Language Bindings for a TensorFlow Server Herd

Zicong Mo, *University of California, Los Angeles*

Abstract

TensorFlow applications use a high level language like Python to set up a dataflow graph describing the computations, then perform the actual computations in a high performance language such as C/C++. In normal circumstances, the time spent setting up the model is insignificant compared to the time spent performing the computations. However, when the number of computations required is small, the program can spend too much time in the code initializing the model instead of the code performing computations. We consider three alternatives to Python as the high level language used to initialize the model in order to reduce the time spent not performing computations.

Introduction

Many TensorFlow applications spend most of their execution time performing computations inside C++ or CUDA code. When the size of the application is large, the amount of time spent in other languages initializing the model is insignificant. However, when a program must execute many tiny TensorFlow models, the overhead associated with creating the model becomes significant. The programming language used to create the dataflow graph is therefore significant when handling many small queries. In addition, the programming language must have good support for event-driven servers, as our TensorFlow models are being executed in response to incoming events and connections.

We consider the benefits and drawbacks of Python to other languages such as Java, OCaml, and Crystal in the context of designing servers that implement TensorFlow models, recognizing that the language must be able to quickly initialize TensorFlow models while maintaining solid support for event-driven servers. We focus on the language's ease of use, flexibility, generality, reliability, and most importantly, performance.

1. Types of Programs

Our application need to be able to support an event-driven server similar to the server herds implemented using asyncio, executing lots of small TensorFlow queries. Therefore, the language needs to be suitable for both the TensorFlow component of the application as well as the event-driven server component of the application.

1.1. TensorFlow

When running a TensorFlow program, a C API separates the user level code written in different languages into the code that is actually executed at runtime[1]. That is, the code written in Python or C++ describing the dataflow graph and operations is translated into C code before being executed on the hardware. This gives TensorFlow the ability to support multiple languages, while relying on the same central C API. A key step in executing any TensorFlow program is therefore the translation of the original code into C code. To increase the efficiency of small-scale applications, the language should quickly set up the model before performing the computations, as the overhead in creating the dataflow graph is more significant in small programs.

1.2. Event-Driven Server

Because the execution time of TensorFlow programs tend to be long, especially if the query is large, a key aspect we want in our language is the ability to asynchronously process new events and messages, and the ability to execute TensorFlow programs in separate threads. To consider why an asynchronous thread-based model may be helpful, imagine a large query comes in and enters the event loop. Since we are running only a single thread, no other requests will be handled while the large query comes in, reducing the throughput of the code. Although a multithreaded approach will be more memory and resource intensive and reliable multithreaded code is more difficult to write, the performance resulting from a multi-threaded event based server may justify the resources.

2. Languages Considered

2.1. Python

Python is an interpreted high level language emphasizing code readability. The language features a dynamic type system and automatic memory management, reducing the need to manually allocate memory or garbage collect. While Python is most commonly written in an imperative way, it also has support for functional style programming through functions such as *map* and *filter*. Python is the first client language supported by TensorFlow and continues to be the language that supports the most TensorFlow features.

A practical benefit of choosing Python over other languages is the easy learning curve both in creating a server and in TensorFlow applications. Because TensorFlow is originally written for Python, there is plenty of documentation and examples available for developers to use when creating a prototype. Furthermore, Python's duck typing makes it easy to write applications. When setting up a server or a TensorFlow model, Python's typing ensures that a working prototype can be created without needing to understand all of the types returned by function calls, and allows the developer to focus more on the concepts of the server or TensorFlow program. While this may be an advantage for initially creating a server, it may be a disadvantage for more senior developers who understand TensorFlow and Python well, as without the need to explicitly declare types, it can become more difficult to determine the type of a variable. In addition, since types need to be checked at runtime, Python programs will run slower than programs whose types are checked at compile time. Furthermore, this makes code less reliable, as a type error could occur during runtime that was not caught during testing.

Memory management in Python involves a private heap containing all Python objects and data structures[2]. Its automatic garbage collection also makes it easier to write programs, again shifting focus from details such as allocating and freeing memory to concepts such as program structure or algorithm efficiency. However, having a garbage collection system built into the language trades ease of use for performance. Having to keep track of reference counts and occasionally scan heap for unused objects is slower than a memory management model like C, which relies on the programmer to explicitly allocate and free memory.

Python is also an interpreted language, meaning the code is never compiled directly into an executable. This allows the program to be portable, and any system will be able to execute the program by interpreting the source code. However, interpreted languages tend to be slower than compiled languages, as the program is executing interpreted byte code instead of direct machine instructions specific to the system like an executable.

Python programs are also subject to the global interpreter lock, which is a mutex that prevents multiple threads from executing bytecode at the same time. Therefore, a Python based server is not able to handle as many requests as a server that runs a multithreaded model, which can create a new thread for each request that it receives, again decreasing the efficiency and performance of the program. However, there exist libraries such as asyncio and aiohttp that make it simple to create an asynchronous event-driven network with TCP and HTTP protocols.

2.2. Java

Java is a general purpose programming language designed to have as few implementation dependencies as possible. The language features a static type system and automatic garbage collection. Java is an object oriented language, that requires almost all of its code to be written inside an object.

Programs written in Java must explicitly declare the types of all of the variables in the program, which are checked by the compiler. Therefore, in order to create a working prototype in Java, the return types of all of the TensorFlow functions need to be declared, which can slow down productivity by forcing developers to learn the details of the library and ensuring that their functions all accept and return the correct types rather than learning the concepts of TensorFlow such as the dataflow graph. Additionally, since the types are checked by the compiler, it becomes more difficult to make a type error that is not caught until later, making code more reliable. Finally, since variable types are checked at compile time, they do not need to be checked at runtime, meaning that the program will run faster than a dynamically typed language like Python.

