Language Bindings for TensorFlow

Kevin Chuang, University of California, Los Angeles

Abstract

TensorFlow applications typically spend most of the time bottlenecked inside C++ or CUDA code when running intensive machine-learning algorithms. But, under our circumstances, where our application server proxy herd runs many small queries and creates/executes many tiny models, our application is bottlenecked executing Python code to set up our models. We explore three different alternatives to Python for language choice to reduce our bottleneck.

1. Introduction

Our previous server application herd architecture has now been extended to set up and execute many tiny TensorFlow models based on very small queries. Typically, when we have TensorFlow applications, most of the execution time is spent inside C++ or CUDA code. However, because we have many small queries, the Python code that we are using to execute our machine learning algorithms mostly spends its execution time on overhead instead. To alleviate this problem, we consider three different languages including Java, OCaml, and Kotlin. We choose to make our decision on which language to use based off of ease of use, flexibility, generality, performance, and reliability.

2. Python

Python is an interpreted language that features duck-typing, automatic memory management, and is an imperative language. Since TensorFlow was originally developed for Python, starting out making TensorFlow applications is exceptionally easy for Python with good documentation and other developers for inspiration and help.

Python is a high-level language with duck-typing which in combination makes prototyping extremely easy. Since it isn't necessary to name variables, a programmer can focus mostly on the body of the code instead of having to ensure types of variables match. Unfortunately, this means that it can run into runtime errors if the types do not match up and hurts Python's reliability a bit.

Since Python is an interpreted language, it is extremely portable and any system can execute the program. But, since it isn't compiled it is slower because it isn't executing direct machine instructions.

In addition, Python is not multi-threaded and is subject to a global interpreter lock. Therefore, Python will have a lower throughput compared to other multi-threaded applications.

3. Java

Java is very different than Python and contains some crucial differences. In terms of ease of use, Java is harder to prototype than Python, partly because of its static typing and its lower level in comparison to Python. While Java has statically typed variables, which means that a programmer must set up the types of variables beforehand. In comparison, Python is duck typed which means a programmer does not need to specify what type variables are. This means a faster prototyping time and better ease of use. Python is also higher level and requires less code to perform basic coding functions, making it easier to start and programmers take less time to write out such code.

In terms of flexibility, Java is better than Python in that it is initially compiled to bytecode and can run on any computer architecture. Certain parts can also be compiled directly to machine code, which together makes Java both extremely portable while still being fast. While it wouldn’t be as fast as C, Java code is still faster than Python.

Java is much more reliable than Python because of its statically typed variables. Because all variables must be correct at compile time, Java programs can never run into a run time error unlike Python. It also has its own garbage collector that runs to free memory which leads to worse performance for more reliable, but Python is like it in that regards.

Java also has the advantage over Python in that it has good support for parallelism. Since Java supports multi-threading and Python does not, its servers would have a much higher throughput.

4. OCaml

OCaml is a functional language that while not explicitly supported by TensorFlow developers, there exist language bindings to use OCaml with the TensorFlow C API.

OCaml is by far, harder to use than Python. A functional programming style is harder to learn and understand and worse for prototyping for a TensorFlow server. This is partly due to not being able to set variables and having to heavily rely on recursion which can be hard to debug.

But, OCaml has a static type system but with type inference. This combines both the advantages of the reliability of static typing, but the ease of use that type inference brings. Unfortunately, since OCaml is a more confusing language, type inference can also be a problem in terms of reliability because you must write code that follows static type checking without knowing what it is yourself. This can make writing OCaml code harder than other languages and it can take awhile to compile code that does not have some sort of type error.

OCaml has its own garbage collector with mark and sweep, so while it may be slower than a language that manages its own memory, it is easier to write code for such programs.

OCaml is completely compiled, which means it has a better performance but worse flexibility and portability. A new executable will have to be generated for every system architecture, but on the other hand does run very fast.

OCaml is similar to Python in that it is subject to the global interpreter lock and can never achieve true parallelism. While it is like Python in that regards, a language like Java which supports multithreading would have a higher throughput than OCaml.

5. Kotlin

Kotlin is a cross-platform, statically typed, programming language that is similar to Java. Since it is a relatively new language, it leaves behind some of legacy code that is in Java for compatibility reasons and uses many of the new tricks and benefits that have come out of the programming field.

Kotlin, while statically typed, also has type inference. This means that it gains from the benefits of static typing in terms of reliability and the benefits of type inference in terms of ease of access. In addition, since Kotlin is not a purely functional language (while it does support it), it makes development much easier than it would be with OCaml and type inference does not cause problems as it does in OCaml.

Since Kotlin is run in JVM, it uses the same garbage collector as Java. This means an automatic memory management system with a garbage collection system that will automatically free memory like Python. This is an increase of accessibility and ease of use with the tradeoff of worse performance.
Kotlin is also like Java in that it is incredibly flexible being run on the JVM. The Java Virtual Machine can be run on any computer architecture and does not give up portability for good performance. While part of it is

interpreted, it means that it won't be quite as fast as a completely compiled language, but it is still relatively fast and is a good middle ground between portability and performance.

Kotlin supports threading like Java, but heavily emphasizes the use of coroutines instead. You can extend the Thread class from Java or use the thread() function that Kotlin provides. Kotlin instead has introduced a new way of writing asynchronous non-blocking code with a coroutine. These coroutines can run concurrently, wait for, and communicate with each other with a cheaper overhead cost.

Unfortunately, since Kotlin is a relatively new language, the support for the language isn't quite there yet despite all the benefits that come with it. TensorFlow doesn't fully support languages other than Python and C++ and is not at the level of support for Kotlin.

Conclusion

After studying all four languages, it seems that Kotlin would be one of the best choices due to its type inference, lack of legacy code, combination of interpreted and compiled code, portability, and its support for multithreading. Unfortunately, the reality is that since Kotlin is a newer language compared to the rest, it does not have the same level of support for TensorFlow applications as the other. So, while right now may not be the best time to prototype TensorFlow applications, maybe in the future it will be. Right now, the best bet would be to use Java, which is both portable, multithreaded, and is relatively fast in comparison to Python. In addition, since it is one of the most widely used programming languages, it will have support for TensorFlow in ways that Kotlin will not. Since our current Python source code is causing problems in terms of overhead, perhaps Java will be a better language to use.

6. References

[1] *Memory Management*. Python Software Foundation. Available: https://docs.python.org/3/c-api/memory.html

[2] Golubin, Artem. *Garbage Collection in Python: things you need to know*. Mar 14, 2019. Available: https://rushter.com/blog/python-garbage-collector/

[4] *What is OCaml?*. Mar 14, 2019. Available: https://ocaml.org/learn/description.html

[5] Build Safer Progams Faster With Ocaml. Mar 14, 2019. Available: https://www.endgame.com/blog/technical-blog/build-safer-programs-faster-ocaml

[6] Java Garbage Collection Basics. Mar 14, 2019. Available: https://www.oracle.com/webfolder/technetwork/tutorials/obe/java/gc01/index.html

[7] Multithreading and Kotlin. Mar 14, 2019. Available: https://medium.com/@korhanbircan/multithreading-and-kotlin-ac28eed57fea

[8] Threads vs Coroutines in Kotlin. Mar 14,2019. Available: https://www.baeldung.com/kotlin-threads-coroutines