Harvard John A. Paulson
School of Engineering
and Applied Sciences

Parallelization of stock prediction

Group 11: Kevin Hare, Sivananda Rajananda, Junkai Ong

Goals

1. Process scalable quantities of high frequency financial transaction data

2. Utilize multiple nodes and GPU to speed up training of sequential models, which are difficult to parallelize

3. Serve predictions based on trained model in a distributed manner

Existing Work

- LSTMs financial data:
  - Krauss and Fischer (2017)
  - Ghosh et al. (2020) - intraday price movements
  - Lanbouri and Achchab (2020)
    - High resolution prediction, done at 1, 5, and 10 minutes ahead

- Paralellized model training
  - Goyal et al. (2018):
    - Training ImageNet in One Hour - 256 GPUs!
  - Jia et al. (2018):
    - Training ImageNet in **Four Minutes** - Mixed precision, 2048 GPUs!

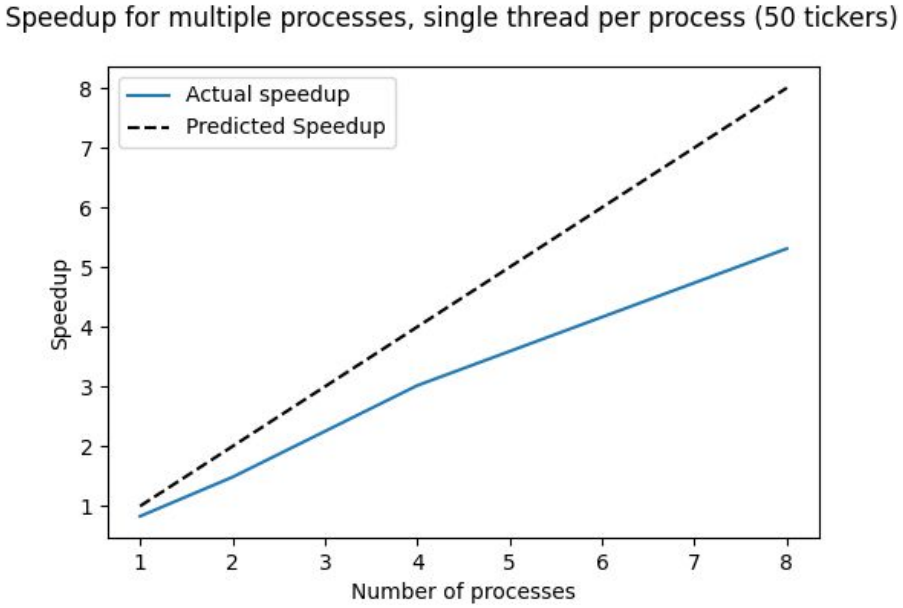
Phase I: Data Processing

```
for t in tickers:
        for d in days:
                for n in sequences_per_day:
                        seq = generate_sequence(t, d, n)
```

Phase I: A first attempt

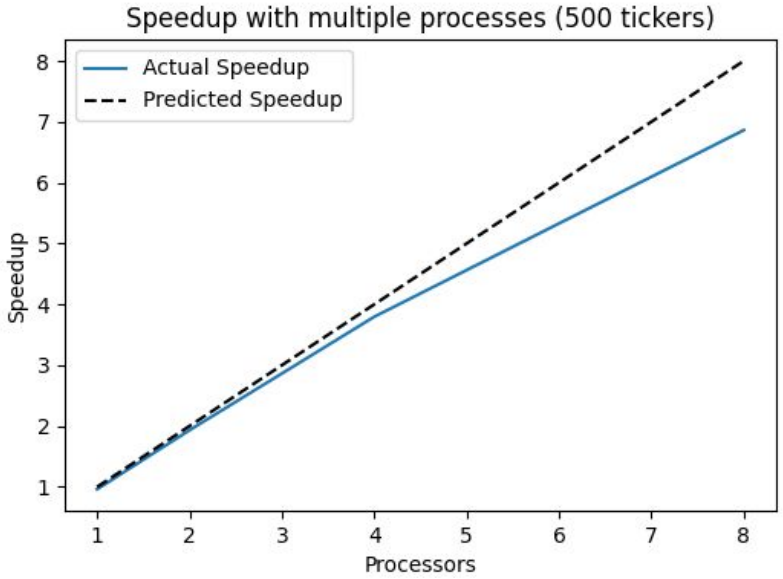
Idea:

