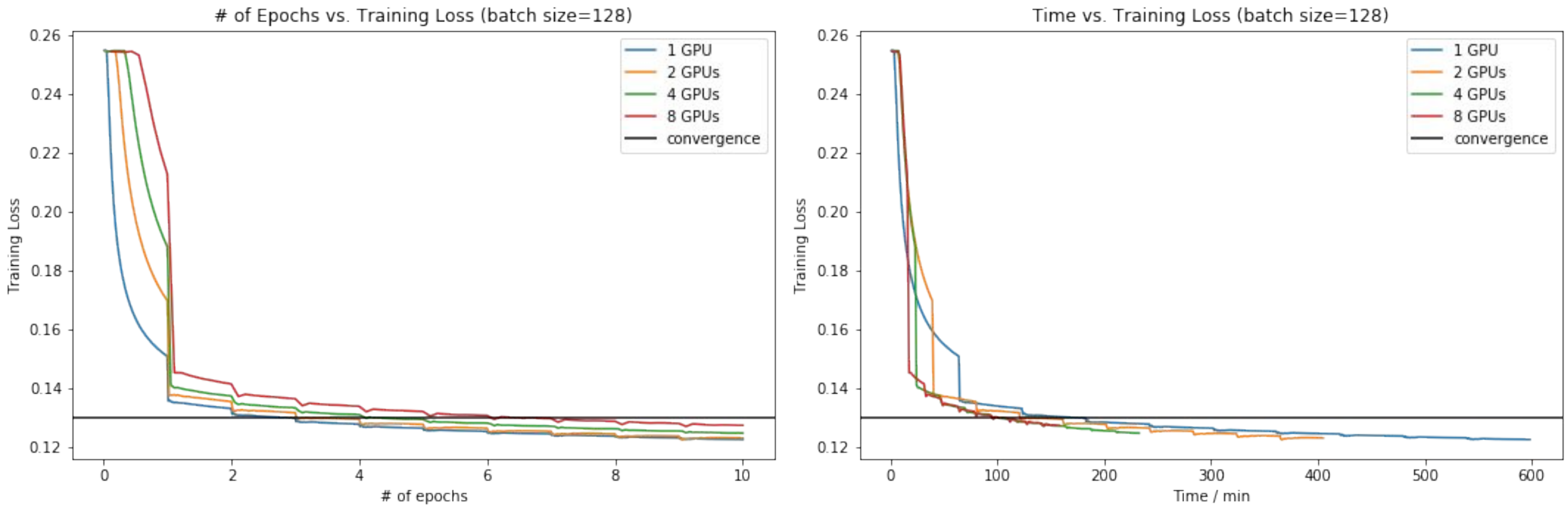






II. RNN+SGD - Distributed RNN Convergence

- The less nodes we use, the higher convergence rate is achieved in terms of the number epochs due to gradient aggregation.
- The model with more GPUs converges faster, which suggests the convergence is accelerated by data parallelism.

