Alternatives to TenserFlow in Python

Aashita Patwari
University of California, Los Angeles

Abstract

TenserFlow is a powerful tool for building machine learning algorithms and applications. Ordinarily, TenserFlow uses Python as a high-level language to define the computation as a dataflow graph, following which the computation is done in C/C++ [6]. The bottleneck then arises from performing the computation on large queries. However, my application is unusual in that it handles may small queries that are tiny by machine-learning standards, and as a result, more time is spent setting up the models than executing them. Hence, in this paper, I am evaluating three other languages that are plausible candidates for implementing my application to reduce the bottleneck – Java, OCaml and Kotlin.

Introduction

As this is an extension of the project, the program must support event-driven servers too, along with reducing the bottleneck mentioned earlier that arises from creating the dataflow graph to do the computation in the high-level Python language.

There are three alternatives here being evaluated – Java, Ocaml and Kotlin. Java is a language used for imperative programming and is object-oriented. OCaml is a functional programming language. Kotlin is designed to be a 'better' version of Java and is popularly used for Android applications but has uses for server side, native and JavaScript development as well. These languages are being compared against Python, which is what is typically used by TenserFlow applications for large datasets.

1. Java

Advantages

Java is an object oriented, imperative language that was designed to be easy to use. The fact that it is object oriented means that developers are able to create modular programs and reusable code.

Additionally, Java is platform-independent which means that it is able to move from one system to another easily. This is because it is compiled in bytecode language and is able to run on any machine that has a JVM installed, regardless of the hardware's specific architecture [2].

Another feature of Java is that it is statically typed. This means that all variables must have their types declared, as contrasted to Python, where variables are dynamically typed and inferred at run time. This results in Java having a better runtime performance than Python in general. Apart from efficiency, static typing also allows for greater reliability compared to dynamic typing as type errors are detected at compile time as compared to runtime.

Java also uses an automatic mark and sweep garbage collection approach as compared to Python, which uses reference counts. This increases the code's efficiency as using reference counts causes assignments to be more expensive. In comparison to C, an automatic garbage collection approach also reduces bugs as it shifts the responsibility of managing memory away from the programmer.

Additionally, another important feature of Java is its support for multithreading as compared to Python, which uses a Global Interpreter Lock that hinders parallelism from being maximized. In our application, we could potentially utilize this to schedule each small query on a different thread to improve on the bottleneck.

Furthermore, as required by our application, Java supports building event-driven servers. It has support for asynchronous programming which means that it can be used to build non-blocking, event-driven programming models. The CompletableFuture interface recently introduced in Java8 comes together with methods that allows the programmer to attach callbacks and together, asynchronous computation is relatively straightforward to do in Java [4].

Disadvantages

The main disadvantage of Java as compared to Python is the need to statically type variables and the added work for the developer to add multithreading support and making sure the code compiles. However, since the focus here is on the performance of the application as compared to how easily it can be written, this is not an important disadvantage.

Another notable thing is that the TenserFlow Java API is not currently covered by the TenserFlow API stability guarantees. This means that some API functions can change in backward incompatible ways when minor releases are made. This affects the sustainability of the program as every time an update is made, the programmer might have to go through the entire code and fix bugs cause by backward incompatibility [7]. However, this applies to all the TenserFlow APIs in languages other than Python and C.

2. Ocaml

Advantages

OCaml is strongly statically typed but uses type inference, where the compiler infers the type of variable instead of the programmer having to explicitly state it. This ensures that OCaml is able to advantage from the performance and reliability benefits of a statically typed system, while getting the convenience of writing code like you would in a dynamic system [9].

Furthermore, machine learning algorithms are heavily reliant on functions and hence using OCaml would be natural as it is a functional programming language [5].

Additionally, OCaml implements its own garbage collecter that uses mark and sweep and generational garbage collection and hence does not require explicit freeing.

It also supports building event-driven servers using the Async library. This library provides high level APIs and syntactic sugar that simplify the code with APIs such as Tcp.Server.create that is similar to python's create_server. OCaml's type system also knows about Async's primitives. This is very useful as the programmer can be confident at compile time about which code blocks and which does not, and type errors make it obvious when asynchronous and synchronous code is being mixed [8].

Disadvantages

Ocaml uses a global interpretor lock, similar to Python and hence is unable to maximize parallelism.

From a developer's point of view, OCaml is less convenient to use as the developer has to debug all type errors before being able to compile the program. It is also trickier to debug in OCaml because the OCaml bindings do not provide line numbers for where the mismatch is coming from [3]. Functions are also less flexible and hence, the developer might need to write more code to implement the same functionality as compared to Python.

Additionally, there are very few resources on how to do machine learning in OCaml, especially as compared to Python.

3. Kotlin

Advantages

Kotlin offers many advantages that Java does too. Most notable of these is the portability of the byte level code such that it can be deployed on a wide range of platforms, and the ability to exploit parallelism more effectively as compared to Python. Kotlin is also statically typed, allowing for performance benefits as well as reliability in eliminating type errors at compile time.

Kotlin also improves on issues that were not addressed by Java such as null pointer exceptions, allowing for safer code. It also supports functional programming, which could be useful for this use case as described in OCaml's advantages.

Additionally, Kotlin provides more flexibility for the programmer as it uses type inference unlike Java. It also provides flexibility in asynchronous programming by providing Coroutines support and delegating most of the functionality to libraries – similar to the approach used by Python [1]. For example, there is a launch function that is used to start a coroutine, similar to Python's start_server. With coroutines, one can write asynchronous code as conveniently as synchronous.

Disadvantages

Being statically typed, Kotlin is less flexible compared to Python, which has more language support for numeric processing which is a dominant part of machine learning algorithms.

Furthermore, Kotlin is a newer language with less documented support and no TenserFlow bindings as of yet.

This would mean that there is significantly less built in functionality that a developer can exploit.

Conclusion

Since the main purpose of using an alternative language is performance, it appears that Java or Kotlin would be preferred over OCaml. Kotlin generates similar bytecode to Java and the performance of the two is comparable. However, for a developer, Java will be preferred as it is a more established language with better documentation. Furthermore, there are Java bindings for TenserFlow by the TenserFlow developers themselves, unlike the other two languages. This ensures that there is a greater degree of reliability for programs built in Java.

References

1. Asynchronous programming techniques, Retrieved Mar 15, 2019 from https://kotlinlang.org/docs/tutorials/coroutines/async-programming.html

2. Benefits of Java over Other Programming Languages, Retrieved Mar 15, 2019 from https://www.invensis.net/blog/it/benefits-of-java-over-other-programming-languages/

3. Deep learning experiments in OCaml, Retrieved Mar 15, 2019 from https://blog.janestreet.com/deep-learning-experiments-in-ocaml/

4. Java 8: Writing asynchronous code with CompletableFuture, Retrieved Mar 15, 2019 from https://www.deadcoderising.com/java8-writing-asynchronous-code-with-completablefuture/

5. Neural Networks, Types, and Functional Programming, Retrieved Mar 15, 2019 from http://colah.github.io/posts/2015-09-NN-Types-FP/

6. TenserFlow Architecture, Retrieved Mar 15, 2019 from https://www.tensorflow.org/guide/extend/architecture

7. TenserFlow Version Compatibility, Retrieved Mar 15, 2019 from https://www.tensorflow.org/guide/version_compat

8. Why Async? Retrieved Mar 15, 2019 from https://opensource.janestreet.com/async/

9. Why OCaml? Retrieved Mar 15, 2019 from https://www2.lib.uchicago.edu/keith/ocaml-class/why.html