- Break down problem into entirely separate domains for each stock and day
- Python *multiprocessing* module
- Challenge: API ultimately accesses the data separately
    - Large I/O Overhead

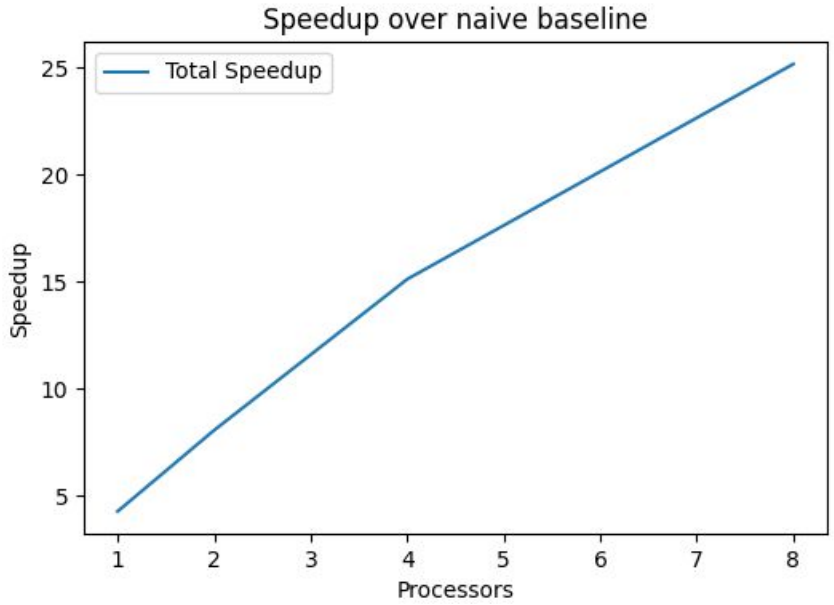


Phase I: Achieving strong scaling

- Solution: *Multithreading*
  - Fine-grained parallelism
  - Optimizes I/O of files
- Pre-implemented in yfinance
- Combine with multiple processes to parallelize both I/O and computation



Phase I: Putting it together



Wall Times:

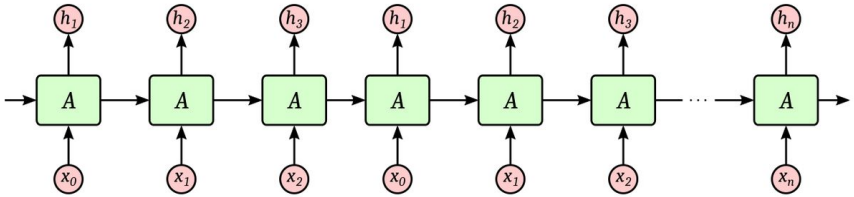
- Naive Baseline:
  - ~2.5 hours
- Multithreading Baseline
  - 36 minutes
- Fully parallelized:
  - 5 minutes

Phase I: Parting thoughts

- Lessons learned:
  - Be wary of I/O, and interacting levels of parallelism
  - Performance gains can enable more experimentation with processing & possibly scientific progress
- Future considerations:
  - *yfinance* implements multithreading on the basis of 2 * # of CPUs available - potentially open to further tuning.
  - Limits to the gains from a practical perspective

