Report for CS131 Project

Zixuan Lu
304990072

2019-06-03

Abstract

This report for Homework 6 looks into the architecture of Tensorflow, and Tensorflow in other languages (Java, OCaml). I will compare the pros and cons of each language and ultimately give a conclusion about choosing other languages as a replacement for Python as the platform of Tensorflow.

1 Introduction

Tensorflow is a widely-used, open source platform that supports machine learning. Typically, larger Tensorlow applications are bottlenecked inside C++ or CUDA code (e.g., in the Eigen or cuDNN libraries). However, in the context of the homework which we build an application with many small queries, the bottleneck is in the Python code which sets up the models. This is the motivation of looking into other programming languages and see if they are good replacements for Python to build Tensorflow framework. I will look into Java, OCaml, and Kotlin on their pros and cons, as well as their ability to support an event-driven server just like that implemented in the project (in Python).

2 Python

Python is an interpreted, high-level, general-purpose programming language[1]. It is dynamically typed, with comprehensive documentation and supports multiple programming paradigms, including procedural, object-oriented, and functional programming[1]. Also, Python has a asyncio module (used for the project) which supports event-driven, asynchronous programming, which is required as stated in the homework spec (to support the event-driven server). Thus, without considering Performance, Python is definitely capable of implementing this server.

2.1 Pros

With its dynamic typing-checking and easy syntax, Python is very easy to use, which enhances prototyping speed (this is the reason why we choose Python to implement the project prototype). Also, with the built-in Tensorflow library, we do not have to worry about language bindings which further increase development efficiency. When it comes to memory management, Python users do not have to worry about memory allocation/deallocation, which is more efficient and generally less error-prone(will be discussed in the cons subsection).

2.2 Cons

Python uses a garbage collector that relies on reference count as stated in the project report. However, objects will not be deallocated with cyclic references. Also, though dynamic-typing makes Python's syntax easy to understand, it means type checking occurs during runtime which hurts performance. This compromise in performance is not obvious when handling large queries (bottlenecks in C++ or CUDA code). However, the proportion of time to set up models in Python code when handling small queries increases significantly, as stated in the homework spec. Lastly, Python does not support "true" parallelism: Python's inefficiency in multi-threading comes from its Global Interpreter Lock (GIL), which "is a mutex that protects access to Python objects, preventing multiple threads from executing Python bytecodes at once"[2]. For example, it is never possible for an individual server to update its client database simultaneously by multiple threads.

3 Java

Just like Python, Java has a built-in garbage collector which implements a mark and sweep algorithm. On the other side, Java uses static type-checking. Also, Java Future, which is introduced around Java 1.5, can be useful when working with asynchronous calls and concurrent processing[3].

3.1 Pros

Since Java is statically typed, this means type checking will be done during compile time, which improves efficiency as compared to Python. Also, static type checking forces users to define variables before using, and the declarations serve as an implicit documentation of the code. This also prevents type error to take place during runtime, making the program more reliable.

Also, Java has very good support for parallelism, which can increase throughput of each individual server by a significant amount, considering most server-side CPU's are multi-cored, multi-threaded. However, since setting up models are not computational heavy, this advantage of Java may not be fully exploited.

3.2 Cons

Although Java's garbage collector is free from side effects of cyclic reference, its mark and sweep algorithm compromises performance: when it runs, normal program execution is suspended.[4] Also, although Java's static typing gives programmers hint about the code and increases reliability, it also means a steeper learning curve, and more cumbersome syntax. This will probably slowdown our prototyping speed.

As for now, Tensorflow has already provided language bindings for Java[5]. However, this library is still experimental and stability is not guaranteed . However, it still indicates Java provides a viable alternate to Python with all the Pros stated.

4 OCaml

OCaml is the main implementation of the functional Caml programming language. It is statically and strongly typed and uses type inference to check variables' types. Ocaml supports concurrent, asynchronous and event-driven programming with Async package[6].

4.1 Pros

OCaml has a similar garbage collector as Java, which uses a mark and sweep algorithm[7]. This frees programmers from allocating and deallocating memories. Also, OCaml's static type-checking makes sure mix-up of data types do not happen during runtime, thus increase reliability. Also, OCaml has an experimental implementation of binding with Tensorflow, which makes it a fast alternate choice of Python.

