CS 131 Homework 6 Report

Siddharth Joshi

Abstract

Tensor Flow is arguably the most effective open source platform for Machine Learning, one that provides a comprehensive set of tools, library and community services that allows developers to easily build and deploy Machine Learning powered applications. Tensor Flow is typically used with Python to define the computation as a data flow graph, after which the more involved computation is done lower level languages like C or C++ that provide better performance. However, this is traditionally the case due to the large query size that guarantees that the computation required for the queries is the bottleneck in the workflow and hence the performance of the language used to define the data flow graph is not of major concern. This report, thus, deals with the case where an application has several queries of small size and thus the overhead of setting up the computation is more computationally intensive than the actual execution. Therefore, this report aims to evaluate possible alternatives to Python for this task and in particular, the report covers 3 potential candidates: Java, Ocaml and Kotlin. The report discusses advantages and disadvantages to choosing each of the language and finally concludes that typically for such an application, Kotlin may be ideal looking purely at performance but community support for and familiarity with Java make a strong case for it as well.

1 Introduction

Machine Learning is a fundamental part of several applications in the industry today and while there are several tools available to developers and data scientists alike, Tensor Flow is arguably the most well known and widely used end to end open source tool available. It has a comprehensive, flexible ecosystem of tools, libraries and community resources hat lets researchers push the state-of-the-art in ML and developers easily build and deploy ML powered applications. As this demand for Machine Learning powered applications grows, the emphasis on ensuring high performance for these applications grows as well. The most conventional approach for implementing such applications is to use Python as the higher level language to define the comptuation as a data flow graph, which is followed by the actual computation occuring a more efficient lower level language such as C or C++. The reason for choosing languages with better runtime performance for the actual query is due to the fact that query sizes are usually extremely large and therefore the bottleneck is usually this stage. However, the application being evaluated in this report is one where a large number of the query are of a size that is miniscule in comparison to what Machine Learning applicatoins typically receive, and thus the bottleneck is observed to be the Python code that sets up the models and the computation, therefore this report attempts to find a suitable alternative that addresses this problem and helps address this problem. The alternatives considered in this report are Java, Ocaml and Kotlin. The following sections evaluate the advantages and disadvantages of each of the aforementioned candidates. The report then concludes with a final recommendation for which language would serve as the ideal alternative for Python in the context discussed.

In addition to these constraints, the application being considered must also support event-driven servers to allow for asynchronous network communication. Thus the languages chosen as potential candidates also provide reasonable support for this.

2 Java

Advantages

Java is an Object Oriented Language and this may well be considered as a pro as it allows developers creating such an application to write modular code that is easily reusable. Moreover, Java's extensive support for inheritance and interfaces allows developers to create arbitrarily complex interdependencies between these modules. In this regard, Java provides everything that Python does and much more, thus is definitely superior.

Another positive feature that it shares with Python is the Platform Independence resulting from every application executing in the Java Virtual Machine (JVM). This allows code to move easily across machines without recompilation etc. The reason being that like Python, Java applications are compiled into byte code (a higher level intermediary machine code for the Java Virtual Machine) and this is then executed by the JVM. Moreover, this process is highly efficient with the emergence of JIT compilers, that allow frequently used portions of the code to be compiled to the actual machine code, thus allowing for better performance. In particular, the JVM and the approach that Java takes to run its code in this platform independent way can be considered to be not just equivalent to Python but superior as it provides significantly better runtime than Python's interpreter.

Another extremely useful feature that Java shares with Python is extensive support for exception handling. Due to the communicative nature of this application, errors can result rather frequently during communication with clients or other servers, and exception hanlding provides a graceful way of recovering from these faults, while allowing the developer to be aware of the exact issue that occured as well.

The static type checking offered by Java can be considered as an advantage over Python's dynamic checking due to the reliability it offers. By ensuring that code that compiles will not run into type errors, an application written in Java is considerably more reliable which can be a major asset especially when there is considerable communication with clients.

Java's garbage collection is a convenience that it shares with Python, however, the approach taken by Java is by far the superior in terms of runtime which is the primary concern in the application being evaluated. Java uses a mark and sweep algorithm which due to its approach of periodic checks to release unreachable blocks may be less efficient in terms of space, beats Python's reference counting approach with regards to runtime by miles. The reference counting approach introduces a significant overhead for both assignments and end of scope (situations where references are deleted/lost).

Additionally, an important consideration when considering the runtime performance of Java in comparison to Python is the extensive support for multithreading that Java provides, whereas in Python multithreading is rather ineffective that is unable to achieve any real parallelism due to the Global Interpreter Lock that prevents any concurrent execution. In fact, Python's multithreading often results in worse performance than sequential applications and thus Java's support for multithreading is a significant advantage especially since the primary concern is runtime.

Furthermore, the requirement of the application for asynchronous network communication and event-driven servers. The support for asynchronous programming allows for the ability to design applications that are able to asynchronously handle I/O operations and be event-driven. In the recent Java8, the CompletableFuture interface introduced provides support for attaching callbacks and constructing something akin to continuations from Scheme. These features make designing such an asynchronous networking application relatively straight-forward in Java.

