Language Bindings for TensorFlow

Abstract

TensorFlow is a machine learning framework written primarily in Python while the actual computations are performed in the lower level language of C/C++. The application under review handles many small queries, and now most of the program's time is spent on executing Python code which is initializing these machine learning models instead of actually performing the computations. We consider three other languages as plausible candidates for implementing our application: (1) Java, (2) Ocaml, and (3) Swift.

1 Introduction

The application under review is an application server proxy herd which utilizes a large set of virtual machines and heavily depends on the machine learning framework, TensorFlow, for its machine learning algorithms. These machine learning algorithms require a lot of computation, and are performed in C++ or CUDA code. The protoype of this application is written in Python and performs as well as expected on large queries. However, it often must handle small queries, which shifts the focus on performing computations faster to now improving upon the time it takes executing Python code to build the TensorFlow models.

Python is responsible for establishing a dataflow graph which defines data dependencies between a number of operations. Now, we turn our focus towards choosing the best programming language for establishing the call-graph and one that is suitable for the rest of the application. In this paper we summarize three alternative approaches using either Java, Ocaml, or Swift and compare against each and Python. For each approach, we greatly consider the tool's ease of use, flexibility, generality, performance, and reliability.

2 Python

Python uses dynamic type checking, meaning that it checks for correct types at runtime. This can lead to potential errors at runtime that would otherwise be caught by a statically typed language like Java. The main advantage of dynamic type checking "Duck Typing" is its ease of use for the developer, with a consequence of making the code more readable for the rest of the development team. Complicated computations are effortlessly written in its concise syntax. This is a major advatange in building a complicated server herd initially, but as the application grows larger, it will become more difficult to understand the data relationships between variables and functions.

Compared to the other languages in consideration, Python is the most common language for creating TensorFlow based applications. TensorFlow is primarily written in Python with a full set of documentation available for developers to help better understand the API, adding to its advantage of ease of use.

In terms of memory, Python's garbage collector uses a reference count method to decide whether objects should be deleted. These objects are stored with an additional data field specifying the number of times it is referenced in memory. It is incremented when the object is referenced by another object and is decremented when the other object does not reference it. If memory management is a concern for application under review, Python's memory management would be an excellent choice since objects are immediately erased as soon as they are no longer reachable. For a large asynchronous application which could use lots of memory, this language choice for handling the memory management would be a wise one.

Since the garbage collector continually runs in the

background, the ease of use for the developer improves as they no longer need to worry about memory allocation. In contrast to languages like C, which depends on the programmer to allocate and deallocate memory themselves, Python's garbage collector trades ease of use for performance. For C, these background checks are not needed thus improving the performance.

If multithreading is a topic of interest to improve performance, Python is not a particularly strong language for this. At Python's core, it has a global interpreter lock to synchronize threads so that only one thread can run at a time, even on multi-core processors. Thus, it does suffer in comparison to a multithreaded model which is able to handle many requests concurrently as an operating server. Python does its best to combat this lock by providing libraries like `asyncio` which allows for servers to accept and handle connections from clients asynchronously using coroutines.

In terms of flexibility, Python is a strong choice as it is an interpreted language, meaning that code does not need to be compiled into an executable to run. By avoiding the need of an executable which would otherwise need to be created for a variety of different systems/platforms, the program becomes extremely portable. As always, there's no such thing as a free lunch, and by avoiding the compilation stage, Python drops in performance since only interpreted byte code is being run instead of the machine instructions one would get from an executable.

3 Java

Java is a multi-paradigm, objected-oriented, imperative, and concurrent language. With the mentality of "write once, run anywhere"[1], java code can be compiled on any supporting platform without recompilation.

Java allows for great portability. Programs are compiled into Java bytecode, and using the Java Virtual Machine, this bytecode is able to run independent of platform or architecture. The Java compiler JIT can at times be so effective that some of the Java byte code can be transformed directly into machine instructions. This offers Java improved performance compared to a fully interpreted language like Python. However since Java byte code can be interpreted, it does not perform as well as languages like C/C++.

Java is statically typed, meaning the types of the variables and functions are checked at compile time. Once written, the code is then extremely reliable as each data relationship is clearly defined. However, this does make it more difficult to write code as it adds an additional responsibility on the developer. Especially if building the prototype in Java, the return types of TensorFlow functions must be researched and specified. Lastly, since types are checked at compile time, execution will run faster compared to a dynamically typed language.

In terms of memory management, Java also has a garbage collector automatically running in the background. In comparison to Python, this garbage collector will delete objects which have not been in use for an extended period of time. Due to the sole nature of having a background garbage collector which focuses on allocating and deallocating memory, it will perform worse against a language like C which forces the programmer to specify. Like Python, this does allow for ease of use.

In terms of multithreading potentials, Java can fully utilize multi-core processors to get more work done in the same amount of time. This can allow for Java based servers to handle and process more requests than a Python server.

4 OCaml

OCaml is another safe and strong programming language that supports functional, object-oriented, and imperative styles[2]. While it is typically not the first language to look when considering a language switch to use TensorFlow, OCaml does have the necessary language bindings to interact with Tensorflow. Here, we summarize a few key features of the language to consider before switching the prototype.

