Serving AI using a Distributed Architecture

Final Year Project 2019/20 - Interim Report

Waqas Ali (3035396771)

Supervisor: Dr. Heming Cui
Mentor: Shixiong Zhao

University of Hong Kong

February 2, 2020

Abstract

In recent years, artificial intelligence (AI) has penetrated multiple dimensions of people's daily lives by making the devices they use smarter. Fueled by data, AI programs imitate human intelligence in terms of their learning and behavioral capabilities. With such widespread usage, however, users demand improved functionalities and speed, pushing developers and data scientists to make their programs smarter in the midst of industry competition. These smarter programs have to deal with more complexities, and developers consequently have to choose whether to prioritize the program's features or performance, posing a dilemma for them. This project proposes to design an AI application with a distributed architecture instead of a centralized architecture (the more common structure in the status quo) in order to improve its latency and throughput. As proof of concept, the project will specifically examine a complex stock price prediction application and the goal is to transfer it onto a distributed architecture. The project's objective is to develop tooling and foundation to automatically instantiate and compare distributed systems of a variety of specifications and scheduling algorithms. There are three milestones in this project. Firstly, the machine learning stage where test models have to be developed and trained. Second, modifying the model serving to work in a distributed manner. Lastly, comparing distributed implementations which is the most crucial aspect of this project. As of now, the first and second milestones have been reached in which a stock price prediction service has been successfully developed, trained and deployed for a single machine and a cluster. As a result, there is a functioning baseline to which further iterations to the service can be made to better evaluate the viability of a distributed architecture. Moreover, a metric tool i.e. web app has been developed which measures latency of the service in an architecture-agnostic way. The project is on track for an enhanced distributed implementation of the stock price prediction service to be developed and compared with non-distributed implementation to assess if latency and throughput could be improved.

Acknowledgements

In addition to my supervisor and mentor who helped me with the technical aspects of my project, I would like to thank Ms. Mable Choi (HKU CAES) for her guidance in distilling and consolidating my work in a better way.

Contents

Abstract **i**

Acknowledgements **ii**

List of Figures **v**

List of Tables **vi**

1 Introduction **1**
1.1 Background 1
1.2 Motivation 2
1.3 Contribution 3
1.4 Objectives 3
1.5 Report Organization 4

2 Methodology **5**
2.1 Choose AI Application for testing 5
2.2 Develop basic application 5
2.3 Run on a centralized system 6
2.4 Convert to a distributed system 6
2.5 Programmatic Deployment 6
2.6 Compare 6
2.6.1 Test Cases 7
2.7 Summary 7

3 Schedule **8**

4 Progress **9**
4.1 Summary 9
4.2 Stock Price Prediction 9
4.2.1 Why? 9
4.2.2 Challenges 9
4.3 Deployment on a single machine 12
4.4 Tooling for measuring metrics 12
4.5 Deployment on a cluster 12

5 Limitations and Difficulties **13**
5.1 Personal Limitations/Difficulties 13
5.2 Potential Difficulties 13

6 Conclusion **14**

Bibliography **14**

List of Figures

1.1 Inference pipeline of a basic stock price prediction service. C2 and C5 are data retrievers which fetch past stock data and twitter mentions of a specified stock symbol. C6 is a caching layer which allows skipping C7-C11 steps if a prediction has been recently made for a specific stock symbol. C7, C8, C9, C10 & C11 are ML models that compete against each other to predict a stock symbol's price. C4 is a sentiment analyzer whose results are taken into account for price prediction. As can be seen, it is a fairly complex pipeline composed of several different steps. 2

4.1 This graph shows how our model learned from GE's historical stock price data. An Epoch means a round of training and Loss means how far off we are from actual values. A lower loss means a more accurate data. Train means the performance of our model on training data (data it uses to train itself). Test means the performance of our model on testing data (data it only uses for measuring performance and does not use to train itself at all). As can be seen, our model trains well quickly as the loss drops rapidly after 5 rounds. 10

