Language Bindings for TensorFlow

Abstract

The TensorFlow platform is an open-source, machine learning platform equipped with comprehensive and flexible tools for specifying and executing complex computation models [1]. Generally, programs that utilize TensorFlow will specify the model with a high-level language such as python for pre-processing purposes and execute the model in a low-level, more performant language such as C or C++. This paper will focus on the model specification step to improve performance by comparing and contrasting Java, OCaml, and Kotlin as alternatives to python. Careful examination will be done to determine the effectiveness of each language and its TensorFlow bindings in the context of event-driven servers along with their performance and reliability impact on the program. Overall, the goal is to reduce the model specification overhead by considering alternative languages for this process.

1 Introduction

The overhead of specifying dataflow graph models and executing the actual models depends on the size and complexity of the models. Generally, when the models are complex and large, most of the time is spent executing the models in low-level languages like C or C++. However, when the models are simple and small, the overhead lies mostly in the specification step developed in a high-level language like python [2]. When a single program defines and executes many simple models, the data pre-processing, set up, and specification steps become bottlenecks in the TensorFlow application. Therefore, we will need to figure out how to minimize this model specification overhead by exploring three alternative languages to python: Java, OCaml, and Kotlin. We will explore each language's performance when specifying TensorFlow models, support for event-driven servers, reliability, generality, flexibility, and learning curve for both newer and experienced developers.

2 Software Goals

Each type of application will prioritize different software goals and use cases. Only once software goals are established can we begin to consider suitable options for the application. In particular, this section will describe two common use cases of the application. The application we are researching to build will be a server proxy herd application running on virtual machines. The common cases that must be optimized are handling numerous, small queries and creating many small machine learning models.

2.1 Handling Small Queries

The software goal of being able to handle small queries efficiently will affect how we consider different languages. In terms of handling queries, there are generally two types of methods. A single-threaded server can continually poll for if a request has been received and process it accordingly. If many requests come at once, the server must be able to store and queue each request. However, a better approach would be to asynchronously react to queries and events, processing each request in a different thread. This multi-threaded approach allows tasks to execute in parallel while maintaining an event-based architecture that avoids polling and executes code when notified of a change. When considering languages for the proxy server herd, it is important to evaluate if the language supports this multi-threaded, asynchronous approach.

2.2 TensorFlow

Another software goal for this application is the successful implementation of creating and executing many small machine learning models. As mentioned in the introduction, we will focus on optimizing the model specification step. When considering languages to implement this step, we need to explore how easy it is to set up the model, how performant this process is, and how flexible the language is in terms of specifying the nuances of each machine learning model.

3 Language Comparisons

In this section, we will compare and contrast the effectiveness of utilizing python, Java, OCaml, and Kotlin for the model specification portion of the program. Along with commenting on the language's basic features, we will compare their performance, support for event-driven servers, reliability, flexibility, and learning curve.

3.1 Python

Python is a popular high-level language that focuses on the readability and ease of use of the language. The language is dynamically typed with a concept called *duck typing*, where the semantics of a class are determined by its ability to respond to some method or property [3]. In other words, if it looks like a duck, it is a duck. Python has automatic memory management and is usually written in an imperative style; however, it exhibits some functional features such as map, filter, and reduce. Below are some advantages and disadvantages for using python in the context of our application.

3.1.1 Python Advantages

The first advantage of python is that it supports event-driven server code well. Libraries like asyncio [4] and aiohttp [5] make it extremely easy to create asynchronous behavior in event-driven servers built on the underlying TCP and HTTP communication protocols. Another advantage of python is that it is the first language supported by TensorFlow and is also the most popular language that supports the most TensorFlow features [6]. By design, python has a relatively easy learning curve in terms of creating a simple server or getting started with TensorFlow. The ease of use comes from the minimal number of lines of code needed to write simple applications, the code readability, and the resources available online such as tutorials, in-depth documentation, and example code. The *duck typing* feature allows beginners to create prototypes without understanding the details of the libraries they are using. It also allows experts to code quickly without declaring explicit types for each variable. Because python requires no compilation due to being interpreted, it is more portable and is not machine specific. Interpreted byte codes improve the convenience of running the program independent of the machine it is running on.

3.1.2 Python Disadvantages

Unfortunately, python has disadvantages in the context of our application. The *duck typing* feature can be a source of error as developers may not fully understand how their program operates and may make false assumptions about library calls without understanding the return types of each method. Furthermore, the runtime type checks, although convenient for the developer, add a performance overhead to the program. Actively keeping reference counts and performing memory scans results in a slower performance compared to a language like C or C++, in which the users must allocate and deallocate memory themselves. Lastly, python is implemented with a global interpreter lock [7], which forces python to execute byte codes one thread at a time (no parallelism). This reduces the performance of a server that must handle multiple requests at a time, as multi-threaded approaches handle requests concurrently, taking advantage of multi-core processors.

