Progress Report 1
Serving AI applications using a distributed architecture

Waqas Ali
Supervisor: Dr. Heming Cui
Mentor: Shixiong Zhao

October 23, 2019

Abstract

Nowadays, artificial intelligence is everywhere from our homes to our hands making everything smarter. Learning from feedback, artificial intelligence programs imitate human intelligence. However, with their increasing usage comes ever-increasing user expectations and industry competition. As developers and data scientists make their programs smarter, they also make them complex. Consequently, it becomes a choice between features and performance, a highly undesirable position. This project explores moving artificial intelligence applications over to a distributed architecture instead of a centralized architecture as is the status quo. As proof of concept, a complex stock price prediction application is considered 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. It's divided into three big milestones. First of all, the machine learning stage where a test model has to be developed ready to be improved upon. As of now, a stock price prediction service has been successfully developed. Second, modifying the model to work in a distributed manner which is next. Lastly, comparing distributed implementations of different kinds which is the most crucial aspect of this project.

Contents

List of Figures

List of Tables

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's 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] Nowadays 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 aforementioned 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)

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. As a pipeline increases in quantity and complexity, it can increase the *latency* (time taken) to execute all these steps. 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 proposed solution.

A distributed application's success depends on how the application divides its tasks (job scheduling) and what kind of resources are available for use. To figure out what works best we need a quick and reliable way of testing different job scheduling algorithms on 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 supplied 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

The remainder of this report is organized as follows. Chapter 2 describes and justifies the methodology. Chapter 3 breaks the project into milestones and gives a schedule with estimated completion time for each. Following this, chapter 4 reports current progress and chapter 5 discusses limitations and difficulties faced so far. Lastly, chapter 6 summarises lessons learned and future steps.

Chapter 2

Methodology

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 model

Every AI or ML application starts with data science. First of all, a machine learning model needs 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 a good enough model. At this stage, accuracy isn't important so we don't need to fine-tune the model. 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 a basic model for it. Moving on, we need to ensure we can successfully run inference on a single machine on the cloud. 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 of 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 isn't 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)

Chapter 3

Schedule

To accomplish the objectives in section 1.4, the project follows the following schedule. It's 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 three milestones mentioned in Chapter 2 are finished. I have chosen an AI model, developed a basic model, ran inference successfully on a single machine and built a latency measuring web app.

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. A rudimentary pipeline is visualized in Figure 4.1.

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 on the news landscape, 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. All in all, this creates huge latency. As can be seen in Figure 4.1, there are a lot of tasks that can be done independently. Hence, stock price prediction service is a prime testing ground for application of distributed computing in artificial intelligence.

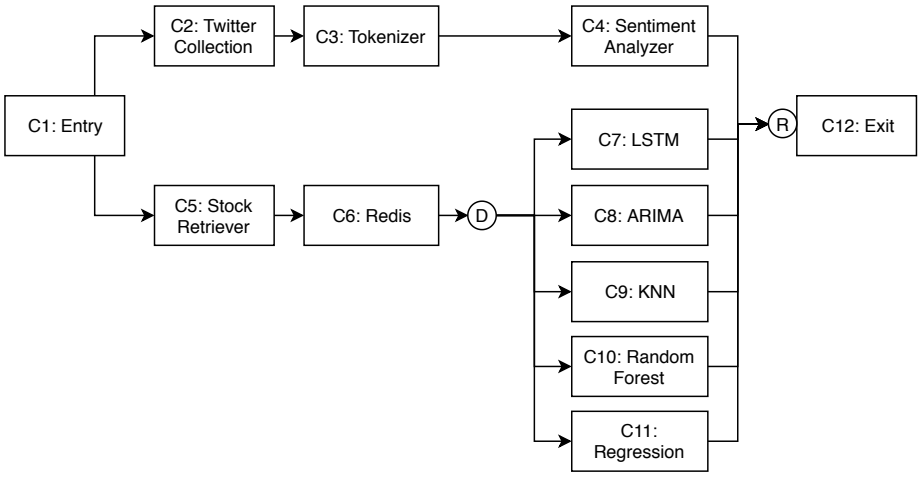

Figure 4.1: Inference pipeline of a basic stock price prediction service. C2 and C5 are data retrievers. C6 is a caching layer. C7, C8, C9, C10 & C11 are ML models that compete against each other. C4 is a sentiment analyzer whose results are taken into account for price prediction.

4.3 Deployment on a single machine

As a baseline, I took the machine learning model I just developed and deployed it onto a single machine ready for inference.

4.4 Web app for measuring metrics

In addition I have setup a basic web app that accepts the server URL above and measures the latency for stock price prediction. I haven't added throughput measurement yet.

Chapter 5

Limitations and Difficulties

5.1 Personal Limitations

As I don't have enough experience working with time series data, a lot of my time has been spent on learning how to properly develop a machine learning model for it. Moreover, I have had to learn a lot of time series forecasting techniques mentioned in Figure 4.1 which has taken considerable time.

5.2 Potential Difficulties

I have never implemented a distributed architecture using python and I can already foresee me having to learn how to do that for my next step. Moreover, 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. Lastly, I will need a cloud service account where I can instantiate required resources. Based on previous experience, it is not the most straightforward process. So I will have to figure out how I can do that.

Chapter 6

Conclusion

As of now, I am on track as far as the schedule is concerned. I have a foundation to build upon and next steps are to convert the stock price prediction service into a distributed architecture. However, as I have only tackled the machine learning aspect of my project until now and I have yet to dive into the distributed system aspect, I am uncertain of how difficult it will be.

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.