4.2 This graph shows our model's predicted price for GE's stock for 120 days and the actual price for those 120 days. As can be seen, our model can do better but it captures the direction of the trend well. Predicted price and real price both show increase around the 20th day and both show a drop around the 90th day. 11

List of Tables

3.1 Project Schedule 8

Chapter 1

Introduction

1.1 Background

Artificial intelligence is an area of computer science that focuses on granting machines the ability to act intelligently [3]. It is a vast field with limitless applications and each application has its own unique solution. Machine learning, specifically, is a subset of artificial intelligence that learns from data. [4] Today we see ubiquitous applications of artificial intelligence such as spam filters [1], recommendations [2], virtual assistants and self-driving.

Customers are demanding smarter and smarter capabilities in their machines, and this trend leads to a new set of software development challenges for AI developers. *Nvidia* summarises them with the PLASTER [5] framework:

- Programmability
- Latency
- Accuracy
- Size of Model
- Throughput
- Energy Efficiency
- Rate of Learning

These challenges carry over to the realm of machine learning, since it is a subset of artificial intelligence. A machine learning application has two main stages:

1. Training (Learning from data)
2. Inference (Given an input, predicting an output)

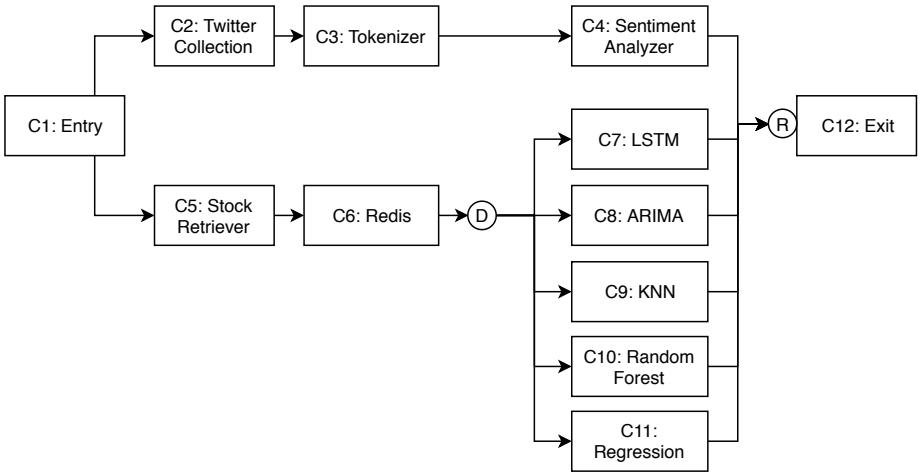

Figure 1.1: Inference pipeline of a basic stock price prediction service. C2 and C5 are data retrievers which fetch past stock data and twitter mentions of a specified stock symbol. C6 is a caching layer which allows skipping C7-C11 steps if a prediction has been recently made for a specific stock symbol. C7, C8, C9, C10 & C11 are ML models that compete against each other to predict a stock symbol's price. C4 is a sentiment analyzer whose results are taken into account for price prediction. As can be seen, it is a fairly complex pipeline composed of several different steps.

Take the example of an application that relies on a machine learning model to transcribe voice. Before the model can be used by the application, it needs to be trained. To do this, developers expose the model to hundreds of voice recordings to allow it to learn which sounds match to which words. Now, the application can use the model by sending it voice recordings and receiving transcribed text in return. In short, this process of predicting an output in response to an unseen input is inference, the second stage of machine learning as mentioned previously.

1.2 Motivation

Since end-users of machine learning applications are only concerned with inference, not training, inference must be quick.

For inference, an input goes through multiple steps, known as a pipeline. Figure 1.1 is an example of a stock price prediction service's pipeline. As tasks in a pipeline increase in quantity and complexity, it can increase the *latency* (time taken) to execute all steps of the pipeline. Moreover, if a *centralized architecture* (single machine) executes the complete pipeline, it can create bottlenecks. For example, if a pipeline for input A is in progress, the pipeline for input B cannot start.