In terms of performance, Ocaml is generally a good choice. Ocaml is generally as fast as C[8]., and is more than ten times faster than Python code[9]. This makes OCaml a good choice in the context of many small queries, as it saves the time of setting up models which has now become the bottleneck.

4.2 Cons

Being a functional programming language, Ocaml is known to have a very steep learning curve. The static type-checking, together with its type inference functionality, makes it hard for programmers to have an accurate handle of the data types among functions. This makes prototyping of the server comparably slower as compared to other candidate programming languages, especially during most of the time, OCaml code is the least readable.

In terms of parallelism, OCAml is similar to Python, which has a global lock prevents multiple OCaml threads from running simultaneously[10]. This prevents OCaml from having "true" parallelism which can enhance server throughput, though this factor is not very important as analysed in Section 3.1 "Pros of Java".

5 Kotlin

Kotlin is a cross-platform, statically typed, general-purpose programming language with type inference. It has a built-in garbage collector just like the previous programming languages. Since it can be compiled on JVM, its garbage collector uses a similar mark and sweep algorithm. It also supports asynchronous, event-driven programming as needed in the given context[11].

5.1 Pros

Similar to Java, Kotlin has a learning curve slightly steepr than Python. However, it is still a better choice for prototyping as compared to OCaml. Kotlin has all the advantages that Java has, but what makes Kotlin shines over Java is its support for coroutines. It allows users to write async code in a blocking fashion. Though Java also has async execution, user has to handle the UI thread and a network call at the same time, which is not as easy.

5.2 Cons

Kotlin does not have very obvious cons, however, in the specific context of creating a server that runs Tensorflow application, it is disappointing to find that Kotlin is not directly supported by Tensorflow. Though language binding is still possible when using Kotlin[12], it is less stable and more difficult as compared the other 3 languages, due to the lack of support.

6 Conclusion

I have given a brief introduction of Python, Java, OCaml, and Kotlin, and analysed their pros and cons in the specific context of creating a server that runs Tensorflow application. Though Python language is the bottleneck when there are many small queries due to its bad performance (partially due to dynamic type-checking), it is still a good choice of implementing this project simply because of the stable Tensorflow module provided. Java and Kotlin can be good alternates for Python because it will alleviate the bottleneck of setting up models due to its higher performance. Also, since the servers only need to handle small queries, instability of indirect support may not be a major issue. However, OCaml is not a good alternate for Python. It not only has indirect support of Tensorflow package, but its functional programming nature also makes it hard to create prototypes.

References

[1] Python (programming Language) - Wikipedia https://en.wikipedia.org/wiki/Python_(programming_language)

[2] Python Wiki, https://wiki.python.org/moin/GlobalInterpreterLock

[3] Guide To Java.util.concurrent.future Baeldung - https://www.baeldung.com/java-future

[4] Mark-and-sweep: Garbage Collection Algorithm https://www.geeksforgeeks.org/mark-and-sweep-garbage-collection-algorithm/

[5] Install Tensorflow For Java — Tensorflow https://www.tensorflow.org/install/lang_java

[6] Real world OCaml, Chapter 18. Concurrent Programming with Async, https://v1.realworldocaml.org/v1/en/html/concurrent-programming-with-async.html

[7] Real world OCaml, Chapter 21. Understanding the Garbage Collector, https://v1.realworldocaml.org/v1/en/html/understanding-the-garbage-collector.html

[8] Why Ocaml? https://www2.lib.uchicago.edu/keith/ocaml-class/why.html

[9] My Os X Programming Blog https://sixthhappiness.github.io/articles/python-scheme-and-ocaml-speed-comparison/index.html

[10] Concurrency, Parallelism, and Distributed Systems https://ocamlverse.github.io/content/parallelism.html

[11] Coroutines For Asynchronous Programming and More https://kotlinlang.org/docs/reference/coroutines-overview.html

[12] Tensorflow in Kotlin/native https://juliuskunze.com/tensorflow-in-kotlin-native.html