Model Training - Overview

- LSTMs are inherently sequential and difficult to parallelize



- Focus on Data Parallelism:
  - Data is distributed across the nodes or GPUs
  - Gradient updates are calculated separately on each node and averaged across all the nodes

Model Training - Programming Model

- Accelerated Computing
  - Parallelization within batches
  - CuDNN, NVIDIA M60 GPU

- Distributed Memory Parallel Processing:
  - Parallelize training between batches
    - 1 - 4 Node
    - 1 - 2 GPUs
  - Horovod
    - Open MPI
    - Ring-AllReduce

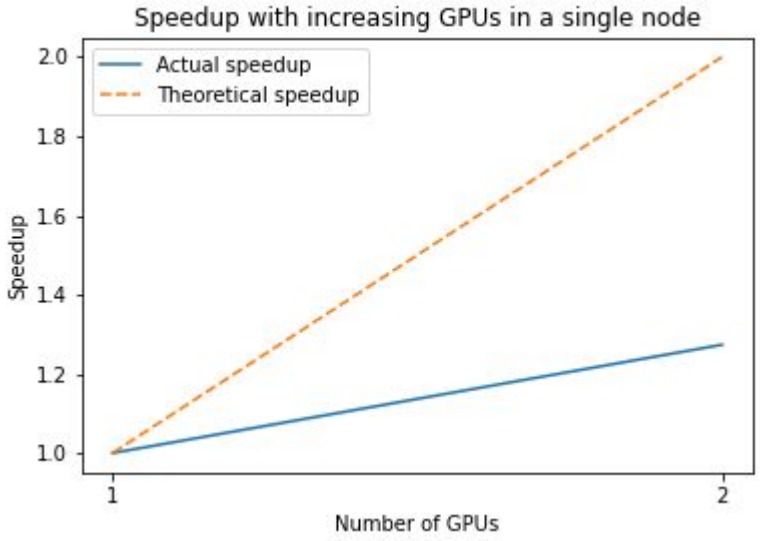
Model Training - Performance

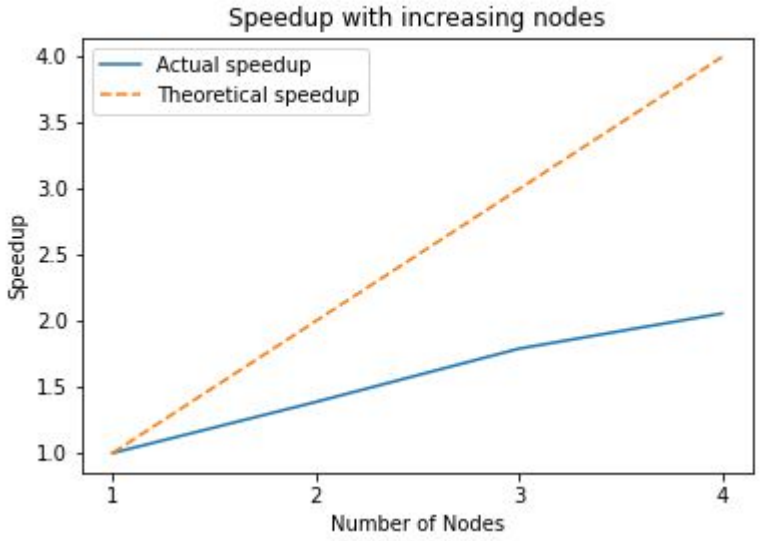
Performance metrics:

