Python, OCaml, Kotlin, and Java as Tensorflow Client Language

Computer Science 131, Professor Paul Eggert
Taylor Lee, 604790788

Abstract

In this report I investigate Python, OCaml, Kotlin, and Java as potential Tensorflow client languages and try to determine which language will best optimize an application server herd performance. The previous Python Tensorflow implementation resulted in bottlenecks when there was numerous but small queries to the servers. After investigating the efficiency, performance, and usability of these four languages, I conclude that a Java implementation will most likely improve the server herd performance due to its capacity for multithreading and optimization.

1. Introduction

Tensorflow is a software library commonly used to solve machine-learning problems. Developed by the Google Brain Team, Tensorflow is written in C++ but can be accessed and controlled by other client languages, most commonly Python. Client languages interact with the API through their language's foreign function interface (FFI), which mates the functions and calls of the client language with the API.

At a minimum, each client language should be able to support running a predefined graph and graph construction. However, Python, which is the most supported client language, supports many more features, including Gradients, Functions, Control Flow, and Neural Network Library.

This report investigates replacing the standard Tensorflow client language, Python, with either Java, OCaml, or Kotlin. Such an investigation is warranted because for the specified application server herd, the Python code was the bottleneck due to a large number of small queries to the application servers. These small queries execute models that are small in computing scale. Although usually the performance bottleneck is training the models (the processing done in Tensorflow), in this situation, the bottleneck is Python's bindings. Ironically, the Tensorflow models execute faster than Python can set them up.

I will investigate the four languages based on their respective ease of use, flexibility, performance, and reliability for Tensorflow bindings.

2. Python

Python is a popular, object-oriented, imperative language that is widely used because of its ease of use and readability. As the standard Tensorflow client language, Python supports the largest number of Tensorflow features.

2.1 Advantages of Python

Firstly, Python is a highly readable, understandable and commonly-used language. Such practical benefits must be considered when deciding which language to use for this application. Given that many developers are extremely familiar with Python and can quickly familiarize themselves with the language, such ease of use is highly regarded. Compared to a less commonly used language such as OCaml, Python readability allows easier debugging and configuring when programming with Tensorflow.

Additionally, Python has the largest number of features already supported by Tensorflow. Since Python is the primary client language for Tensorflow, it has the most documentation and support, which allows for extensive flexibility and debugging. This is especially useful if the Tensorflow models the application operates on are complex and need different feature supports.

Furthermore, Python's flexible type checking allows for developers to develop models without having to know all the specific low-level APIs provided with Tensorflow. Tensorflow is an extremely extensive application library and having to specifically declare individual types when programming can be cumbersome and unnecessarily difficult. With Python's duck checking, which doesn't require explicit type declaration, the Tensorflow bindings can be done in a much more simple and easy manner.

2.2 Disdvantages of Python

Python does not support multi-threading, which serves as a huge disadvantage for our application server herd. Multithreading allows the servers to optimize performance and execute the same amount of computation in less time. Since Python can only execute threads sequentially, if given a large query, our Python-implemented server herd would have to completely process that query individually, which can lead to bottlenecks when receiving multiple queries from different servers.

Also, Python is an interpreted language, and while this makes development easier since it avoids a compile step, this attribute comes at a cost of performance. Whereas Java programs are first compiled into executables and can be highly optimized during that compilation, Python exchanges ease of use for lesser performance since Python programs are never optimized executables.

Furthermore, Python implements a private memory heap that contains all Python objects and variables. When an object has a reference count of 0, the garbage collector collects the object. While this memory management is easier to use since developers do not have to worry about allocating memory as in C or C++, the garbage collector overhead can be detrimental to an application server herd since the garbage collector will constantly have to scan all objects and check their reference count. This differs in comparison with C++, in which the developer can specifically optimize the allocation and deallocation of memory.

3. OCaml

OCaml is a functional programming language. As an extension of the standard ML family, it supports object-oriented development and also is supported for Tensorflow bindings.

3.1. Advantages of OCaml

Firstly, as a functional language, OCaml is well-suited for a variety of numeric computations, including machine learning. The functional design of OCaml sets up mathematical modeling nicely.

Additionally, because OCaml statically type-checks at compile time, run-time overhead that is used in dynamically checked languages like Python is avoided, and OCaml has better performance. Furthermore, although it is statically type checked, OCaml does not require explicit type declarations, which gives developers flexibility when developing. This flexibility is especially useful when implementing a new and extensive library system such as Tensorflow.

Static type checking also has another big advantage. In queries that take a lot of computing power and time, dynamic type-checking languages have the capacity to throw a type error in the middle of operations after many computations have already occurred, rendering the previous computations useless. In statically checked languages like OCaml, such type errors are avoided completely since they are first checked at compile time.

3.2 Disadvantages of OCaml

As a functional programming language, OCaml's ease of use pales in comparison to Python, Java, and Kotlin. Since functional programming languages are fundamentally distinct from imperative languages, adopting OCaml for this Tensorflow application requires developers who are comfortable with developing in a language that may not be as familiar with. This steep learning curve must be a factor in determining which language to develop with.

As noted in tech blogs who have used OCaml to bind with Tensorflow, the Tensorflow feature support for OCaml is not nearly as extensive as Python's [1]. This lack of feature support in addition to the lack of resources and assistance available for OCaml Tensorflow bindings is extremely detrimental to the developing process. For a large server herd like our application, documentation and feature assistance can play a critical part in developing an efficient application. Without the support of the extensive libraries and documentation like Python's, application server herd development in OCaml can be difficult.

