Language Bindings for TensorFlow

Abstract

TensorFlow is an open source machine learning library that utilizes dataflow graph before performing computations. It has many language bindings built on top of the C API. Therefore, we are looking for other languages that could replace Python due to its expensive setup time, and also keep the TensorFlow and still run efficiently. We mainly compare Java, Ocaml, Python and Kotlin against each other with respect to ease of use, generality, reliability and performance; and we found out that Kotlin could be the best fit among them if we were to use to to handle many small queries in the server proxy herd application.

Introduction

In our server proxy herd application, we design the server herd architecture in such a way that it can handle rapidly-evolving data. In other words, it needs to deal with many small queries, which will be considered tiny by machine-learning standards. Therefore, although typical TensorFlow applications are bottlenecked indie C++ or CUDA, the previous application will be costly when executing Python code because it spends most of its time to set up models. Thus we want to know if we could speed up performance without giving up TensorFlow by looking to other languages to replace the application's Python code, such as Java, Ocaml and Kotlin, and compare them against each other, especially with respect to ease of use, flexibility, generability and reliability.

Java

Java is a statically typed language that supports concurrency and functional programming, plus automatic garbage collection and simple memory management. Moreover, Java code is compiled into bytecode which is highly optimized by the Java compiler and thus JVM can execute Java applications at full speed. Considering the above features, we can conclude that Java can offer us better reliability and higher performance but less flexible and ease of use compared to Python. When it comes to using TensorFlow API, Python will be a better choice since it is a dynamic type checking and known for its simple syntax. Plus, we don't need to focus too much on the type when implementing servers herd with TensorFlow API. However, in terms of performance, Java has the upper hand because of its multithreading capabilities and that it is a compiled language. For the server proxy herd application, Java's multithreading capabilities will boost up performance a lot since the servers need to handle many small queries; and multithreading are really good at working on many small mutually independent tasks. For example, if one thread is waiting for the results from Google Places API, others can process inputs and parse the messages; and if the command in the message is IAMAT, other threads can store the message in the server dictionary. Java's multithreading capabilities will be perfectly fit in in such cases. Furthermore, The JIT compiler is part of the Java runtime environment, which improves performance of Java programs by translating Java bytecodes into native machine code "just in time" to run; and then JVM can call the compiled code directly. Moreover, since Java is not a fully interpreted language, Java requires less processor time and memory usage. On the other hand, Python is an interpreted language which will slow down Python programs during runtime. Also, because of static type checking, Python determines the variable type during runtime, which will increases the workload of the interpreter. Finally, in terms of generality, both Java and Python have their own built-in garbage collection. Also, Python supports multiple inheritance. Although Java does not have multiple inheritance, interface inheritance implementation is a good enough substitute. In addition, Java is a server side language but Java applet indicates that Java also supports client side application so both Java and Python can be run on both server side and client side; and thus both are able to answer queries directly on the client.

Ocaml

Ocaml is a ML-derive functional language with a strongly static type system, type inference and automatic memory management and more. Omcal's static type checking is similar to the Java's because Ocaml also eliminates type-related runtime problems at compile time. On the other hand, Ocaml is also similar to Python's type inference, except that Ocaml does type inference at compile time while Python does it at runtime. Unlike Python, Ocaml's static type checking prevents runtime type mismatches and thus removes runtime type and safety checks that burden the performance of Ocaml programs. And unlike Java, Ocaml's type-inferring compiler greatly reduces the need for the manual type annotations that are required in most statically typed languages. Therefore, Ocaml offers reliability and some flexibility, as well as high performance at the same time. In addition, some functions in the Ocaml standard library are implemented with faster algorithm than equivalent functions implemented by other languages because Ocaml exploits the immutability of input data and reuse parts of the inputs in the output. Because of the immutability of data in Ocaml, functions in Ocaml have no side-effects while Python may have some serious side effects. In terms of ease of use, Ocaml's pattern matching plays a big role in its ease of use because one can do almost anything in Ocaml just by using pattern matching. Ocaml's pattern matching makes the code more concise and easy to read. However, although there are some Ocaml language binds for TensorFlow, most of the training libraries in TensorFlow are Python-only, which means Ocaml has limited support from TensorFlow API and we need to write our own if we need those training libraries. Therefore, if we want to keep TensorFlow, we may need to give up Ocaml for Python. In terms of generality, OCaml was originally used to develop applications that involve symbolic computation but now it's widely used for teaching programming.

Kotlin

Like Java, Kotlin is a statically typed language; and like Ocaml and Python, Kotlin support type inference. Also, since Kotlin has full Java interoperability, Kotlin can run on JVM and thus uses the same garbage collection mechanism as Java does. Yet Kotlin's syntax is intuitive and easier to understand than Java. There is no Kotlin binding for TensorFlow yet and thus there is no library that supports Kotlin programs. In addition, Kotlin has <Any> type for data, which means it can accept different type of arguments. With type inference feature and <Any> type, Kotlin is more flexible than Java and Ocaml. In terms of performance, similar to Java, static type checking and being a compiled language give Kotlin programs reliability and high performance. Moreover, Kotlin provides null safety, which Java does't have. It is aimed at eliminating the danger of null references from code and thus reduce runtime errors. Kotlin not only supports multithreading but also coroutines. This gives Kotlin the upper hand on Python in the server proxy herd application since Kotlin can use parallel coroutines to handle many small queries.
Since Kotlin run on JVM and is fully compatible with Java, Kotlin will be as portable as Java. Finally, in terms of generality, Kotlin can be used for different kind of development, such as server-side, client-side web with Javascript and android mobile application.Therefore, Kotlin is capable enough to deal with server-side application as well as client-side web. In server proxy herd application, query can be directly answered on the client without shipping the problem off to the servers. Since Kotlin is really similar to Java and has full Java interoperability, with Java already binding with TensorFlow, it's may not be hard to develop Kotlin binding for TensorFlow; or Kotlin could even reuse Java binding for TensorFlow.

Conclusion

Although most of the training libraries in TensorFlow are for Python-only, it's too costly to set up models using Python code in terms of server proxy herd application. In terms of generality, Kotlin has a upper hand on Java since Kotlin supports both server-side applications and web applications. Although Java has a program called Applet that can be embedded into web page, Kotlin can collaborate with Javascript and uses its libraries. It means Kotlin have more support and resources when working on web application than Java does. Also, while Ocam is fast and reliable, Ocaml does not have much utilities and libraries, considering that Kotlin is also fast and reliable but with many well-documented libraries; and Kotlin can even use Java libraries. Kotlin's coroutines and multithreading feature allow Kotlin programs to handle many small queries easily and fast. Therefore, I would suggest that Kotlin is a better candidate to replace Python.

References

[1]*Java Applet Basics.* Available: https://www.geeksforgeeks.org/java-applet-basics/

[2]*Language Design Criteria.* Available: http://www.cse.aucegypt.edu/~rafea/CSCE325/slides/2/LanguageDesign.pdf

[3]*Java vs Python Performance.* Available: https://www.snaplogic.com/glossary/python-vs-java-performance

[4]*Ocaml.* Available: https://en.wikipedia.org/wiki/OCaml

[5]*Ocaml Document.* Available: https://ocaml.org/learn/description.html#A-Widely-Used-Programming-Language

[6]*Kotlin Expertise Blog.* Nov 05, 2018. Available: https://kotlinexpertise.com/kotlin-coroutines-concurrency/

[7]*Kotlin Document.* Available: https://kotlinlang.org/docs/reference/faq.html