- Speedup
  - Multi-Node vs Multi-GPU
  - Different effective batch sizes

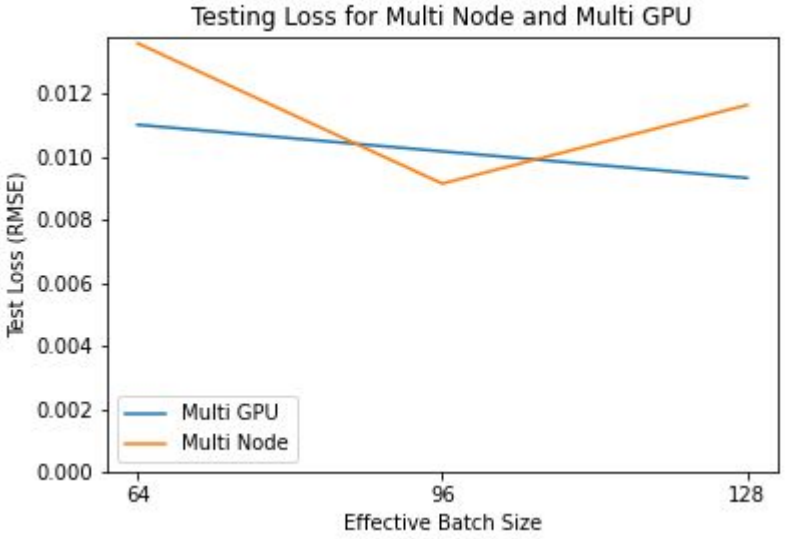
- Time/step & time/epoch
  - Multi-Node vs Multi-GPU
  - Different effective batch sizes