Disadvantages

Perhaps the only real disadvantage to using Java rather than Python, is a controversial feature that can also be considered an advantage (as it is here in this report): the static typing of variables. The added convenience and simplicity in application that relies on many moving parts such as this provided by dynamic typing allows for rapid prototyping and implementation.

Finally, another disadvantage is the fact that TensorFlow Java API does not come under the TensorFlow API stability guarantees and thus may be changed in the future, in backwards incompatible way, which may potentially cause issues for our application.

3 Ocaml

Advantages

Ocaml is a functional programming language, which makes it a natural fit for a machine learning applications due to the heavy reliance of functions (due to the presence of significant mathematical expressoin evaluations) in Machine Learning. In addition to this the functional approach prevents the existence of any side-effects and thus eliminates problems of transparency i.e a variable's value is evident at all points, and also makes debugging far simpler as functions can be treated as black boxes that take in a input and provide an output. Finally, the absence of side-effects increases reliability for the aforementioned reason as well.

Another advantage that Ocaml offers is the strong static type checking system that establishes type by inference. This is the perfect resolution of the conflicts of static and dynamic type checking in some ways as it provides the simplicity of not explicity having to state types continuously but also provides the reliability of conventional static type checking. The type checking system of Ocaml is a major asset as it also provides significant help in debugging - incorrect (unexpected) types point towards incorrect implementation.

Ocaml, like Java, uses a mark and sweep algorithm along with generational garbage collection for garbage collection

which provides thus the same performance benefits over Python as Java does.

The *async* library provides support for developing event-driven servers as desired by the application being considered. This library provides developers with a clean high-level API to handle tasks such as starting servers in a manner as convenient as Python's *asyncio* networking library.

Disadvantages
A disadvantage of using Ocaml is that it may not completely resolve the concerns regarding runtime performance that were the primary issue with Python, due to the fact that like Python, Ocaml too uses a global interpreter lock which thus prevents any parallelism from being achieved, thus rendering multithreaded no faster or even slower than single threaded code.

There is no offical support for Ocaml by TensorFlow, therefore support is even more limited than the mere potential for backward incompatiblity of Java. While there are definitely authoritative bindings that have been established for Ocaml, the support is no way close to as comprehensive as it is for Python or even Java.

4 Kotlin

Advantages
Kotlin is considered to be a successor of Java in many ways and as a result inherits most of the positives that Java possessed. Notable instances of this are the platform independence and ease of portability offered by the use of byte code alongside virtual machine rather than directly compiling to machine code. Moreover, the vast abilities that Java offers to developers with reference to multi-threading are supported by Kotlin as well, thus allowing for significant performance gains (thus addressing the primary concern of this application).

Additionally, also deals with some of the flaws of Java. In particular, it supports functional programming that allows it to tap into the merits that Ocaml offers if a functional approach is adopted. Moreover, Kotlin deals with the issue of null pointer exceptions making it more reliable even than Java. Furthermore, Kotlin like Ocaml, provides type inference while maintaining static type checking and thus similarly merges the benefits of static and dynamic type checking much in the same way Ocaml did.

Disadvantages
Like Java and Ocaml, being statically typed can be considered a pitfall of using Kotlin since it doesn't allow for the same rapid prototyping that Python does.

Kotlin is a new langauge and thus in general doesn't have nearly as many resources for reference available as Java or Ocaml. In particular, there doesn't seem to be any one authoritative set of bindings for TensorFlow and thus developers do not have access to a whole plethora of functionality that may be the standard for Python or Java.

5 Conclusion

While from a purely runtime performance and reliability standpoint, Kotlin clearly outperforms both Java and Ocaml, the lack of significant support for TensorFlow in Kotlin can be a significant deterrent for developers and pose as a real hurdle. Thus, Java which has nearly equivalent performance (especially with reference to runtime - which was the primary concern), could potentially be a more suitable candidate. To conclude, both Java and Kotlin are rather promising alternatives and either would serve the purposes of this application sufficiently, thus the final choice depends on the developer's preferences.

References

[1] "TensorFlow." TensorFlow, www.tensorflow.org/.

[2] "Asynchronous Programming Techniques." Kotlin, kotlinlang.org/docs/tutorials/coroutines/async-programming.html.

[3] "Benefits of Java over Other Programming Languages." Invensis Technologies, 6 Apr. 2018, www.invensis.net/blog/it/benefits-of-java-over-other-programming-languages/.

[4] Artem Golubin. "Garbage Collection in Python: Things You Need to Know." *Artem Golubin*, Artem Golubin, 7 Jan. 2019, rushter.com/blog/python-garbage-collector/.

[5] "Neural Networks, Types, and Functional Programming." Neural Networks, Types, and Functional Programming – Colah's Blog, colah.github.io/posts/2015-09-NN-Types-FP/.