4. Kotlin

Kotlin is an object-oriented, statically-typed programming language that interoperates fully with Java and the JVM. Additionally, Kotlin supports useful functional features, such as the `map` function and lambda functions. Although Kotlin does not have a binding capacity with Tensorflow, there is a binding for Kotlin/Native which allows compilation in Kotlin.

4.1. Advantages of Kotlin

Kotlin was developed to be a successor and extension of Java but still be fully interoperable with Java [2]. This is a huge advantage when developing because Kotlin can be supported by Java and the JVM. In addition to being an object-oriented language, Kotlin also supports functional programming styles and differs from Java in allowing static methods to exist outside of a class. Kotlin supports greater flexibility than Java in instances like these, which is an advantage for Kotlin.

Like Java, Kotlin is statically-typed. Additionally, Kotlin offers type inference which allows for the system to determine the object type. This type checking is useful, because like OCaml, Kotlin can avoid run-time type check errors midway through large model computations by statically checking at compile time. Kotlin can also have better performance than a dynamically typed language like Python, since it avoids a lot of the overhead that comes with dynamically typed languages.

When dealing with high volumes of server traffic and information, Kotlin's statically typed checking and performance will start to outperform a dynamic system like Python's. Inherently, Kotlin performs better because of the optimizations that can be created at compile time.

4.2 Disadvantages of Kotlin

Naturally, there is not as much Tensorflow support for Kotlin as there is for Python, which serves at a disadvantage for Kotlin. Specifically for Kotlin (and Java), machine learning applications have more library support in other libraries, such as the newly released Deeplearning4J, which is a machine learning library that is written for Java and languages that use the JVM [3]. Therefore, for this applica-

tion, since we are specified to use Tensorflow, Kotlin may not be the most ideal language to use as a client language given that Tensorflow does not support Kotlin as well as it supports Python or other languages. However, further research should be done in implementing Kotlin in Deeplearning4J, since DL4J is made for the JVM and will have much more extensive library support and features.

5. Java

Java is an extremely popular, object-oriented, statically-typed programming language. Java programs are compiled into bytecode and are run on the JVM. Java is also supported by Tensorflow bindings.

5.1. Advantages of Java

One of the primary advantages of Java is its ability to multithread. This key capability allows for much greater performance than Python or OCaml, which do not support multithreading. Especially in our application server herd, which will intake multiple queries at different servers, the ability to multi-thread increases performance drastically. While Python implements concurrent programming through Python libraries such as `asyncio`, Java does not need to "fake" its concurrency because it can implement multiple threads that truly run concurrently. Such concurrency is much more scalable than Python

Additionally, as a compiled language that is interpreted with the JVM, Java provides much more reliability than Python. Optimizations done at compile-time can drastically improve the performance of the application server herd, especially in situations like our Python bottleneck in which numerous but small queries are overloading our Python framework. Compiling also provides another layer of reliability over Python. With a Java framework, developers don't have to worry about dynamic type errors midway through an extensive computation. The combination of compilation optimization in addition to multi-threading can provide significant performance boosts over our Python framework.

Because Java runs on the JVM, Java programs are also extremely portable, so not only are Java programs more reliable than Python programs, they have similar portability, which is another big advantage.

Furthermore, although Java may be slightly more difficult to learn given its static type checking, Java is still an extremely popular programming languages, and many developers have extensive experience with the language. Such ease of use is an important factor to weigh.

5.2 Disadvantages of Java

An important disadvantage of a Java framework is its statically type checking that requires explicit declaration of types. When implementing a comprehensive library like Tensorflow, this can be cumbersome and frustrating for developers who may not know the specific types of certain Tensorflow functions and would have relied on Python duck checking.

Additionally, like Kotlin, Java is not as extensively supported in the Tensorflow library. Like Kotlin, a Java framework would be more fitting with the DeepLearning4J library, which was specifically made for Java. This would be a compelling option to consider: not only just changing the client language of the server, but the entire server library itself. It would be an interesting comparison to see a DL4J library paired with a Java binding compared to the Tensorflow library with a Python binding.

6. Conclusion

A decision for a client language for Tensorflow is a multi-faceted one that requires considerations on the application server herd's needs, queries, and performance issues. Depending on the application, different languages better suit different server herds.

Because this application called for a new language binding that would best improve the performance of our server that intakes numerous small quires, I would switch from Python to Java, primarily because of Java's multithreading, static type checking, compile-time optimizations, and reliability. This switch would come at a disadvantage to user ease of use, portability, and Tensorflow feature support. An investigation worth researching would be a server herd using DL4J along with Java bindings and comparing it with our current Tensorflow implementation.

7. References

[1] Mazare, L. (2019). *Deep learning experiments in OCaml*. [online] Jane Street Tech Blog. Available at: https://blog.janestreet.com/deep-learning-experiments-in-ocaml/ [Accessed 7 Jun. 2019].

[2] JRebel.com. (2019). *JVM Languages Report: Interview with Kotlin Creator*. [online] Available at: https://jrebel.cxzom/rebellabs/jvm-languages-report-extended-interview-with-kotlin-creator-andrey-breslav/ [Accessed 7 Jun. 2019].

[3] Sheil, H. (2019). *What Java needs for true Machine / Deep Learning support*. [online] Medium. Available at: https://medium.com/@hsheil/what-java-needs-for-true-machine-deep-learning-support-1571ffdbb594 [Accessed 7 Jun. 2019].

[4] Eggert, Paul. *Project. Proxy Herd with Asyncio*, web.cs.ucla.edu/classes/spring19/cs131/hw/pr.html.