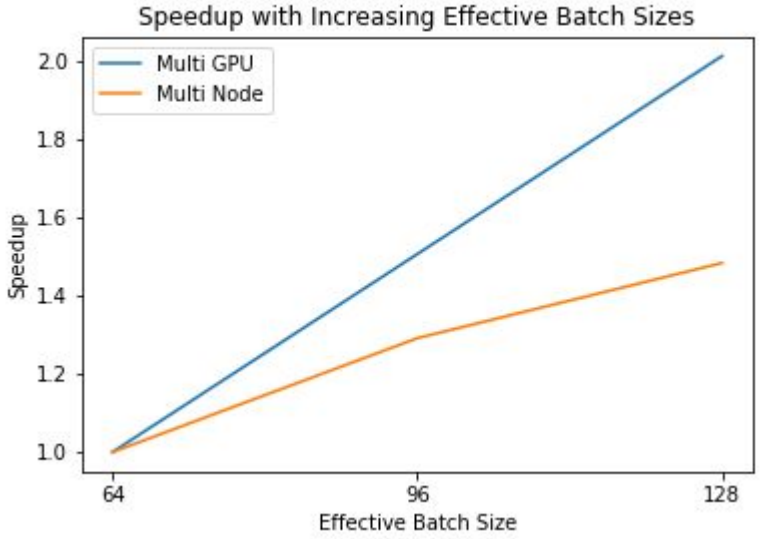
Model Training - Performance



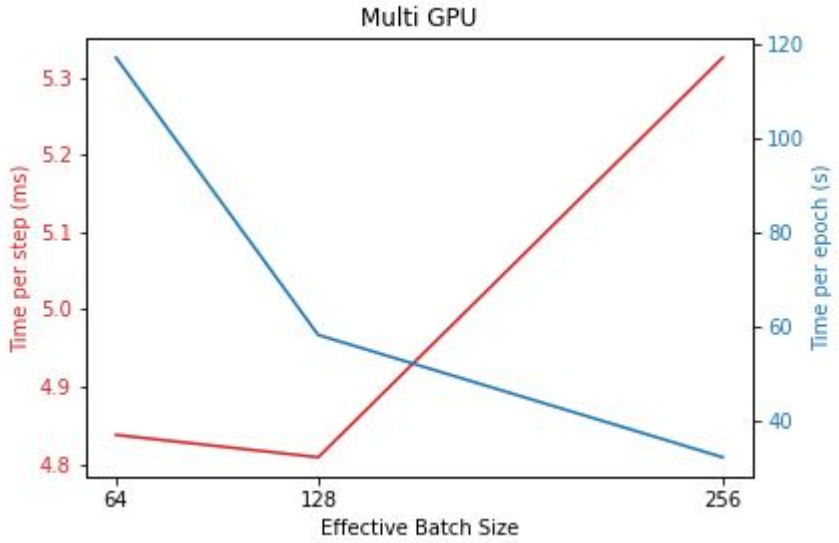


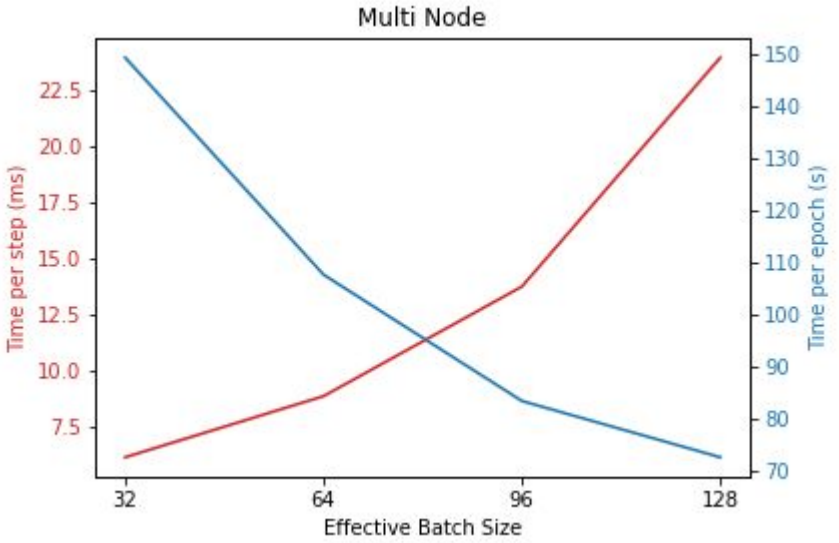
Model Training - Performance





Model Training - Performance





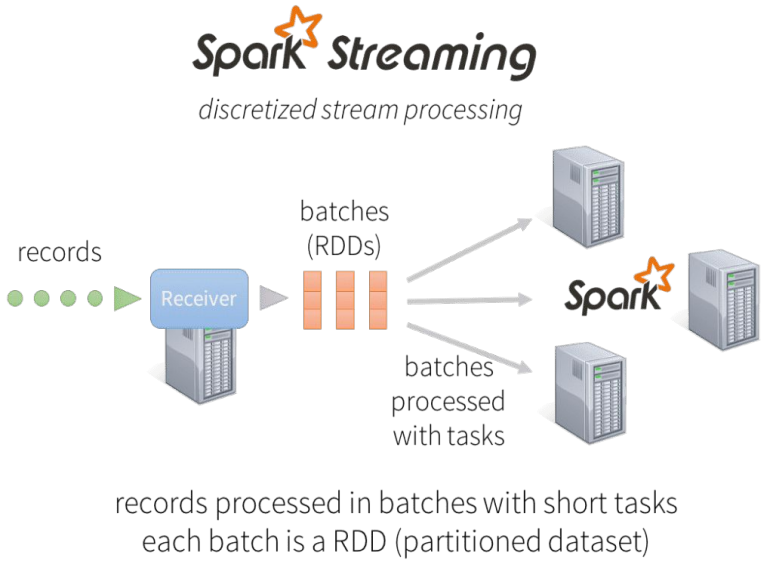
Model Training - Key Findings

- Scaling efficiency diminishes fairly quickly
- Using larger batch sizes increases speedup (up to a certain point)