On a centralized architecture, all tasks have to be done sequentially (even

if they are independent of each other). This could take a long time and hence increase the latency. Moreover until all tasks for a specific request have finished, processing for a new request cannot start. Thus, the service cannot handle a high number of requests in a given period i.e. *throughput*.

1.3 Contribution

Several methods could be considered to optimize the latency and throughput of an artificial intelligence application.

This project attempts to tackle this optimization problem by efficiently distributing the pipeline tasks over several machines. The claim is that, theoretically, we can improve the latency and throughput of an AI application (concerns which were highlighted in the PLASTER framework [5]) if decentralized architecture is employed instead of centralized architecture.

1.4 Objectives

There are two steps to developing any distributed application:

1. Program the application in a way as to take advantage of multiple machines.

2. Deploy the application on a network of machines.

Therefore, moving an artificial intelligence application from a centralized architecture to a distributed network not only requires deploying it on a network of multiple machines but modifying it to utilize the newly available resources.

Since every application is unique, devising a general technical solution for accomplishing the above two steps will be infeasible. Therefore, the project will choose one AI application as a proof of concept and use that as a testing ground of the solution proposed in previous section.

A distributed application's success depends on how the application divides its tasks (job scheduling) and quality/quantity of resources available for use. To figure out what works best we need a quick and reliable way of testing different job scheduling algorithms on networks of different sizes composed of machines of different specifications.

Consequently, the project's objectives are to develop the following programs:

1. An AI application with a complex inference pipeline which can accept different job scheduling algorithms.

2. A deployment script that can programmatically buy cloud resources and deploy our AI application according to provided specifications.

3. A web app to measure latency and throughput.

Given the above objectives are fulfilled, we can confidently argue for or against using distributed systems for AI applications.

1.5 Report Organization

Composed of artificial intelligence, distributed systems, cloud services and web development, the project's methodology involves several fields of computer science and software engineering to achieve the objectives mentioned in section 1.4. Naturally, that invites its own set of complexities and uncertainties. Therefore, chapter 2 accomplishes the important task of narrowing down and justifying the methodology. Moreover, to keep a project as large as this on track, chapter 3 breaks the project into milestones and plans for both the short-term and long-term. Following this, chapter 4 reports in detail the current progress of the project which, fortunately, is on track. Adding to that, chapter 5 discusses limitations and difficulties the project has encountered so far which may be of interest to those who want to reproduce or extend the project's work. Lastly, chapter 6 summarises this report and the project so far.

Chapter 2

Methodology

From developing an artificial intelligence application to running it on a distributed architecture to comparing it to traditional implementations, there are many steps to this project and without proper breakdown it could get overwhelming quickly. The following sections describe and justify the steps through which the project aims to accomplish its objectives.

2.1 Choose AI Application for testing

As the project proposes a distributed architecture for artificial intelligence applications in general, our test AI application must be a sufficient representative of most if not all AI applications for a fair investigation. Naturally, a fairly representative application is one with a pipeline composed of different kinds of tasks with a mix of mutually dependent and independent ones. Ergo, choosing a single AI application as a testing ground for our solution is an important task that requires studying popular AI techniques and implementations in the community.

2.2 Develop basic application

Every AI or ML application starts with data science. First of all, machine learning models need to be trained and an inference pipeline needs to be developed. This requires studying current techniques for the application of our choice and using that knowledge to build and train good enough models. At this stage, accuracy is not important so we do not need to fine-tune the models. Once the model training is done, it needs to be ready for inference. Therefore, we need to ensure that all the steps required to accomplish inference on a new unseen input have been implemented at a satisfactory level.

