CS 131 Homework 6: Language Bindings for TensorFlow

Bradley Mont
UID: 804-993-030
University of California, Los Angeles

Abstract

TensorFlow, one of the most prominent open source libraries used in machine learning applications, is traditionally bottlenecked by either its C++ code or its CUDA code. However, when developing our own machine learning application using Python, we notice that the Python code to set up the models is taking the most time to execute. Keeping in mind that our application handles several small queries and that we wish to process queries without sending them to an application server when possible, we decide to explore alternative options besides Python for implementing our application. In this case, we investigate Java, OCaml, and Kotlin, and we determine if any of them are suitable replacements for Python in our application.

1. Introduction

For this project, we are developing an application server proxy herd on a large set of virtual machines. Our application uses TensorFlow for machine learning algorithms, and we've built a prototype using Python.

Our application performs sufficiently on large queries, but our application must also handle several small queries that require machine learning models. Surprisingly, we notice that our main bottleneck isn't the C++ or CUDA code like it typically is, but it is the Python code to set up the models. Therefore, we seek to increase the performance of our application by possibly using another language instead of Python that is better suited for our specific application. We also have a preference that we try to process all small queries directly on the client, opposed to sending them to an application server. There are cases in which server communication is necessary, but a majority of queries can be optimized this way.

Luckily, TensorFlow has bindings for several other languages besides Python, so we have several possible replacements for Python in our application. Specifically, we examine the programming languages Java, OCaml, and Kotlin. Java and OCaml both have explicit bindings for TensorFlow, and although Kotlin does not yet have bindings for TensorFlow, we can use TensorFlow in Kotlin/Native. Focusing on ease of use, flexibility, generality, performance, and reliability, we analyze Java, OCaml, and Kotlin as replacements for Python in our application, and we determine the optimal language to use in our proxy herd application.

2. Python

Python, one of the most widely used programming languages today, is a general-purpose, interpreted language with an extensive collection of libraries for many different areas of programming [1]. Python is a strongly, but dynamically typed language, meaning that all type consistency checks happen at runtime. Additionally, Python uses what is known as duck typing, where we care about the functionality, not the explicit type of the variable. Finally, Python has automatic memory management and can only execute a single thread at a time [1].

2.1. Advantages of Python

In terms of ease of use, Python is the clear favorite because it is known for having extremely readable syntax that is almost pseudocode-like. Another factor that contributes to Python's ease of use is the fact that we do not need to include type annotations since Python is dynamically typed. This also allows Python code to be extremely general since we do not have to declare all of our variables as strict types before execution, like we have to do in a statically typed language like Python. Python's automatic garbage collection also lifts the burden of memory management away from the programmer as well.

Not to mention, Python is an extremely flexible language; since Python is interpreted and not compiled into source code, it can run on any machine that has a Python interpreter, without having to worry about compatibility between different compilers, executables, etc. [2]. Plus, Python supports "object-oriented, functional, imperative, and procedural programming styles," so it definitely is flexible in which programming paradigm you choose to code in [2]. Including the `asyncio` library which fits our needs for an application server herd, Python's extensive standard library makes it extremely flexible in what you want to develop in Python.

For our specific application, Python also has the advantage of being the most common user-level language used with TensorFlow. Although there are bindings for other languages, they might be missing some core TensorFlow features that we know Python will have.

2.2. Disadvantages of Python

At runtime, Python is not one of the most performant languages. While a statically typed language like Java does its type checking before the program runs, Python has to check types as the code is being interpreted, which hurts performance. The dynamic type checking also makes Python less reliable than statically typed languages since Py-

thon code is more susceptible to runtime errors and undefined behavior.

Although relatively simple, Pythons' approach to garbage collection has several drawbacks. Since Python's garbage collection relies on reference counts, Python cannot have any race conditions when updating reference counts, or else memory leaks or freeing in-use memory could occur. As a result, only one thread can execute at a time in Python, and Python uses a Global Interpreter Lock to accomplish this [1]. Therefore, even though Python has fast single-threaded code, it does not support multithreading and cannot take advantage of multiple cores or CPUs. Not to mention, Python also takes a hit in performance because it has to constantly update and check the reference counts of objects for garbage collection. Finally, the reference count approach does not work for cyclical references, so Python will either have memory leaks or have to do mark and sweep in that case.

