Research into Language Bindings

Kevin Zhang

104939334

Abstract

TensorFlow applications reside in the realm of machine learning algorithms that often require large training sets. However, in certain unusual cases, when the number of computations required is small, the bottlenecks shifts from actually running expensive training algorithms to setting up models and their corresponding graphs. In our case, specifically, we handle a small number of queries and infer on small models. This causes the bottleneck to be in the alternative languages provided and not the C/C++ computation but instead on the programming languages used to set up the models in TensorFlow(Python, Java, OCaml, etc.).

1 Introduction

The problem of this research project is the unusual circumstances of an application server herd. Our specific project is different than normal applications of TensorFlow. Most machine learning applications spend the largest portion of their time performing computations to train the model and this large time commitment makes time that is spent setting up the models insignificant. However, in our application, we handle many short queries that group together to make the actual model creation a bottleneck. Therefore we must examine the languages used to set up these models. Specifically, we consider the benefits and drawbacks of four different languages: Python, Java, OCaml, and Kotlin. We focus mainly on these technologies' effects on ease of use, flexibility, generality, performance, reliability.

2 Languages

2.1 Python

Python, as analyzed in the previous report, is a powerful high-level scripting language, that features a dynamic typing system and fast/automatic memory management. Python is the first language that was supported by TensorFlow and is the most popular language to implement the front-end model construction. However, the Python language is designed to prioritize ease of development. Many of its production features support ease-of-development in exchange for certain performance overheads [2].

2.1.1 Advantages

As the oldest and most popular language that is compatible with TensorFlow, Python has the most extensive support. It has the most extensive documentation which makes developing under Python the quickest and most painless. TensorFlow's support for Python coupled with Python's relatively shallow learning curve makes it relatively easy to create a server and model for TensorFlow applications.

The Python language's typing is perfect for Tensorflow development. Python uses dynamic and duck typing. Therefore, setting up a server and creating a TensorFlow model in Python provides the same advantages as the advantages Python has over Java in implementing a server herd architecture. Developers do not need to figure out typing beforehand and can rely on dynamic typing to catch errors.

2.1.2 Disadvantages

The largest disadvantage of Python is its incompatibility with multithreading. Python programs are restricted by the GIL (Global Interpreter Lock). Because of this Python is restricted to quasi-multithreading and this restricts the speed of the model program. There exist packages like asyncio that was used in the project that provide workarounds to speed up performance. However, nothing beats the multithreaded performance offered by multi-threading compatible programs.

Moreover, Python's memory management scheme and construction as an interpreted language makes creating code in the language easier. However, ease-of-use is exchanged for performance as interpretation of code during runtime is slower than executing direct machine instructions. The overhead of keeping track of object reference counts and periodic linear time sweeps makes Python less efficient than other languages like Java [5].

Python typing is also another performance overhead that Python forfeits in exchange for ease of use. It is easier to set up code for TensorFlow because type clarification is not an issue. However, this results in necessary overhead during runtime to figure out the typing. Dynamic typing also means less reliability since type errors may cause runtime crashes that may cause unforeseen issues.

2.2 Java

Java is an extremely portable, object-oriented programming language that is very similar to C/C++. It provides very good concurrency mechanisms, static typing, and an extremely efficient automatic garbage collection system. Java has its own Java Virtual Machine (JVM) and all Java programs are compiled down to special Java byte code that is passed to the JVM to get interpreted.

2.2.1 Java Advantages

One main advantage over other languages is Java's support of concurrency and multi-threading. Because of this support, a Java-run server would have more overall throughput than single-threaded asynchronous servers. Most importantly, Java-run servers scale well and as the number of requests to a server increases, the overall throughput gains will also become more significant.

Java also features an efficient garbage collection system that frees memory using the mark-and-sweep method. This system is run on a different thread than usual program work so it doesn't affect performance as much as garbage collection in Python. It also performs certain optimizations such as reducing the number of times a generation segment is visited if that segment survives multiple sweeps. This garbage collection is holistic and prevents memory leaks as well as other memory related errors efficiently [5].

Java also requires static typing and compile-time type checking. Working prototypes in Java must be type-checked before it compiles which makes programs written in Java much more reliable. Although static type checking slows down productivity on the developer's end, it significantly speeds up runtime speeds since the overhead of type checking is removed.

Moreover, Java supports the Java Native Interface which allows inter-language calls to C and C++ APIs. Since C/C++ is implemented more efficiently, this also contributes to the efficiency of Java overall.

2.2.2 Java Disadvantages

Java's static typing makes it more difficult to produce quick prototypes as recommended by the homework specification. Similarly to Python, Java also affords certain performance overheads for ease-of-development. Java compiles into Java byte code which is then interpreted by the JVM. Although it is not necessarily a completely interpreted language since Java byte code is very close to machine code, there still exists overhead to interpret Java byte-code. This sacrifices performance to allow for ease-of -development and also portability. Java byte code also allows programs to be ported to any Java-supported device. Unfortunately, the cost is certain performance overheads.

Furthermore, Java's TensorFlow binding repository contains certain warnings against quick prototyping. Static typing constraints Java and makes it difficult to generate projects quickly. Since Tensorflow was originally designed with Python as the main language, certain incompatibilities arise when trying to bind it with Java. Java doesn't allow multiple inheritance, operator overloading, and certain other restrictions that can cause issues with binding.

2.3 OCaml

OCaml is a functional programming language that is one commonly used implementation of ML. It supports object-oriented and imperative mechanisms too which makes it easily scalable and suitable for large scale projects. OCaml uses a static typing paradigm and is compiled directly into executable code. There are language bindings that bind OCaml to the TensorFlow C API.

