CS131 H6. Language bindings for TensorFlow Report

Abstract

Normally, typical TensorFlow applications are bottlenecked inside C++ or CUDA code; however, this application is unique in that it handles many small queries, that create and/or execute models that are tiny by ML standards. Previously, we implemented the application server proxy herd using Python, but due to the unique nature of the application's new workflow, most of the time is spent executing Python code to set up the models. Therefore, primary objective of this report is to analyze several programming languages in deciding which would be better suited for the implementation of an application server proxy herd that uses machine-learning algorithms and relies heavily on TensorFlow.

0. Introduction

The performance bottleneck correlates with the language chosen at the user-level to implement the client, which is used to set up the TensorFlow models. Thus, this report seeks to analyze the current language (i.e. Python) used to implement the application server proxy herd, in addition to three alternative languages: Java, OCaml, and Kotlin. The four languages will be assessed based on five main criteria: performance, reliability, flexibility, generality, and relationship to TensorFlow. Furthermore, these languages will be assessed on their suitability with regards to the scope of the application, which involves asynchronous event-driven servers.

1. Python Overview

Python is an interpreted, high-level general-purpose programming language that features a strong, dynamic type system and automatic memory management [1]. In addition, Python supports multiple programming paradigms, including object-oriented, imperative, functional, and procedural. Moreover, Python exhibits a comprehensive standard and open-source library, which supports asynchronous event-driven servers via the asynchronous networking library asyncio.

1.1 Python Advantages

In terms of generality, Python is incredibly easy to use as it features a very simple, easy to learn syntax that comes with an interactive shell for testing and debugging. In addition, code is highly readable, as Python code resembles pseudocode. For the aforementioned reasons, Python supports rapid prototyping speed.

With regards to flexibility, Python supports multiple programming paradigms to suit the given constraints of the application. Additionally, Python has an extensive standard and open-source library at its disposal (i.e. asyncio, aiohttp), and is also the original binding for TensorFlow. Moreover, Python is fairly portable as it is an interpreted language that translates source code to byte code, so any destination machine that has the Python interpreter installed can run the program (although it is not as portable as Java).

Lastly, with its relationship to TensorFlow, Python being a dynamically typed language makes it so that developers don't have to worry about all the various TensorFlow object types with regards to graph and ML model construction. And, the automatic memory management capabilities of Python make it so that the developer doesn't have to worry about allocating/deallocating memory for TensorFlow objects.

1.2 Python Disadvantages

The same features that make Python a highly popular language are also negatives in terms of performance. As Python is an interpreted dynamically typed language, code is interpreted at runtime (instead of compile time like the other three languages); and thus, type checking occurs at runtime, which hurts performance. In addition, the Global Interpreter Lock (GIL)—which is a mutex that protects access to Python objects in shared memory, preventing multiple threads from executing on the object at the same time—effectively inhibits the use of multithreading and prevents true parallelism from being achieved [2]. Lastly, the automatic memory management feature of Python—which employs reference counting and mark-and-swap garbage collection—also hurts performance as the program constantly involves TensorFlow objects quickly being spawned/destroyed, so these features reduce efficiency. Finally, with regards to reliability, the fact that Python is dynamically typed could lead to run-time type errors.

2. Java Overview

Java is a high-level, general-purpose programming language that is highly concurrent, highly portable, and features a strong, static type system [3]. Similar to Python, Java has a comprehensive standard library and supports the asynchronous requirement via NIO2, which is an asynchronous networking API that supports TCP connections and an event-driven style via futures and callbacks similar to asyncio.

2.1 Java Advantages

In terms of flexibility, Java is platform-independent due to Java code being compiled to byte code, which can run on any Java Virtual Machine (JVM) if it is present on the destination machine. In addition, Java supports extensive standard and open-source libraries with supported bindings with TensorFlow.

With regards to performance, Java being a strongly, statically typed language makes it so that type checking occurs at compile-time, which is more time-efficient than in Python's case. Moreover, instead of code being interpreted at runtime with Python, it is compiled at compile time, which is better for performance as well. Furthermore, Java has strong support for multithreading and is able to achieve true parallelism as opposed to Python. Lastly, the automatic garbage collection of Java is much more efficient than in Python as it doesn't involve the extra time-intensive tasks of reference counting and the employment of a cycle detector (a problem that can arise with reference counting). Also, users have to explicitly allocate dynamic memory, which makes garbage collection more efficient as it doesn't have to keep track of objects (where in Python, the interpreter must keep track of them). With regards to reliability, strong static typing makes it so that type checking occurs at compile time, so runtime type errors don't occur as in Python's case.

2.2 Java Disadvantages

In terms of generality, Java code can sometimes be overly verbose, less readable, and results in slower prototyping speed than Python. With regards to its relationship to TensorFlow, Java's static typing nature makes it so that developers need to be concerned with the types of the TensorFlow objects, which makes development more complicated and sluggish.

3. OCaml Overview

OCaml is a high-level functional language that features a strong, static type system, type inference, and automatic garbage collection [4]. In addition, OCaml supports multiple programming paradigms, including object-oriented, imperative, functional, and procedural. It also supports the asynchronous requirement via the lwt (or Jane Street's Async) libraries library, which handles both TCP and HTTP connections, and is similar to Python's asyncio library in that it runs in a single-thread.