3.2 Java

Java is an object-oriented language that is statically typed. It also has automatic memory management. Below are some advantages and disadvantages of Java in the context of a proxy server herd running TensorFlow.

3.2.1 Java Advantages

A key advantage of Java is the portability that comes with a Java Virtual Machine [8], which runs the byte codes that a Java program compiles into [9]. This means Java is more performant than a purely interpreted language but less performant than a language like C. Another advantage is the reliability of static type checking. Because the compiler will complain at any mismatched type, this ensures that no type errors will occur during runtime. Java's automatic memory management reduces the responsibilities of a developer and the development overhead and error-prone process of manually allocating and deallocating memory. Lastly, Java provides many options for multi-threaded programs. Built-in key words like synchronized or libraries like java.util.concurrent.locks [10] offer support for creating a multi-threaded server-herd application that concurrently handles queries while maintaining careful control over locked critical sections. Furthermore, libraries like NIO [11] allow for asynchronous event-driven functionality, an important software goal for this application.

3.2.2 Java Disadvantages

Java's statically typed feature means that applications are not as quickly as developed due to the development overhead of declaring types and waiting for a compiler to check for type errors. There is also performance overhead in the garbage collection process due to extra software periodically marking and sweeping each object during runtime.

3.3 OCaml

OCaml is a specific, modern implementation of ML. Although it supports the functional programming style of ML, it is not limited to this; it also exhibits both imperative and object-oriented features [12]. The language is statically typed but does not require the explicit declaration of the types. Instead, the types are inferred by the compiler to create an executable. OCaml can also act like an interpreter due to the language's top-level interactive loop.

3.3.1 OCaml Advantages

OCaml offers reliable execution of code due to its static type checking and automatic type inference. This is a performance advantage because no runtime type checking is necessary to ensure reliability. Because OCaml is compiled into machine code, there are many performance optimizations the compiler can do to take advantage of the current machine that the program is executed on. Another advantage of OCaml is the automatic memory management that runs a mark and sweep algorithm to automatically free memory when objects are no longer referenced. This simplifies the developer's job by eliminating the need to manually allocate and deallocate memory by tracking references to objects. OCaml also has support for developing asynchronous, event-driven servers with libraries such as Async [13] to asynchronously handle arriving queries.

3.3.2 OCaml Disadvantages

Currently, TensorFlow cannot be directly utilized in OCaml. This is inconvenient because a developer must use bindings for OCaml to the TensorFlow C API in order to specify machine learning models [14]. Furthermore, there is less documentation than python in terms of setting up OCaml code to work with TensorFlow. The functional development style of OCaml exhibits a steeper learning curve compared to more imperative languages. The compiler also strictly enforces consistent types. Thus, beginning to develop a simple prototype for a server, especially that of a server-herd architecture, becomes more difficult. Furthermore, the type inference mechanism increases the difficulty of conforming to the specification of TensorFlow API methods. Without the explicit declaration of types, it may be difficult for the developer to understand and keep track of types throughout the program. OCaml also has a performance overhead when it runs its automatic garbage collector during runtime. Furthermore, because OCaml is compiled into machine code, the OCaml programs are less portable by nature and may have different optimizations and machine code representations depending on the machine being executed on. Lastly, similar to python, OCaml is also limited by a global interpreter lock that eliminates the advantages of true parallelism and multi-core processors [15].

3.4 Kotlin

Kotlin is a programming language designed to interoperate with Java and the JVM version of Kotlin's standard library. It can also be compiled into JavaScript and Native code. Kotlin is statically typed and equipped with type inference, much like OCaml, which produces a syntax more concise than Java. Kotlin has both functional and object-oriented constructs, allowing developers to mix both styles of programming within the language [16]. Below are some advantages and disadvantages of the language relative to the context of a proxy server herd specifying TensorFlow models.

