An Investigation of Different TensorFlow Language Bindings

Jianzhi Liu
University of California, Los Angeles

Abstract

This paper discusses possible alternatives for Python in an Application Server Herd relying on TensorFlow. Three alternative programming languages are presented: Java, OCaml and F#. For each of them, pros and cons are examined in terms of reliability, performance, flexibility, etc. It is found that Python outperforms all three alternatives in flexibility and ease to use, whereas Java and F# provide more convenient concurrency and OCaml more efficiency. And all alternatives are more reliable than Python. In conclusion, all four languages are reasonable choices for the server herd.

1. Introduction

TensorFlow is a library that efficiently carries out computations, especially for machine learning algorithms. Most frequently, TensorFlow is used in conjunction with Python. The programmer can define the computation needs as a dataflow graph in Python. Then Distributed Master will split the graph into smaller tasks that are assigned to Workers. Each Worker will realize its task in a more efficient language, like C++, and send results back to Python environment.[1] TensorFlow significantly boosts performance when most heavy-duty computation can be carried out efficiently in C++, which is the bottleneck of the program.

However, if a great portion of computation is located in the Python environment, the Python code will become the bottleneck. For example, in an application server herd, queries from clients are handled by machine learning algorithms, implemented efficiently using TensorFlow. The server herd needs to handle a large number of small queries. As a result, the computation incurred by each query is fairly light and fast, while the large number of connections and transmissions across the herd are much more costly. In this case, the Python code is the performance bottleneck.

Since Python is not designed for highly efficient computation, in order to improve the server herd, it is reasonable to consider implementing the server herd in a different language that focuses more on performance. This also requires the programmer to bind those languages to TensorFlow. This paper analyzes the pros and cons, in this context, of three alternative languages, Java, OCaml and F#.

2. Java

Java is an object-oriented programming language. It is designed to be portable and Java byte-code runs on any machine that supports Java Virtual Machine. Java has a comprehensive set of mechanisms for parallelism. Also, it is strongly typed and adopts static type checking.

2.1. Feasibility of Java

It is possible to implement the aforementioned application server herd in Java for two reasons. First, the NIO2 API in Java provides a convenient framework for the server herd[2]. Anything Python asyncio achieves can be reasonably replicated in Java. Second, Java is compatible with TensorFlow. Java Native Interface can interact with C++ directly and thus build the computation graph in TensorFlow[3].

2.2. Advantages of Java

If the application server herd is implemented in Java instead of Python, parallelism can be more conveniently achieved. Due to the limitation of Python asyncio Library and Python Global Interpreter Lock, multithreading computation is hard to realize in Python. However, Java has a full set of mechanisms, like volatile and synchronized, to support multithreading. Once several threads works for one server at the same time, the performance of our server herd will be notably enhanced.

Since Java uses static type checking and is strongly typed, the bugs in Java code are more detectable than those in Python, which adopts duck typing at runtime.

Compared with Python code, Java code tends to be more portable across platforms. It is supported in Java

Virtual Machine environment, independent of the machine architecture.

2.3. Disadvantages of Java

Again, due to the difference in type checking of Java and Python, Java code is less flexible and harder to write. This may slow down the progress of development. Also, Python's type checking flexibility allows really concise code, which Java can hardly compete with.

3. OCaml

OCaml is a functional programming language, at time same time supporting object-oriented programming. For many architectures, OCaml can produce machine code that runs very efficiently.

3.1. Feasibility of OCaml

OCaml has Async library that organizes the execution of the code concurrently and asynchronously[4]. Async library used in conjunction with Tcp.Server class in OCaml can adequately substitute asyncio in Python. Besides, it is possible to have an OCaml binding for TensorFlow[5].

3.2. Advantages of OCaml

OCaml uses type inference at compile time, which removes the type checking overhead at runtime Python suffers from. Also, the type inference and static type checking allow more sophisticated optimization when OCaml code is turned into machine code.

Due to type inference, OCaml catches bugs in the program at the very early stage, whereas Python usually waits until runtime. Thus OCaml bring about more reliability,

What's more, the machine learning algorithms, which rely heavily on functions, are likely to benefit from the functional nature of OCaml. A lot of operations the algorithm requires OCaml can conveniently implement.

3.3. Disadvantages of OCaml

The static type checking and inference make OCaml less flexible than Python. The developer has to deal with a lot of type errors before the code can run, which can hinder the development progress. Furthermore, the strict type checking may reduce the flexibility of functions and thus inflate the amount of code.

Also, similar to Python in multithreading, OCaml uses a Global Interpreter Lock that does not allow simultaneous accesses to the same object. That is an obstacle OCaml creates for the application server herd.

4. F#

Similar to OCaml, F# is a functional language using type inference. F# has a really simple and concise syntax, as well as a set of mechanisms that support multithreading[6].

4.1. Feasibility of F#

F#, just like Python, has a library for asynchronous code execution, Async. Async provides implementations for various tools used by the application server herd like await, creating and canceling workflows[7]. Also F# can integrate with other libraries. .NET libraries provide us with a ready-to-user prototype of server and client. Furthermore, F# can interact with C++ as TensorFlow requires[9].

4.2. Advantages of F#

F# comes with various built-in tools for parallelism. Since in F# the data structures are immutable by default, parallelism can be conveniently implemented without data racing and deadlock[6]. Compared with Python, F# definitely has an advantage in performance.

F# uses static type checking and type inference just like OCaml. As discussed in 3.2, F# is more reliable and easier to debug.

Similar to OCaml, the functional nature of F# makes it more suitable to implement machine learning algorithms.

4.3. Disadvantages of F#

Python seems still to be easier to use than F#. Python duck typing avoids a lot of pain during the development. Despite the simplicity of F# syntax, it still uses static type checking and type inference, which greatly restricted the flexibility of F#.

5. Conclusion

All three alternatives, Java, OCaml and F#, are eligible candidates for implementing the application server herd involving TensorFlow. To summarize the pros and cons, compared with the three alternatives, Python is definitely advantageous in ease to use and flexibility. Nevertheless, Java and F# has better support for parallelism and multithreading; OCaml code is more optimized and efficient. Furthermore, due to static type checking, all alternatives are more reliable than Python. In conclusion, all four programming languages are adequate to implement the server herd and each of them has some distinct features.

References

[1] "TensorFlow Architecture"
https://www.tensorflow.org/extend/architecture

[2] "A Guide to NIO2 Asynchronous Socket Channel"
http://www.baeldung.com/java-nio2-async-socket-channel

[3] "How to invoke a trained TensorFlow model from Java programs"
https://medium.com/google-cloud/how-to-invoke-a-trained-tensorflow-model-from-java-programs-27ed5f4f502d

[4] *Real World OCaml*
https://realworldocaml.org/v1/en/html/concurrent-programming-with-async.html

[5]
https://github.com/LaurentMazare/tensorflow-ocaml/tree/master/src

[6] "Introduction to the 'Why use F# series"
https://fsharpforfunandprofit.com/posts/why-use-fsharp-intro/

[7] "Asynchronous programming"
https://fsharpforfunandprofit.com/posts/concurrency-async-and-parallel/

[8]
https://github.com/migueldeicaza/TensorFlowSharp