2.3 Run on a centralized system

By now we have chosen a test AI application and trained basic models for it. Moving on, we need to ensure we can successfully run inference on a single machine. This step is important for two reasons. Firstly, this gives us a baseline performance we can compare our distributed implementations with. Second, we will have a working implementation of our application and we can refer to it while converting our application to a distributed implementation.

2.4 Convert to a distributed system

Consequently, the next step is to convert the application from a centralized implementation to a distributed implementation. This will require modifying the source code to use distributed system techniques such as RPC (remote procedure call). To ensure consistency, we should ensure our distributed implementation running on one machine has the same performance as the centralized implementation from earlier.

2.5 Programmatic Deployment

The only way to test a distributed implementation is to deploy it on a network of computers and measure performance. Cloud services make it considerably easy to do so without having to deal with actual hardware. After this stage, we should be able to automatically instantiate cloud resources according to provided specifications and deploy the distributed implementation of our application on them. We can also deploy our implementation on the cloud manually but that will take a lot of time and considering the number of times we would have to do it, it is not feasible. Besides, we need to ensure all implementations are reproducible and consistent.

2.6 Compare

With all these different implementations, we need a reliable way of comparing each implementation's performance that is completely decoupled from its intrinsic qualities. A fair and reliable way to compare is to create a web app that accepts a server URL and sends numerous requests to it. Consequently, it measures latency for each request and throughput in general. As the web app is run in the browser, it measures these metrics from the client-side and all it cares about is input and output. Thus, it does not matter for the web app if the server implementation is on a centralized or distributed architecture as long it receives an output.

2.6.1 Test Cases

To study whether a distributed architecture can indeed improve latency and throughput, performance will be compared across systems of various specifications:

1. Centralized implementation (baseline)
2. 1 machine for n tasks (should be same as above)
3. Less than n machines for n tasks
4. n machines for n tasks (optimum)

2.7 Summary

To summarise, the project picks an AI application, builds it to run on a centralized architecture, converts the application to run on a distributed architecture, runs the application on distributed systems of different specifications and, lastly, compares the application across all these implementations to assess whether latency and throughput can be improved using a distributed architecture.

Chapter 3

Schedule

To accomplish the objectives in section 1.4, the project follows the following schedule. It is composed of milestones with a target completion month for each.

Table 3.1: Project Schedule

2019	
October	Choose & Design AI application to work on Develop a basic inference pipeline on a centralized architecture Progress Report 1
November	Convert pipeline to work on a distributed system using RPC Web app to measure latency and throughput Progress Report 2
December	Manually deploy pipeline to distributed architecture on cloud
2020	
January	Programmatically instantiate cloud resources and deploy model First Presentation Detailed interim report
February	Vary deployments by cloud resources and measure latency/throughput on each
March	Enhance AI inference pipeline by adding more steps
April	Finalize implementation Final Presentation Final Report

Chapter 4

Progress

4.1 Summary

As of now, the first four milestones mentioned in Chapter 2 have been reached. I have chosen an AI model, developed a basic model, ran inference successfully on a single machine, built a latency measuring web app and converted the service to run on a distributed architecture using RPC (Remote Procedure Call).

4.2 Stock Price Prediction

4.2.1 Why?

After consultation with my mentor and careful research, I chose stock price prediction as my AI application. Input any stock symbol and it will predict its price on the next day. There are many ways to make such a prediction. One approach is to analyze past prices and predict the stock price behavior based on that. However, this approach completely ignores the market and current affairs. An interesting approach would be to use cutting edge time series forecasting models combined with sentiment analyzers to combine the best of both worlds.

4.2.2 Challenges

As the stock price prediction service should be able to predict prices for any number of the thousands of stock symbols out there and there is always something new happening in the context of current affairs, it is infeasible to train the model in advance for all stock symbols. Therefore, whenever the service is tasked with predicting price for a stock symbol it has to retrain itself and then carry out inference. Consequently, this process of training and then inference creates huge latency. As can be seen in Figure 1.1, there are many tasks that can be done independently. Hence, stock price prediction service is a prime testing ground for application of distributed computing in artificial intelligence.

