CS131 Project. Language bindings for TensorFlow

Calvin Chen
University of California, Los Angeles

Abstract

Python *asyncio* library supports event-driven servers well. However, the application sever herd built handles many small queries that only executes tiny machine learning models. In this assignment, the time cost is mainly from application server herd to set up the *TensorFlow* models. In the study, I review *TensorFlow* Architecture and consider other three languages: Java, OCaml, and F# that are plausible to be a replacement for the original application server herd which queries heavily on *TensorFlow*.

1. Introduction

The *TensorFlow* runtime is a cross-platform library. This assignment mainly wants to improve the efficiency in the client layer written by users, which builds the computation graph for *TensorFlow*. *TensorFlow* supports various client languages. Although *TensorFlow* supports mainly in Python and most of the training libraries are Python-only, in application server herds, small queries may have negative impact on constructing or executing models, especially in Python, a less efficiency-oriented language.

2. Python

Python is a dynamic language and also a strongly typed language as the interpreter keeps track of all variables types. It features automatic memory management and interpretative execution. Python also supports multiple programming paradigms, including object-oriented, imperative, functional and procedural, and has a large and supporting object-oriented, imperative, functional and procedural, and has a comprehensive standard library.

The following describes the advantages and disadvantages of Python as the language for application sever herd for *TensorFlow*.

2.1. Advantages

Python is a dynamically typed language, which determines the object type in runtime. When constructing the graphs and machine learning models, this feature can lead to quick implementation since there is not much assumptions on the input objects.

Also, Python's memory management system makes it easier to develop because Python is a heap based memory model and is performed by the interpreter itself and that the user has no control over it. when the objects are no longer referenced, the system deletes it on its own. That is, developers don't need to allocate memory for *TensorFlow* objects on their own.

Moreover, Python's comprehensive library and open source package make it easier than other languages to implement application server herd and the machine learning models.

2.2. Disadvantage

The convenience of Python slows it down at runtime. Since Python is dynamically typed languages, it must perform type-checking at runtime. Also, Python's memory management system has to keep track of objects, so that it can perform garbage collection when the object is no longer used. These two things slow the constructing of *TensorFlow* graphs and machine learning models down. These could make the bottleneck in the application in Python code segment since the type-checking at runtime and the tracking of heap data take a lot of execution time.

3. Java

Due to the limitation of executing time, Java is one of the plausible candidates for implementing the application. Java is an object-oriented language that is concurrent, class-based and designed to be executable regardless of computer architecture since Java code is compiled to Java bytecode that can be executed by Java virtual machine.

The following describes the advantages and disadvantages of Java as a candidate for implementing the application.

3.1. Advantages

Unlike Python, Java is a statically typed language, which determines type during compile time. This makes the code more reliable and efficient than dynamically type, since it doesn't need to do type-checking when execution. Java is also a mature programming language with comprehensive library. The application server herd can be implemented by Java socket programming with NIO and NIO.2. For applications with more frequent I/O and asynchronous I/O, we need these non-blocking APIs introduced from Java. NIO package offers multi-

thread non-blocking I/O channels and NIO.2 extends non-blocking I/O libraries to support more tasks. In NIO.2, we can implement a echo server by NIO.2's *AsynchronousServerSocketChannel*, which is similar to Python's *asyncio*, a stream-oriented listening sockets, which allows TCP connections.

Another advantage of Java is that it is portable. According to the spec, the application server proxy herd is executed on a large set of virtual machine. Since Java bytecode runs on JVM, the application can run easily on different architecture. This compiled bytecode also has advantage on Python since Python code has to be interpreted in runtime. Moreover, Java type-checking is done in compile time so that it saves many time in execution.

The memory management system in Java is also more efficient than Python. Developers have to allocate memory on their own, this makes the garbage collection efficient because it doesn't have to keep track on objects.