3. Java

Java, a general purpose, object-oriented, statically typed language, is one of today's most prominent programming languages. Java lies somewhere between a compiled and interpreted language because Java code is compiled into bytecode that runs atop the Java Virtual Machine. Java also has automatic memory management, great support for multithreading, an extensive standard library, and the ability to use JIT compilation for increased performance.

3.1. Advantages of Java

Much like Python, Java code is flexible since Java code is compiled into bytecode, and that bytecode can execute on any platform with a Java Virtual Machine installed.

Java's static typing also gives it an edge over dynamically typed languages like Python in the areas of performance and reliability. While dynamically typed languages do their type checking during runtime, Java checks types before runtime so it does not have its runtime slowed down by type checking. Also, Java is more reliable since we can catch much more errors before runtime and more easily debug them as well.

Another reason for Java's superior performance is its great support for multithreading, specifically with its `Thread` class. Plus, the Java Memory Model defines the behavior of multithreaded Java applications to satisfy the "As-If Rule," which states "you can implement a language any way you like, so long as it's done as if the model were executed." For short-term locks, Java has a `synchronized` keyword to put around critical sections and prohibit races inside of them. For longer-term locks, Java has the `wait`, `notify`, and `notifyAll` keywords. Finally, Java contains an extensive high-level synchronization library containing classes such as `Semaphore`, `Exchanger`, `CountDownLatch`, and `CyclicBarrier` [3].

Additionally, Java's memory management approach outperforms Python since it periodically performs mark and sweep, so it doesn't have the constant overhead of maintaining reference counts.

3.2. Disadvantages of Java

Although it increases reliability, Java's static type checking does make the language less flexible and general since we must know the type of all variables at compile time. Java does have great support for generic classes to combat this, but it is still less flexible and general than a language like Python.

Additionally, the TensorFlow for Java repository states that the "TensorFlow Java API is not currently covered by the TensorFlow API stability guarantees" [4]. Therefore, Java could lack some functionality than Python would definitely have.

4. OCaml

OCaml can be described as an "industrial strength programming language supporting functional, imperative, and object-oriented styles" [5]. Like Java, OCaml uses static type checking at compile time, but it is unique due to its strong type inference system. Type annotations are not necessary in OCaml; the compiler infers the type of variables based on context, which is important for parametric polymorphism as well. OCaml also uses automatic storage management and assumes the existence of a garbage collector. Additionally, it has great support for high-order functions and currying, and it has powerful pattern matching capabilities as well.

4.1. Advantages of OCaml

OCaml receives the same performance and reliability benefits as Java from its static typing, but OCaml also has increased ease of use and readability from the fact that type annotations are not necessary because of its type inference.

OCaml's strong type inference also makes it extremely flexible and general due to its support for parametric polymorphism, which is when the type of a function can contain parameters. For example, the `List.length` function in OCaml has type `'a list -> int`, which means that it can take a list of any type as a parameter, and it will return the length of it as an integer.

In addition, the OCaml documentation describes its garbage collector as "a modern hybrid generational/incremental collector which outperforms hand allocation in most cases," and it allocates a significantly smaller amount of memory than the Java garbage collector [6]. Finally, OCaml contains bindings for TensorFlow and has a large collection of libraries.

4.2. Disadvantages of OCaml

Although it has its benefits, OCaml's strict type inference severely hinders its ease of use because it can be extremely difficult satisfying these strict conditions, whereas a dy-

namically typed language like Python is much easier to use. Plus, OCaml is best-fitted for functional programming, which can have a much higher learning curve than imperative programming in a more traditional language like Java or Python.

OCaml is also extremely inflexible due to the fact that it does no implicit casting; for example, there are separate + and +. operators for integer addition and float addition respectfully, and OCaml doesn't implicitly cast between them.

Similar to Python, OCaml also uses a Global Interpreter Lock; therefore, OCaml's performance is worse than a language such as Java that can take advantage of multithreading and multiple cores. OCaml also faces the same problem as Java: there are bindings for TensorFlow, but there's no guarantee that they have all the functionalities that Python has.