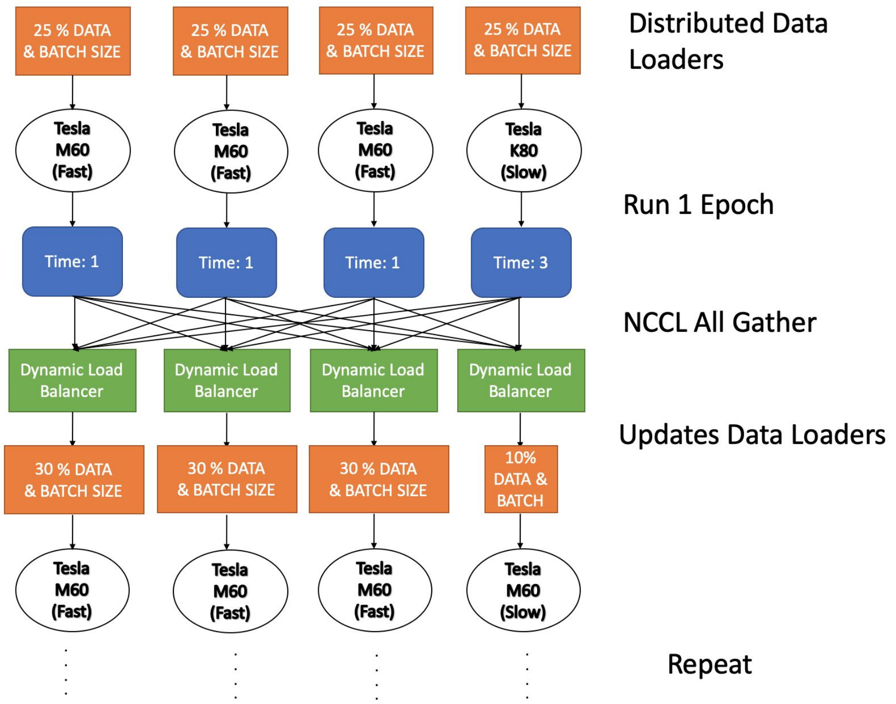


II. RNN+SGD - Different Distributions of GPUs

- Fixed total number of GPUs: 4
- Single node with multiple GPUs is fastest

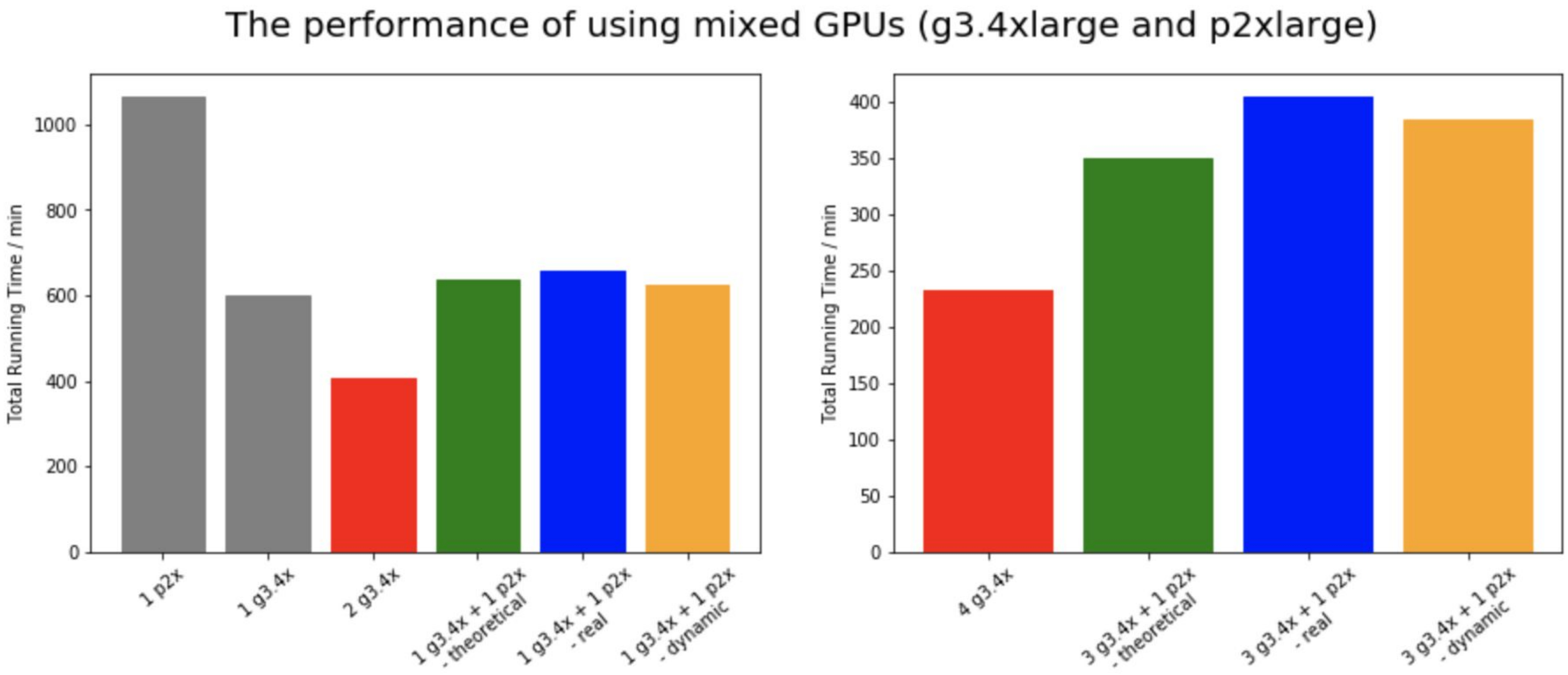
# of Node	# of GPUs per Node	Time (min/epoch)	Speed-up
1	4	21.9	2.73
2	2	26.4	2.27
4	1	23.3	2.57

II. RNN+SGD - Advanced Feature



- Distributed RNN on PyTorch
- Bootstrap actions to install additional softwares on AWS EMR
- Dynamic load balancer that work seamlessly with PyTorch’s Distributed Sampler

II. RNN+SGD - Mixed GPUs Experiment



Money - Speed Tradeoff

Experiment	# p2.xlarge	# g3.4xlarge	# g3.16xlarge	Seconds	Hours	Total Price
Single g3.4xlarge	0	1	0	35920	9.98	11.38
Single p2.xlarge	1	0	0	63923	17.76	15.98
Single g3.16xlarge	0	0	1	13156	3.65	16.64
Two g3.4xlarge	0	2	0	24316	6.75	15.39
Two g3.16xlarge with 2 GPU only	0	0	1	15820	4.39	20.02
Four g3.4xlarge	0	4	0	13973	3.88	17.69
Two g3.16xlarge	0	0	2	9531	2.65	24.17
Three g3.4xlarge + one p2.xlarge	1	3	0	24256	6.74	29.12
Three g3.4xlarge + one p2.xlarge + dynamic	1	3	0	23007	6.39	27.6

Discussion

- Reduced from 18 hours (using p2.xlarge) down to 2.5 hours (2 g3.16xlarge)
- Custom bootstrap actions to install packages on EMR worker nodes
- AWS GPU recommendations:
  - G3.4xlarge is the cheapest option
  - G3.16xlarge is the cheapest option amongst the fastest options (< 5 hours / speedup > 2)
  - Single node with multi-GPUs best value for money
  - Use same GPU model for multi node setup
  - Mitigate load imbalance overhead with dynamic load balancer

Future Work

- Testing different batch sizes
- NFS vs local copies

For citation, please see this page discussion section: https://sophieyanzhao.github.io/discussion