The language is unique in how types are automatically inferred by OCaml's type-inferring compiler. Similar to Python in that types do not have to be explicitly typed by the programmers, OCaml can allow for an ease of development in this respect. In theory, this a strong advantage but in practice, dealing with OCaml's auto-inferred types can be a major hassle for the programmers as more work might be down trying to correctly match the return type of the TensorFlow functions. Lastly, OCaml's static type system will have the advantage of catching errors at compile time making the application more reliable.

OCaml is a compiled language, meaning that the code source code is translated into machine instructions put into an executable. When running the executable, the machine code allows for the program to run that much faster than an interpreter trying to progress through the

program. With a major theme in computer science of ”there’s no such thing as a free lunch”, OCaml is able to gain performance benefits from being compiled, but it loses portability. Since executables are created, a different executable will be required for each software architecture the program will run on, adding more of a hassle.

In terms of memory management, OCaml also has a fast and incremental garbage collector that automatically runs in the background. Just like the previous languages, the fact of OCaml having a garbage collector in the first place leads to a sacrifice of performance for ease of development.

OCaml is a functional programming language. This means that functions themselves can be saved as any other variable and passed as potential arguments or returned by some other function. This functional style can be difficult for programmers who have only worked with the more popular imperative languages. Thus, a switch over to OCaml might be the most difficult, coding wise, in comparison to the other languages.

Like Python, OCaml cannot offer true parallelism, only concurrency. This is due to its global interpreter lock which only allows one process to run at a time. Also like Python, it does have external libraries to help build asynchronous applications such as the server herd we are mentioning.

5 Swift

Swift, the third possible replacement language, is a relatively new, reliable, and popular general-purpose programming language originally developed by Apple Inc in 2014. It is a compiled language, with a strong focus on safety and performance. All memory allocations are inserted by the compiler following a similar ARC[3] memory management technique as Objective-C. Swift is best known for its mobile development, but the language is being pushed by its strongly backed community for server and cloud based applications. Swift has recently announced in April of 2018[4] that it contains the necessary bindings to interact with TensorFlow allowing us to consider it as a candidate.

In terms of ease of use for the developer, Swift has a familiar imperative syntax as if it were an extension of the C family. With such a shallow learning curve, the language is very teachable and can allow for the development of a working prototype in a shorter amount of time. Though it is statically typed, Swift gains reliability, safety, and performance. To extend, Swift catches potential logic bugs at runtime, and since Swift is compiled, the translated machine instructions will run quicker than an interpreted language like Python.

Swift depends on LLVM and Clang, making it a significant dependency for TensorFlow[5]. In terms of portability, Swift has cross-platofrm capabilities without dependencies on the Objective-C runtime. If Python is potentially needed for data science prep before feeding a machine learning algorithm, Swift offers Python interoperability[6] allowing Python to be imported. This ensemble model could prove most efficient for the application under review.

Currently, Swift does not have language support for concurrency, which is a major weakness for our current application of constructing an asynchronous server herd. But, Swift does have a work queueing API called Dispatch (GCD) for performing concurrent operations. Swift has a difficult time dealing with asynchronous APIs but a new concurrency model which incorporates async is currently being worked on for asynchronous dispatches in GCD[7]. As these libraries are more continually worked on, a prototype in Swift should be strongly considered.

6 Conclusion

To reiterate for a final time, ”there’s no such thing as a free lunch”. Languages must make importance decisions in terms of ease of use, reliability, generality, and performance. Each decision comes with a tradeoff of its own. To review, OCaml and Swift have the advantage of performance as they are compiled languages with static type checking. In terms of ease of development, Python is the easiest language to develop in while Java and Swift follow shortly behind. While considering other factors like generality and portability is importance, ultimately what was important was choosing a language with strong performance to set up the TensorFlow model, and one which works well to asynchronously handle client requests. Looking ahead, Swift is the language of choice for this application. It is still in its infancy in terms of allowing for asynchronous programs but with a strong community, a working prototype should be able to be built quickly. If time is of the essence, Java is the second language we should consider for rewriting the prototype in.

References

[1] Nick Langley, *Write once, run anywhere?*
Available: https://www.computerweekly.com/feature/Write-once-run-anywhere

[2] *What is OCaml?*
Available: https://ocaml.org/learn/description.html

[3] *Automatic Reference Counting - Wikipedia*, 24 Aug. 2018,
Available: https://en.wikipedia.org/wiki/Automatic-Reference-Counting

[4] *Introducing Swift for TensorFlow - TensorFlow - Medium*, 26 April 2018,
Available: https://medium.com/tensorflow/introducing-swift-for-tensorflow-b75722c58df0

[5] *Tensorflow/Swift - GitHub*,
Available: https://github.com/tensorflow/swift/blob/master/docs/WhySwiftForTensorFlow.md

[6] *Python Interoperability - GitHub*
Available: https://github.com/tensorflow/swift/blob/master/docs/PythonInteroperability.md

[7] Duemunk, *Duemunk/Async - GitHub*, 21 Oct 2018
Available: https://github.com/duemunk/Async