3.4.1 Kotlin Advantages

Kotlin is inspired by common languages like Java and JavaScript, so the syntax is easy to learn. The type inference feature makes Kotlin more concise than Java. It is estimated that Kotlin programs eliminate 40% of the number of lines of code in an equivalent Java program [16]. The static type checking improves the runtime performance of Kotlin programs. Kotlin is also more functional than Java because it offers higher order functions and lambdas [17]. Because Kotlin can run in a JVM like Java, it is also both portable and performant. Kotlin has high reliability due to the null-pointer exception feature as well as nullable data types declared with a ? postfix after the type name [18]. Kotlin supports event-driven servers because of the available options for multi-threaded, asynchronous code. It also has a rich API and well-documented library for co-routines [19], which are powerful for asynchronously handling a large number of incoming requests. Kotlin also has the same garbage collection that Java has, along with all of Java's other advantages due to Kotlin being 100% interoperable with the Java programming language. Common frameworks for server-side development such as Spring Boot [20], vert.x [21], and JSF [22] are available to help build an efficient proxy server herd.

3.4.2 Kotlin Disadvantages

Kotlin has no namespaces. If multiple functions with the same name exist within different levels in the package hierarchy, it is difficult to identify which function is being called [23]. Another problem is the lack of a static modifier. While Java supports static attributes, Kotlin requires annotations and function attributes to achieve the same effect. Kotlin's conversion tool from Java works well when it succeeds but can cause major problems when it fails. Furthermore, Kotlin does not have bindings to TensorFlow yet, but one can use TensorFlow in Kotlin/Native [24] as a workaround to this problem.

4 Conclusion

After researching python, Java, OCaml, and Kotlin as potential candidates for building a proxy server herd that handles many requests and specifies many small models, I think Kotlin is the best choice for the defined software goals. Kotlin is portable because it can be compiled into multiple forms (including Java byte codes). It is reliable due to static type checking and convenient and readable from its type inference. It supports multi-threaded, asynchronous code, and all of the Java benefits, including its documentation, surplus of libraries, and automatic garbage collection. Overall, Kotlin is an effective alternative to python, offering reliability without sacrificing performance.

5 References

[1] *TensorFlow*, https://www.tensorflow.org/.
[2] *Homework 6. Language Bindings for TensorFlow*, https://web.cs.ucla.edu/classes/winter19/cs131/hw/hw6.html.
[3] *Python Duck Typing*, https://hackernoon.com/python-duck-typing-or-automatic-interfaces-73988ec9037f.
[4] *Asynchronous I/O*, https://docs.python.org/3/library/asyncio.html.
[5] *Welcome to AIOHTTP*, *https://aiohttp.readthedocs.io/en/stable/*.
[6] *TensorFlow in Other Languages*, https://www.tensorflow.org/guide/extend/bindings. http://www.ocaml.org/.
[7] *Global Interpreter Lock*, https://wiki.python.org/moin/GlobalInterpreterLock.
[8] *What is the JVM? Introducing the Java Virtual Machine*, https://www.javaworld.com/article/3272244/what-is-the-jvm-introducing-the-java-virtual-machine.html.
[9] *Java Programming/Byte Code*, https://en.wikibooks.org/wiki/Java_Programming/Byte_Code.
[10] *Package java.util.concurrent.locks*, https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/locks/package-summary.html.
[11] *Package java.nio*, https://docs.oracle.com/javase/8/docs/api/java/nio/package-summary.html.
[12] *OCaml*, *http://www.ocaml.org/*.
[13] *Why Async*, https://opensource.janestreet.com/async/.
[14] *OCaml Bindings for TensorFlow*, https://github.com/LaurentMazare/tensorflow-ocaml. https://kotlinlang.org/docs/reference/faq.html.
[15] *Multicore OCaml*, http://ocamllabs.io/doc/multicore.html.
[16] *Kotlin FAQ*, https://kotlinlang.org/docs/reference/faq.html.
[17] *Higher-Order Functions and Lambdas*, https://kotlinlang.org/docs/reference/lambdas.html.
[18] *Null Safety*, https://kotlinlang.org/docs/reference/null-safety.html.
[19] *Coroutine Basics*, https://kotlinlang.org/docs/reference/coroutines/basics.html#structured-concurrency.
[20] *Creating a RESTful Web Service with Spring Boot*, https://kotlinlang.org/docs/tutorials/spring-boot-restful.html.
[21] *Vert.x Core Manual*, *https://vertx.io/docs/vertx-core/kotlin/*.
[22] *JavaServer Faces (JSF) Tutorial*, https://www.tutorialspoint.com/jsf.
[23] *Kotlin: the Good, the Bad, and the Ugly*, https://medium.com/keepsafe-engineering/kotlin-the-good-the-bad-and-the-ugly-bf5f09b87e6f.
[24] *TensorFlow in Kotlin/Native*, https://juliuskunze.com/tensorflow-in-kotlin-native.html.