Future steps

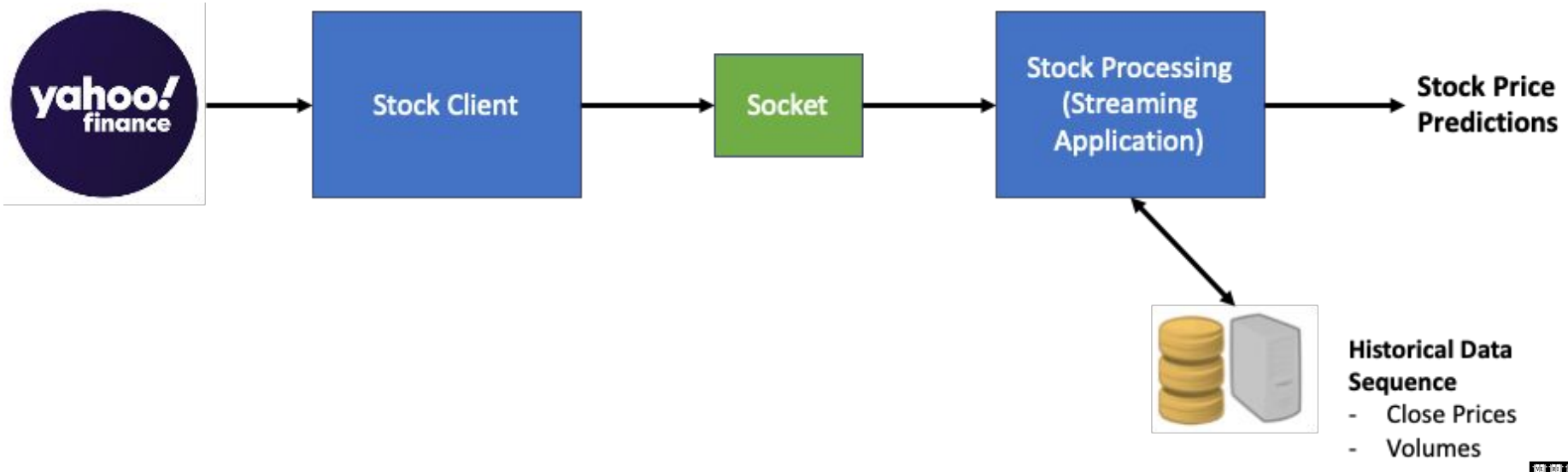
- Increase accelerated computing:
  - Increasing feature vectors
  - Larger hidden units in the LSTM
- Decrease sequence length