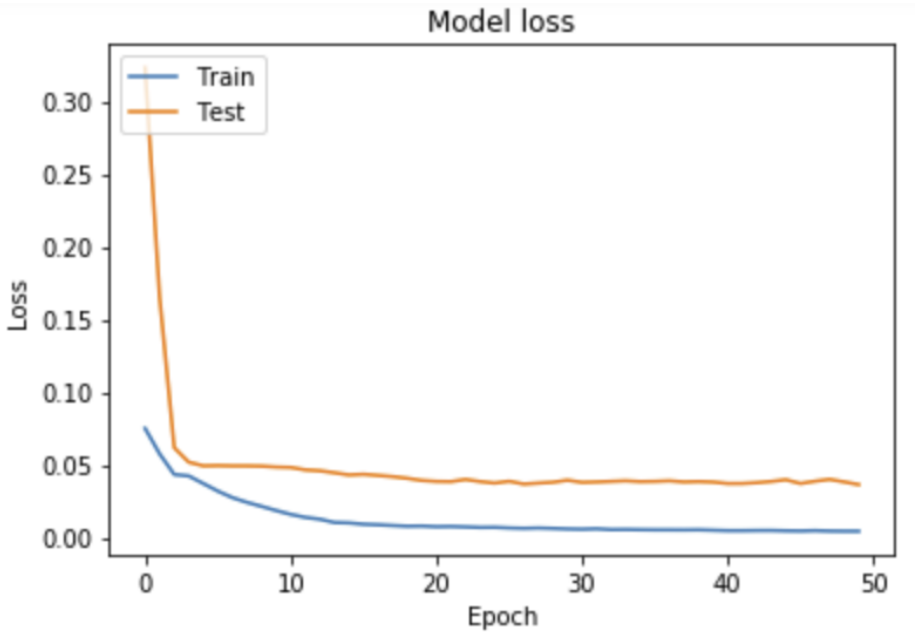

Figure 4.1: This graph shows how our model learned from GE's historical stock price data. An Epoch means a round of training and Loss means how far off we are from actual values. A lower loss means a more accurate data. Train means the performance of our model on training data (data it uses to train itself). Test means the performance of our model on testing data (data it only uses for measuring performance and does not use to train itself at all). As can be seen, our model trains well quickly as the loss drops rapidly after 5 rounds.