5. Kotlin

Released in 2011, Kotlin is marketed as an improved version of Java. Therefore, one of Kotlin's main focuses is interoperability, specifically the ability to use existing Java libraries and migrate from Java to Kotlin. Also like Java, Kotlin is designed to work atop the JVM, so it has the same benefits as Java in areas like garbage collection and JIT compilation [7]. Kotlin is designed to be concise, safe, interopable, and tool-friendly, and some of its main applications are Android development, server-side applications, native applications, and Javascript development [8].

5.1. Advantages of Kotlin

First off, Kotlin is an extremely safe language due to its static type checking, as well as its functional aspect which prioritizes immutable collections and data types that are free of side-effects [7]. Plus, while Java programmers frequently encounter `NullPointerExceptions` when trying to use a reference that is actually null, Kotlin is much more null-safe [7]; in Kotlin, values cannot take on a null value unless they are explicitly indicated as "nullable" with a question mark after the type. So unless the programmer chooses nullable data types, Kotlin is free from `NullPointerExceptions`.

To reiterate, Kotlin has several of the same benefits that Java has, including increased performance due to multithreading and automatic garbage collection. Plus, with its data classes, type inference, and ability to make functions without belonging to a class, Kotlin has much more readable, general, and flexible syntax than Java [7].

5.2. Disadvantages of Kotlin

Unlike any of the previously mentioned languages, Kotlin does not have any binding for TensorFlow as of now, which is understandable considering that the language is relatively new. However, we can use TensorFlow in Kotlin/Native, but there could be less functionality compared to Python or even a language with bindings such as Java or OCaml.

6. Conclusion

After researching all four of these languages in the areas of ease of use, flexibility, generality, performance, and reliability, it is evident that there is no true perfect language; each of one these languages has its own benefits and drawbacks for using it to implement our proxy herd. However, we must look at the specifics that our application requires, and from there, we can conclude which programming language is most suitable.

As previously stated, we are trying to implement an application server herd using TensorFlow that can effectively handle several small queries, while minimizing the amount of queries sent to an application server. Out of the four languages discussed, Python would probably be the least efficient due to the overheard from its dynamic type checking and reference count-based garbage collection, as well as its inability to support multithreading; therefore, setting up the models in Python would have too much overhead.

Although more efficient than Python in this case, OCaml also cannot take advantage of multiple cores and run multithreaded code; if we are frequently receiving messages, processing them concurrently will give a huge performance boost, and Python and OCaml cannot support that. Plus, OCaml's inflexibility and strict type inference give it the worst ease of use out of all of the languages mentioned so far.

Finally, we are left with Java and Kotlin as the two best choices for our server herd, mainly due to their great support for multithreading, reliable static type checking, and mark-and-sweep based garbage collection that has less overhead than the reference count approach. Between these two, it is really the choice of the programmer since these languages have so many similarities. Java is optimal if you are looking for a language that has bindings for TensorFlow and is more well-known among programmers. On the other hand, Kotlin is optimal if you want all of the benefits that Java has, in addition to being more null-safe and concise; however, Kotlin does not have bindings to TensorFlow and is a newer and less-known language compared to Java.

References

[1] *TA Kimmo Karkkainen's Slides on Python*: https://piazza.com/ucla/spring2019/cs131/resources

[2] *How Flexible is Python?*: http://www.allaboutweb.biz/how-flexible-is-python/

[3] *Java 7 java.util.concurrent Documentation*: https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/package-ssummary.html

[4] *TensorFlow for Java Github Repository*: https://github.com/tensorflow/tensorflow/tree/master/tensorflow/java

[5] *Official OCaml Website*: http://ocaml.org/

[6] *OCaml Tutorials: Garbage Collection*: https://ocaml.org/learn/tutorials/garbage_collection.html

[7] *TA Kimmo Karkkainen's Slides on Kotlin*: https://piazza.com/ucla/spring2019/cs131/resources

[8] *Official Kotlin Website*: https://kotlinlang.org/