Real-Time Prediction - Spark Programming Model

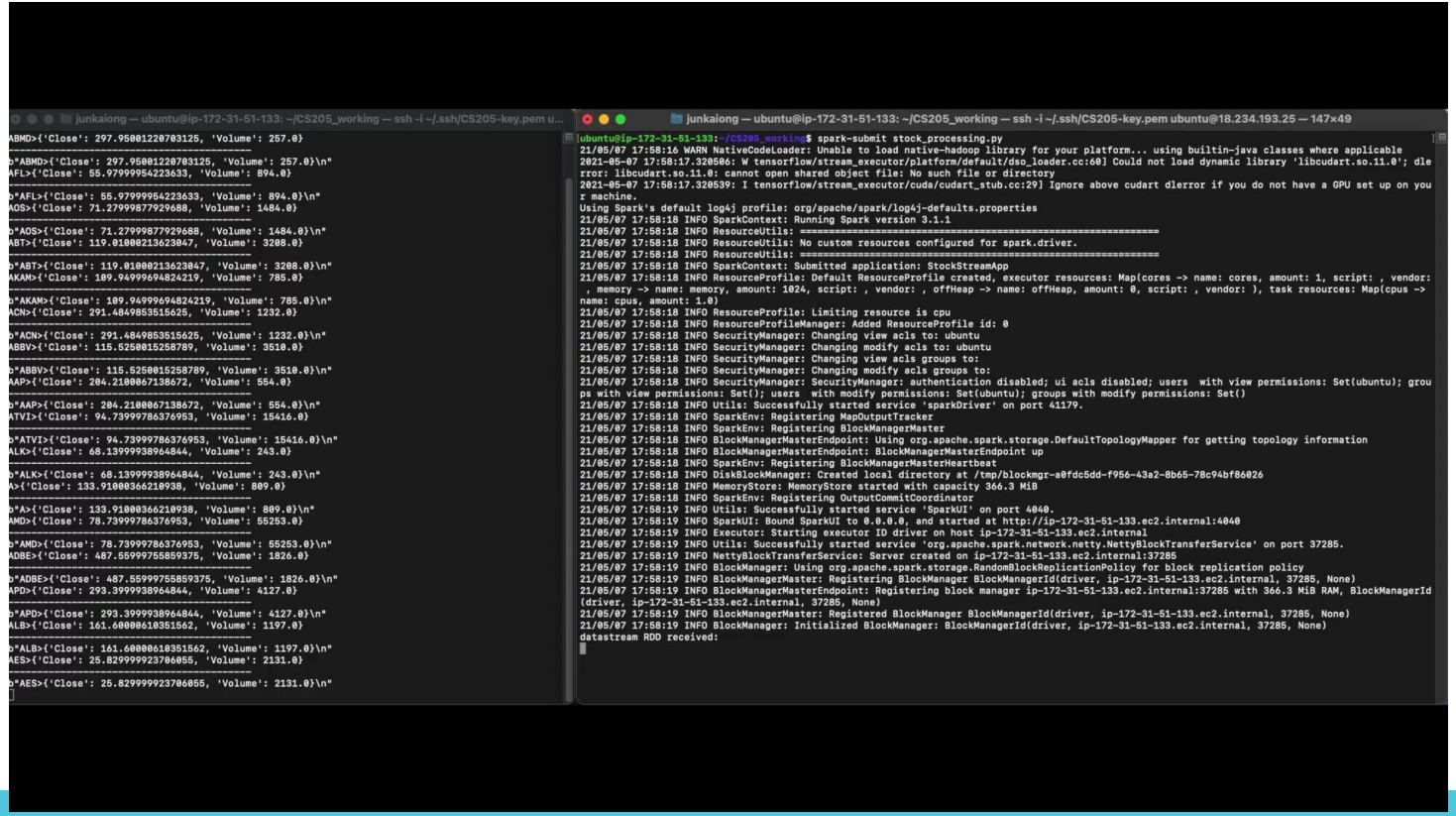


- Streamed Dataset is constructed using data from yfinance API
  - Unbounded dataset
  - Defined as amount of data that has entered system so far
  - Processing is event-based, continues until explicitly stopped
- Stream Processing
  - Predictions are computed based on current stock data
  - Current stock data incorporated into data sequence (60 min)
  - Trained LSTM model predicts stock price using the data sequence

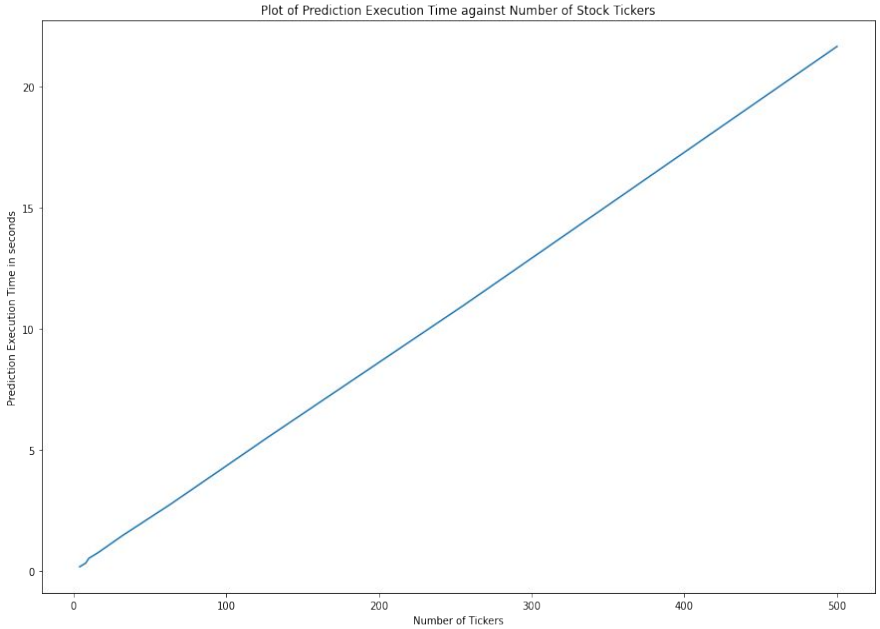
Real-Time Prediction Application Architecture

