Homework 6: Language Bindings for TensorFlow

Shannon Hoang
UCLA CS 131 Professor Eggert Winter 2019

Language Bindings for TensorFlow

1. Intro to Python

Python is an extremely popular general purpose interpreted language that is used for web application development, systems management, big data, machine learning, and other applications. As a language, it is extremely readable and it was actually created with a major emphasis on readability and the use of whitespace to achieve this.

Because of its ease of use and asyncio library which allows us to easily write concurrent servers, it was the first programming language of choice for our server herd prototype.

However, after further testing, it was found that Python may not be best the best option for later iterations of our server herd as it has been found to cause bottlenecks when used with TensorFlow. As a result, the main topic of investigation today will be the comparison of Python to three other alternative languages -- Java, OCaml, and Kotlin – in terms of implementing our application server proxy herd with TensorFlow.

1.1 Intro to TensorFlow

Tensor flow is a Python Library for efficient numerical computing and was released by Google. TensorFlow was created with the intention easing the integration of machine learning into programs. More specifically it is a library of symbolic mathematics and neural networks that can be used to implement machine learning. At the core of TensorFlow lies the concept of a "neural network," which are made up of a large number of neurons that each perform their own simple mathematical operation. The neural network can take the results of each of these individual neurons and analysis the data to make predictions.

2.0 Java

Java is one other possibility for implementing our server-herd application. It was written with portability in mind. Since it is run on the JVM, it can be executed by various computer architectures. This flexibility is just one of many advantages of Java.

Java has massive support for multi-threading – i.e. the java.util.concurrent package which provides many tools for running code concurrently. The multi-threading possibilities in Java allow us to build code that is extremely efficient and would allow us to run our server herd in parallel. Another advantage Java possesses in efficiency is its methods for garbage collection – which are far more efficient than Python's. Java uses generational garbage collection and lazy garbage collection.

Another reason for Java's greater efficiency is the fact that Java programs are not compiled into executables but instead into byte code which the JVM (Java Virtual Machine) executes at runtime. Python in comparison is translated into intermediate code which is then executed by a virtual machine (PVM – Python virtual machine).

One reason Java is so reliable is its extensive system for exception handling. If an exception is raised, an Exception Object is automatically created and given to the JVM. Every Exception Object has a name and a description of the exception and the state of the program when the exception was raised. In addition, the Exception class allows programmers to control behavior when exceptions are raised.

Java is also extremely reliable due to its strict static type checking. At compile time, all the types of the variables are known, making it more likely for the compiler to catch errors in expression. Python on the other hand uses dynamic type checking, which means it delays its type-checking to run-time. Delaying type-checking to run-time makes Python slightly less reliable in that in can be less likely to catch certain errors.

However, a disadvantage of Java is actually tied to one of its strengths. Due to its strict type checking, abstraction is much more difficult in Java. Because Python has duck type checking ("if an object can use all the methods of another object are considered to be of the same type") there are many powerful methods that can be

applied across many different object types. For instance, to check if a certain object is in a general iterable one simply has to use the "in" function in Python. Java on the other hand because the types are so strictly separated has to have much more type specific methods, decreasing ease of use.

Lastly, the GitHub repository for TensorFlow for Java actually contains a warning at the top: "The TensorFlow API is not currently covered by the TensorFlow API stability guarantees." Therefore, while Java may have high reliability as a language, this may not necessarily be the case for using TensorFlow.

3.0 OCaml

OCaml is a member of the ML family and extends the Caml language to included object-oriented features, even though it is mostly intended to be used as a functional programming language.

The first advantage of OCaml is that it is a functional programming language, making it very suitable for machine learning. One advantage of functional programming languages in terms of machine learning is the closeness of the language to mathematical expressions. Another advantage of functional programming is that the programs themselves tend to be shorter and do more with less which allows for easier maintenance of code.

Overall OCaml is an extremely safe language. OCaml has type inference like Python. However, unlike Python OCaml is statically-typed and strongly typed, allowing more errors to be caught at compile-time if inferred types do not align where they should. This allows for a style of error checking that does not exist in many programming languages as the OCaml interpreter from inspecting types can point out various errors to the programmer that are not necessarily directly related to types.

For our specifications, OCaml has an async library of its own so we would be able to implement asynchronous functions easily. The Async library also has a TCP module that will allow us to make TCP connections with clients and between servers.