2.3.1 OCaml Advantages

OCaml's type inference mechanism is extremely useful in that it ensures runtime execution safety. It also allows for very powerful type polymorphism and currying which provides for easier programming.

OCaml's functional programming style is suitable for fast machine learning prototyping. Recursive execution is very suitable for searching for optimal solutions using neural networks which means that OCaml's core functionality is suitably designed for TensorFlow [6]. OCaml functional programming does not allow side effects and works best over immutable data structures. This means that it works well when optimizing over immutable matrices that are seen often in machine learning [6].

OCaml also has an efficient memory management system that boasts an efficient garbage collector. This gives programmers the flexibility to avoid manual memory management. The garbage collection OCaml implements is a hybrid of generational and reference counting garbage collection so it works better than both Java and Python's garbage collection system, sharing both language's advantages at the same time [5].

Furthermore, OCaml provides the necessary asynchronous server functionality that is needed to implement the server herd architecture.The asynchronous library Lwt is accessible in OCaml and provides the functionality required to implement an efficient server herd architecture. The promise entity in Lwt is similar to the coroutine entity in asyncio and Lwt_process/lwt_preemptive is similar to await [4]. Not

only does Lwt provide the functionality that Java and Python do, it also contains HTTP and TCP support [1].

2.3.2 OCaml Disadvantages

One of the biggest problems with OCaml is its readability and difficulty of use. The functional programming style relies on recursion; therefore, common imperative language tools that are taken for granted are not found in OCaml. For example, OCaml doesn't support overloading or error handling constructs which are very helpful in quick prototyping and fast development.

OCaml also doesn't support multithreading. It has a restriction that is similar to Python's Global Interpreter Lock and therefore suffers from some of the same issues as Python. OCaml does not offer true parallelism which decreases throughput overall. This presents an issue for both server management in the server herd and processing when creating the model in TensorFlow. Overall, not having multithreading available is a large performance hit when compared to other concurrency supporting languages like Java.

2.4 Kotlin

2.4.1 Kotlin Advantages

Kotlin is a statically-typed open-source programming language that can run on the Java Virtual Machine. Although it is not compatible with Java, it can interoperate with Java code. This means that Java and Kotlin can be executed by the JVM in conjunction with each other.

2.4.1 Kotlin Advantages

Kotlin/Native allows Kotlin code to be compiled into native binaries which allows it to be executed on a variety of applications including embedded applications, iOS, and WebAssembly. This versatility and flexibility is a huge advantage as it can be binded to both C and Java which can in turn, bind to TensorFlow. Although it takes more work to bind to C/Java, the C and Java bindings to TensorFlow are easy to use so it's not a huge hassle [3].

Kotlin also provides ease-of-development in many aspects. Kotlin provides seamless integration with the existing Java infrastructure and also provides its own efficient compiler. Kotlin also reduces the complexity of its code compared to Java, making code more concise and easier to maintain. This is due to Kotlin's smart casting, data classes, and powerful type inferencing. Data classes prevent the need for large class declarations in Java where multiple methods such as getters, setters, hash codes, and other class fields can be omitted. It also introduces Nullable and Non-Nullable which helps developers completely avoid NullPointerExceptions by declaring all variables as default non-null [3].

Kotlin also has great support for asynchronous execution. Kotlin uses their own version of coroutines which is similar to Python's coroutines in the asyncio library. Kotlin builds off of Java's concurrency mechanism and adds its own coroutine to make it much easier to implement and read [3].

2.4.2 Kotlin Disadvantages

Although Kotlin has a lot of benefits over Java, it has a large disadvantage in that its relatively new and has a very small support base. Not only is the documentation for Kotlin scarce, Kotlin also has less support overall. There is also very little incentive to switch to Kotlin, if Java already does the job well. Although Kotlin might have many ease-of-development features that developers can take advantage of, it is still hard to make that leap and learn Kotlin.

3 Conclusion

Overall, each language examined in this report has its own advantages and disadvantages. In our specific use case, Kotlin seems like the best choice overall in terms of performance standards. It provides the same asynchronous functionality and some of the ease-of-development features in Python while having many of the benefits of Java including efficient multithreading. The only problem with Kotlin is its relatively small user base. Although seasoned users of Kotlin are ideal for this project, Kotlin is used by relatively few, so developers must be trained in Kotlin before they can make reliable contributions. Fortunately, Kotlin is very similar to Java so the learning curve isn't insurmountable.

4 References

[1] "Asynchronous Execution OCaml." *Chapter 18. Concurrent Programming with Async / Real World OCaml*, v1.realworldocaml.org/v1/en/html/concurrent-programming-with-async.html.

[2] "Asyncio - Asynchronous I/O¶." *Asynchronous I/O and Concurrency Python Documentation*, docs.python.org/3/library/asyncio.html.

[3] "Comparison to Java Programming Language." *Kotlin*, kotlinlang.org/docs/reference/comparison-to-java.html.

[4] "Lwt." *Lwt Manual*, ocsigen.org/lwt/4.1.0/manual/manual.

[5] "Memory Management Reference." *2. Allocation Techniques - Memory Management Reference 4.0 Documentation*, www.memorymanagement.org/mmref/lang.html.

[6] Xu, Joyce. "Functional Programming for Deep Learning." *Towards Data Science*, Towards Data Science, 29 June 2017, towardsdatascience.com/functional-programming-for-deep-learning-bc7b80e347e9.