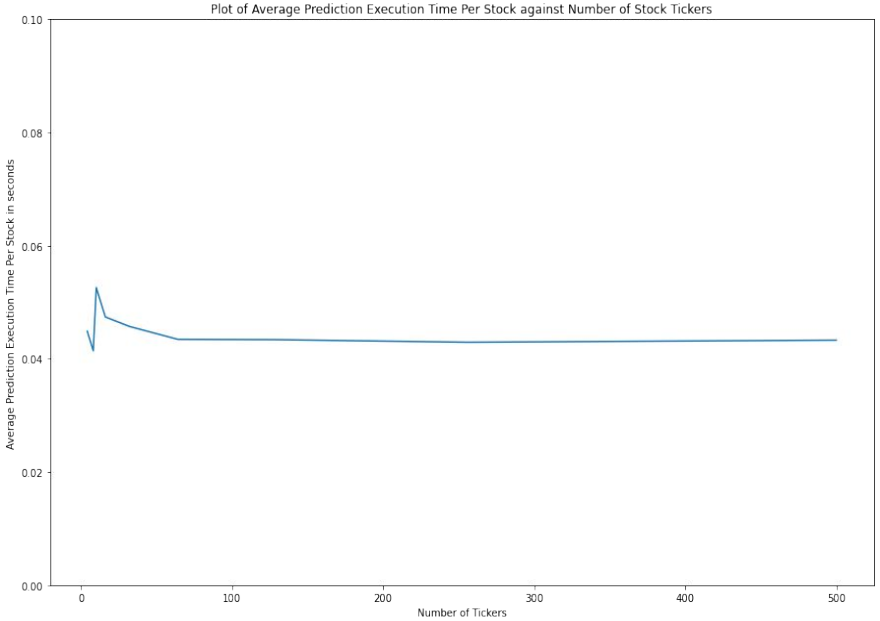


Real-Time Prediction Demonstration



Real-Time Prediction Execution Time





Infrastructure (AWS)

Data Processing:

- t2.2xlarge instances

Model Training

- g3s.xlarge & g3.8xlarge instances
  - NVIDIA Tesla M60 GPU
  - Horovod
  - OpenMPI

Prediction:

- t2.2xlarge instances

Storage:

- S3 Standard

Lessons and Future Directions

- Interaction with external APIs or data sources may set parameters for parallelism outside of the theoretical domain.
- The garden of forking paths is strong:
  - Many ways to develop parallel programs within these constraints.
  - Important to evaluate resources in conjunction with program design.
- Increase the resolution of prediction from 5 minutes ahead to 2 minutes ahead
- A potential future direction is to experiment with the use of Elephas for the prediction phase
  - Potentially allow us to perform model prediction directly with the RDDs received
- Future exploration of using an even smaller machine for prediction
  - Helps to increase access to larger group of end-users and reduce cost

Thank You!

Links to Project Resources

Project Website and GitHub Repository:
https://github.com/vrsivananda/CS205_FinalProject

Project Presentation Video Download:
https://github.com/vrsivananda/CS205_FinalProject/blob/master/presentations/Group11_CS205_FinalProjectPresentation.mp4

Project Presentation Slides Download:
https://github.com/vrsivananda/CS205_FinalProject/blob/master/presentations/Final%20Project%20Presentation%20Submission.pdf