However, a big disadvantage of OCaml is its steep learning curve for imperative programmers. Since our original prototype was written in Python, our colleagues are likely mainly imperative programmers. Forcing the team members to switch to an entirely different style of programming may not outweigh the potential benefits of switching our implementation to OCaml.

In addition, unlike Java, OCaml in its current implementation cannot run in true parallel. The garbage collector for OCaml is not reentrant, meaning it cannot be paused and restarted reliably, which could cause in corrupted data during attempts to run programs in parallel. While OCaml has a library called Async_parallel that allows execution that is almost parallel, it does not allow for shared resources among processes.

4.0 Kotlin

As a language, Kotlin was partially written with the intent to take out Java, an extremely old and outdated(according to some). A huge advantage of Kotlin is that Java code is 100% convertible to Kotlin. Overall, Kotlin seeks to address many of the old problems of Java, while retaining many of Java's features. Another interesting feature of Kotlin is that while it possesses many object-oriented features, it also possesses many functional programming features. As a result, it can be seen as a combination of the two.

Kotlin is statically typed like Java so it is very reliable just like Java because the static type checking eliminates a lot of error. However, while Kotlin is strongly typed, it also supports type inference as well. Therefore, when declaring a mutableList for example, the programmer could explicitly declare the types of the elements in the list, or they could have the interpreter infer the types based on the elements of the list. Kotlin also is much less verbose as a language compared to Java which makes it much less error prone. Kotlin also is safer in that all types are non-nullable, eliminating the cause of most null references – a major problem in Java.

Kotlin has access to most Java libraries and frameworks. In terms of readability, Kotlin was designed with Java programmers in mind so the code is extremely readable for programmers coming from Java. Since Kotlin is run in JVM, it uses the same garbage collector and garbage collection as Java so the garbage collection is relatively efficient.

In terms of parallelism, Kotlin seemingly has the best of both worlds. For one, Kotlin actually has a thread function that allows for the creation of Java threads. Yet, Kotlin also has coroutines like Python if concurrency is preferred.

Kotlin is also extremely flexible in usage as you can import many different types of libraries (including Java libraries) to Kotlin.

Extension functions in Kotlin lend the language to even greater flexibility. Extension functions allow programmers to call a function from an object as if it were any other member function. As a result, they allow the modification of class behavior without ever touching class code.

However, Kotlin according to some does not have any real efficiency gains over Java(see *Kotlin's Hidden Costs)*. This is logical as the bytecode generated by Kotlin is very similar to that of Java.

In addition, because Kotlin is compatible with Java, both the Kotlin standard library and Java standard library are factored in during compilation – resulting in a larger file size.

Conclusion

In order to improve the efficiency of our application server proxy we have examined three alternative languages – Java, OCaml, and Kotlin. I argue that OCaml, while a very safe and efficient language, is not the best alternatively largely due to readability issues and the steep learning curve. Since our prototype was originally in Python, adjusting the team – which is likely most acquainted with imperative programming is not worth the potential performance benefit. This is even more so the case when we realize that Java can provide the performance increase that we desire with its ability to run programming truly in parallel with its multi-threading library. While Kotlin could be a better alternative to Java in the future – as Java is outdated in ways and possesses some problems – Kotlin is currently too young to reliably support TensorFlow. Most notably, Kotlin does not have bindings for TensorFlow yet or much official documentation and support for the language yet (compared to others). Kotlin may also not be suitable in that sense since our project is intended to be professional and it may still be best at an experimental level. As a result, we can see from analyzing these languages that Java is likely the best alternative to increase the performance of our server herd application.

References

Python Internals

https://www.python-course.eu/python3_execute_script.php

Static Typing

http://web.mit.edu/6.005/www/fa16/classes/01-static-checking/

Exception Handling in Java

https://www.geeksforgeeks.org/exceptions-in-java/

Functional Languages

https://hub.packtpub.com/what-makes-functional-programming-a-viable-choice-for-artificial-intelligence-projects/

Introduction to TensorFlow

https://machinelearningmastery.com/introduction-python-deep-learning-library-tensorflow/

Async Parallel

https://www.cs.cornell.edu/courses/cs3110/2014sp/recitations/19/async-parallel.html

Kotlin's Hidden Costs - Benchmarks

https://sites.google.com/a/athaydes.com/renato-athaydes/posts/kotlinshiddencosts-benchmarks

Kotlin- What's so special?

https://www.exlrt.com/blog/kotlin-what-is-so-special

On Python

https://www.python.org/doc/essays/blurb/