Java also features an automatic memory management system with a garbage collection system that automatically frees unused memory. Similar to Python, this leads to the advantage of accessibility at the cost of performance.

Java is unique in that it lies somewhere in the middle between an interpreted language and a compiled language. While the Java program is initially compiled to bytecode, the Java Virtual Machine is able to execute on any computer architecture, even compiling some bytecode sections directly to machine code using the JIT compiler. This aspect of Java makes the extremely portable, while still maintaining good performance. While the need to be interpreted means Java will not be as fast as a language such as C, Java provides a good middle ground between portability and performance.

Java also has good support for parallelism, meaning that a Java-based server would have more throughput than single-threaded servers such as Python. There also exist easy to use libraries such as Netty and NIO that allow an asynchronous event-driven network to be set up with TCP, HTTP, and TLS protocols.

2.3. OCaml

OCaml is a modern implementation of the ML family of languages with support for ML's functional style as well as the imperative and object-oriented styles. OCaml features a static type system, but automatically infers types. The language can be compiled directly into an executable, but also offers an interactive top level similar to an interpreter[3]. Although not explicitly supported by TensorFlow developers, there exist language bindings for OCaml to the TensorFlow C API.

The functional programming style is perhaps more difficult to learn and understand than imperative program, making it more difficult to prototype a working TensorFlow server. However, OCaml's static type system and type inference can help eliminate problems at runtime by catching them at compile time, making the programs more reliable. The obvious benefit of static type checking is that at runtime, the type and safety checks that dynamically typed languages perform are not necessary. Although it can be helpful, OCaml's unique type inference can also make it difficult to write code that conforms to the specification of the functions. Therefore, writing a working protocol is perhaps the most difficult in a language such as OCaml.

OCaml also implements its own garbage collector using the mark and sweep models and generational garbage collection. A language-implemented memory management model will be slower than a language that requires explicit allocation and freeing, but also makes it easier for developers to write programs.

OCaml is a compiled language, allowing the execution of optimized machine code directly on the machine. However, this trades performance for portability, as a new executable will have to be generated for each system architecture the program is intended to be run on. In addition, functional programming languages are in general more challenging to compile efficient machine code, due to issues such as the funarg problem[4]. OCaml features a foreign function interface that links to C primitives, including support for efficient numerical arrays, which is particularly useful for machine learning algorithms which deal with large matrices of numbers.

Similar to Python, OCaml programs are subject to the global interpreter lock, preventing true parallelism. Therefore, an asynchronous server implemented with OCaml will not have as much throughput as an asynchronous server that spawns new threads for each request, although it will be easier to write. There also exist numerous OCaml libraries to write asynchronous event-driven servers, such as the Async library.

2.4. Crystal

Crystal is a general-purpose object oriented programming language advertising itself as being "fast as C, slick as Ruby"[5]. The language is statically typed, with syntax similar to Ruby. Although not explicitly supported by TensorFlow developers, there exist language bindings. However, the language bindings are not frequently updated, with the last update occurring over a year ago. Therefore, TensorFlow code written in Crystal may not be the most maintainable or reliable, as bugs may not necessarily be patched quickly.

Similar to OCaml, although Crystal is a statically typed language, the types of variables do not explicitly need to be declared. This allows the easy learning curve of Python as well as the speed of statically typed languages. In addition, since type errors can be caught at compile time, programs will be more reliable.

Crystal is a compiled language, meaning it will execute more efficiently at the cost of portability. However, Crystal runs extremely fast, with Crystal web servers outperforming other frameworks written in languages such as Python, Rust, and Go[6].

Although Crystal does not support true parallelism, it supports fibers, which allow developers to easily write reliable programs using event loops and asynchronous I/O. It also supports asynchronous servers, and can automatically dispatch a new fiber to handle a message using the socket library[7]. Therefore, a server written in Crystal combines the event-driven model with a fast asynchronous multithreaded model.

Conclusion

Each language has advantages over the other languages in different areas, and the best language for an application therefore depends heavily on the specification of the application. Python and Java have the advantage of ease of use and portability, as they are interpreted languages with a large developer base. However, OCaml and Crystal have the performance advantage, as they are compiled languages with static type checking. In our program, our primary goal is performance, setting up the TensorFlow model as fast as possible in order to quickly start computations. In addition, the language must have good server support, as our application will asynchronously handle client connections and messages. Since our application is intended only to be run as a server, portability is not as important for our application. We therefore consider Crystal as a language to prototype our new application in, and we will perform reliability and performance tests on a Crystal prototype before rewriting the entire application.

References

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

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

[3] *FAQ*. Ocaml. Accessed Jun 06, 2018. Available: https://ocaml.org/learn/faq.html

[4] *OCaml*. Wikipedia, The Free Encyclopedia. Accessed Jun 06, 2018. Available: https://en.wikipedia.org/wiki/OCaml

[5] *The Crystal Programming Language*. Crystal. Accessed Jun 06, 2018. Available: https://crystal-lang.org

[6] Suzuki, Taichiro. *Which is the fastest?* GitHub respository. Accessed Jun 06, 2018. Available: https://github.com/tbrand/which_is_the_fastest

[7] Johnson, Sam. *Why Crystal is the most promising programming language of 2018*. Medium. Jan 23, 2018. Available: https://medium.com/@DuroSoft/why-crystal-is-the-most-promising-programming-language-of-2018-aad669d8344f