3.2. Disadvantages

Java's efficiency in runtime make it complicated to build machine learning models in *TensorFlow*. Developers have to be focused on object type in *TensorFlow*'s graph.

Another issue in Java is that it performs asynchronous server by multithread. In this situation, if the computer architecture doesn't have enough processors, it may perform worse than Python *asyncio*.

4. OCaml

The second plausible language candidate for implementing the application is OCaml. OCaml is a functional programming language that unifies functional, imperative, and object-oriented programming under an ML type system.

4.1. Advantages

OCaml is a statically typed language, however, it emphasizes on performance. Its static type system prevents runtime type mismatches that burden the performance of dynamically typed languages, but it doesn't need developers to define the type, instead, it infers type by itself in compile time. Thus, it combines both the advantages in type checking in Java and Python.

On the other hand, OCaml has *Async* library, which combines both features in single-threaded event-driven systems and multi-threaded systems. This hybrid model provides the best of both worlds. This system has similar function to Python *asyncio* that supports TCP protocol and HTTP.

OCaml also combines the advantages in Python and Java in memory management system. In OCaml, the garbage collection doesn't keep track of objects constantly. This modification makes OCaml's garbage collection performs better than Python.

4.2. Disadvantages

OCaml is a powerful language, however, compared to Java and Python, functional programming is somewhat harder to implement. And its type checking system also restricts developer to carefully deal with the type in *TensorFlow* objects.

Although *Async* in OCaml is a hybrid model, it is basically similar to Python, which threads cannot access to a single object at the same time. In this case, this may somehow limit its performance from real multithread method.

5. F#

F# is the third plausible language candidate for the application. F#, as a member of the ML language, is a strongly typed, functional language. Originating from OCaml, F# also unifies functional, imperative, and object-oriented programming.

5.1. Advantages

Like OCaml, F# doesn't need programmers to declare types, instead, the compiler infers types during compilation. This static type system prevents runtime type mismatches that burden the performance of dynamically typed languages.

As a member of .NET framework, F# provides asynchronous workflows *async{…}* to do asynchronous programming. F# also provides parallel programming that run asynchronous blocks in parallel. F# also has similar functions to OCaml that it supports event-driven TCP protocol and HTTP in *async{…}*.

F# is also suited to machine learning efficient execution, data access capabilities and scalability. It is also well-used at Microsoft Research.

5.2. Disadvantages

Although F# is powerful, the popularity of F# seems to be a weakness. F# is a potential language that used for quantitative analysis, machine learning, optimization. It has been seen as an optimized alternative to C# in 2010s.

6. Conclusion

Each language described has some advantages over another. In prototyping and convenience, Python seems to be the best since it is dynamically type and with comprehensive libraries, it is easy for implementation. Java's portability and efficiency make it to be a great fit

with the application server herd since the goal is to deal with the bottleneck of client layer in *TensorFlow*. OCaml may also be a great fit because its type-checking and memory management also performs more efficient than Python. Last but not least, F# has a great potential to be the plausible language to implement application server herd. Though it is less popular than OCaml, Java, and Python, in performance aspect it seems to be a great match to the application.

References

[1] Tensorflow. See https://www.tensorflow.org

[2] Python. See https://www.python.org/

[3] Python (programming language). See https://en.wikipedia.org/wiki/Python_(programming_language)

[4] Java (programming language). See https://en.wikipedia.org/wiki/Java_(programming_language)

[5] Socket programming in Java for scalable systems. See https://www.javaworld.com/article/2853780/core-java/socket-programming-for-scalable-systems.html?page=2

[6] OCaml. See https://ocaml.org/

[7] OCaml. See https://en.wikipedia.org/wiki/OCaml

[8] Real World OCaml Chapter 18. Concurrent Programming with Async. See https://realworldocaml.org/v1/en/html/concurrent-programming-with-async.html

[9] F Sharp (Programming Language). See https://en.wikipedia.org/wiki/F_Sharp_(programming_language)

[10] F#. See https://fsharp.org/