3.1 OCaml Advantages

In terms of performance, OCaml's strong static typing nature does type checking at compile time (as does Java and Kotlin), but unlike Java, developers don't have to specify the type (which is similar to Python) as OCaml uses type inference. In addition, the automatic garbage collection of OCaml features a hybrid generational/incremental garbage collector that performs better than Python and Java, and also doesn't require as much up-front memory that Java's GC requires [5].

With regards to flexibility, OCaml has extensive standard and open-source library support (similar to the other languages) and supports bindings with TensorFlow. Additionally, OCaml has portability similar to Python (though not as portable as Java) as the compiler translates OCaml code to byte code. Lastly, in terms of reliability, strong static typing prevents runtime type errors from occurring as in Python's case.

3.2 OCaml Disadvantages

For most developers, functional programming is much more difficult to implement than imperative programming, and the type checking system of OCaml makes OCaml development considerably slower than Python, Java, and Kotlin in most circumstances. With regards to TensorFlow, OCaml (similar to Java and Kotlin) makes it more difficult on the developer when defining TensorFlow object types due to its type checking system, but this is more pronounced due to OCaml's type inference (similar to Kotlin). Lastly, in terms of performance, OCaml suffers from a similar problem as Python with regards to multithreading due to a global lock similar to Python's GIL that prevents multiple threads from running simultaneously; and thus, preventing true parallelism from being achieved [6].

4. Kotlin Overview

Kotlin is a statically typed, general-purpose programming language that was designed to be a better "Java", features type inference, and runs on the JVM [7]. Kotlin also supports multiple programming paradigms including object-oriented, functional, and procedural. With regards to the asynchronous requirement, Kotlin has an asynchronous networking library known as Korio that supports both TCP and HTTP connections, and is similar to Python's asyncio library with the use of an event loop and coroutines.

4.1 Kotlin Advantages

Kotlin benefits from similar performance to Java as it runs on the JVM, and employs the same static type checking, automatic garbage collection, and multithreading strengths. In terms of reliability, strong static typing prevents runtime type errors from occurring as they do in Python. However, as opposed to Java, Kotlin has added reliability with regards to null-safety as Java has the problem of NullPointerExceptions, whereas Kotlin ensures that no null references can occur. As far as flexibility is concerned, Kotlin shares the same high portability as Java. Lastly, Kotlin has Java beat in terms of generality, as Kotlin code is more readable and concise as opposed to the overly verbose Java syntax.

4.2 Kotlin Disadvantages

In terms of Kotlin's relationship to TensorFlow, Kotlin doesn't have official TensorFlow bindings (as it is a relatively young language compared to the other languages mentioned), but users can still use TensorFlow in Kotlin/Native. Lastly, Kotlin (similar to OCaml) can make it more difficult on developers when defining TensorFlow objects due to its static typing and type inference capabilities.

5. Conclusion

Overall, each of the languages outlined above feature favorable aspects that give a strong argument that they could be used for the implementation of this project. However, due to the unusual nature of the project, which features a large number of small queries and employs heavy use of TensorFlow, some of the languages are better suited for the project than others.

Python's dynamic typing and automatic memory management serve as a serious bottleneck due to the overhead associated with the Python code to set up the TensorFlow models. OCaml, while being the most efficient compared to the other languages, isn't the best fit for the asynchronous event-driven nature of the project due to it not being able to achieve true parallelism (similar to Python and its GIL). In addition, with defining the wide range of TensorFlow objects, the complexity associated with writing OCaml code and the type checking nature are also disadvantages. Finally, both Java and Kotlin seem to be good choices for the project as they both feature a wide range of advantages that coincide well with the constraints of the project (i.e. static typing, high portability, multithreading capabilities, asynchronous networking libraries, etc.). The only tradeoff is that Kotlin has added reliability and generality guarantees, but comes with the downside of being a newer language, which doesn't feature official TensorFlow bindings.

6. References

[1] "Python (Programming Language)." *Wikipedia*, Wikimedia Foundation, 7 Mar. 2019, en.wikipedia.org/wiki/OCaml.
[2] Wouters, Thomas. "GlobalInterpreterLock." *GlobalInterpreterLock – Python Wiki*, 2017, wiki.python.org/moin/GlobalInterpreterLock.
[3] "Java (Programming Language)." *Wikipedia*, Wikimedia Foundation, 14 Mar. 2019, en.wikipedia.org/wiki/Java_(programming_language)
[4] "OCaml." *Wikipedia*, Wikimedia Foundation, 7 Mar. 2019, en.wikipedia.org/wiki/OCaml.
[5] "Garbage Collection." *Ocaml,* ocaml.org/learn/tutorials/garbage_collection_html.
[6] "Concurrency, Parallelism, and Distributed Systems." *OCamlverse,* ocamlverse.githib.io/content/parallelism.html.
[7] "Kotlin (Programming Language)." *Wikipedia*, Wikimedia Foundation, 7 Mar. 2019, en.wikipedia.org/wiki/Kotlin_(programming_languag e).