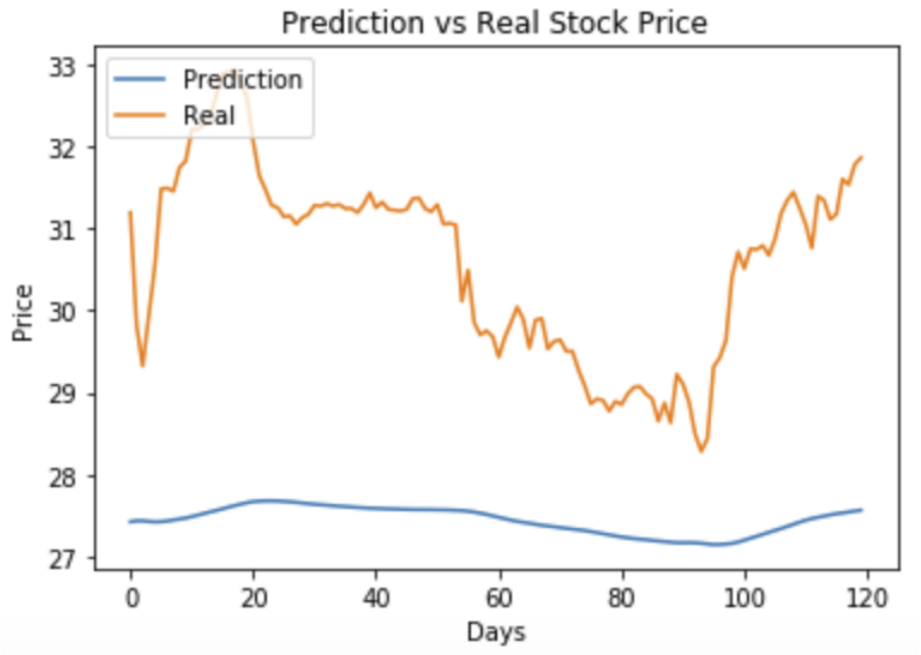

Figure 4.2: This graph shows our model's predicted price for GE's stock for 120 days and the actual price for those 120 days. As can be seen, our model can do better but it captures the direction of the trend well. Predicted price and real price both show increase around the 20th day and both show a drop around the 90th day.

4.3 Deployment on a single machine

As a baseline, the developed machine learning model was deployed onto a single machine ready for inference.

4.4 Tooling for measuring metrics

Moreover, a basic web app was developed which accepts a URL for the service and measures the latency for stock price prediction. It must be noted that, in addition to latency, throughput is an equally important metric for the project but it has yet to be added to the web app.

4.5 Deployment on a cluster

Building upon the previous work, a gRPC (Open Source Remote Procedure Calls Implemenation) server was developed which served the machine learning model. After that, a Docker Image was made so the service can be easily run as a microservice on any machine. Then, Terraform (Infrastructure as Code) was used to deploy the docker image onto a HKU department server (with GPU). Ultimately, a gRPC client on a separate machine was used to call the microservice. The client was also a Flask server which responded to requests of the above-mentioned metric-measuring web app (built in React). All in all, a rudimentary distributed implementation was successfuly made ready to be enhanced further.

Chapter 5

Limitations and Difficulties

5.1 Personal Limitations/Difficulties

As I do not have enough experience working with time series data, much of my time has been spent on learning how to properly develop a machine learning model for it. Moreover, I have had to learn several time series forecasting techniques mentioned in Figure 1.1 which has taken considerable time. In addition, learning gRPC and using it in conjunction with Docker and SSH tunneling was challenging.

5.2 Potential Difficulties

The current baseline model takes a long time to train and I am not sure whether I can bring down the time enough even if I use a distributed architecture.

Chapter 6

Conclusion

The project has three crucial parts to it: machine learning, metric tooling and distributed architecture. As shown in chapter 4, a functional stock price prediction service has been developed and deployed which closely resembles most machine learning applications in the real world. Moreover, reliable metric tooling has been successfully setup and is ready to measure further iterations of the stock price prediction service. Adding to that, a basic distributed implementation has been developed and further work is being currently done to improve it. With more than two of three parts covered, the project is on track for a distributed architecture to be further developed for the stock price prediction service. As mentioned in chapter 2, the hypothesis mentioned in section 1.3 (distributed architecture is better than traditional architecture) will be tested using production-representative machine learning application and reliable metric tooling once the distributed architecture is fully implemented. With the data obtained from these tests, the project will assess whether latency and throughput of AI applications can be improved using a distributed architecture.

Bibliography

[1] Ion Androutsopoulos et al. "Learning to Filter Spam E-Mail: A Comparison of a Naive Bayesian and a Memory-Based Approach". In: *CoRR* cs.CL/0009009 (2000). URL: http://arxiv.org/abs/cs.CL/0009009.

[2] George Lekakos and Petros Caravelas. "A hybrid approach for movie recommendation". In: *Multimedia tools and applications* 36.1-2 (2008), pp. 55–70.

[3] John McCarthy. *What Is Artificial Intelligence?* Tech. rep. Stanford University, 2007.

[4] Thomas Mitchell. *Machine Learning (McGraw-Hill Series in Computer Science)*. McGraw-Hill Education, 1997.

[5] David A. Teich and Paul R. Teich. *PLASTER: A Framework for Deep Learning Performance*. Tech. rep. TIRIAS Research, 2018.