Language Bindings for Tensorflow

Tanzeela Khan
University of California, Los Angeles

Abstract

The report reviews TensorFlow architecture and considers several languages for our application. The application we have uses machine learning algorithms and is built on a Python Prototype. Tensorflow is built for Python, but our goal is to evaluate whether we can build a more efficient application using TensorFlow, but in a different language.

1 Introduction

TensorFlow is heavily used in our Python application, and we are considering and evaluating more efficient languages to implement our application. We must explore the binding ability of TensorFlow for Java, OCaml, and F#. To fully assess each of the languages, we also will take a look at asynchronous event handling, and language-specific characteristics.

2 Java

There are several APIs and frameworks that support server event-driven herding. Tensorflow currently supports binding in Java [1], but Tensorflow does not guarantee stability. For Java, graphs are constructed using python and must be translated to create Tensorflow bindings for Java.

2.1 Support of Event-Drive Servers

For asynchronous blocking, Java has Netty and Play Framework. Netty is a high performance event-drive framework that has support for HTTP[2], directly applicable to the server-herding. Play Framework is a modern web framework based on Netty and inspired by Ruby on Rails[3]. It has a very short development Cycle, good for development.

Another option ngnix. This framework can be used as a proxy in front of a Java container. ngnix can be used a reverse proxy/load balancer to achieve scalability [4].

One more option is a new non-blocking I/O API called NIO. It has been included in the J2SE platform, and helps with the development of event-driven Java web servers[5].

2.2 Ease of Use and Flexibility

To work with asynchronous blocking frameworks in Java, the programmer must run all long-running processing in the background thread. Otherwise, the server will be blocked while running in the main thread.

2.3 Performance and Reliability

Frameworks like Netty have the issue that the io on the sockets is event driven instead of blocking. The ngnix framework address this but more library support is needed to address this problem in Java overall.

Research shows that NIO-base server can outperform commercial native-compiled web server. It simultaneously reduces the complexity of code and system resources required to maintain it.

2.4 Technical Challenges

Java has the technical challenge that programmers must run all long-running processes in the back-

ground thread. To work with asynchronous blocking frameworks in Java, the programmer must run all long-running processing in the background thread. Otherwise, the server will be blocked while running in the main thread. To deal with this issue, NIO is a possible alternative.

2.5 Evaluation

Java's bindings to TensorFlow are written by the developers of TensorFlow, which is a huge boon. Java is definitely usable for aysnchronous event-driven programming. In both of these matters, Java is faster than python and would be good option to replace the python backend.

3 OCaml

OCaml is a single-threaded event-driven system, which executes a single task at a time. Most of the support for this type of system comes from Ocaml libraries. OCaml currently has some bindings supported by TensorFlow [6]. OCaml is currently in experimental mode for TensorFlow.

3.1 Support of Event-Drive Servers

OCaml has an Async library that supports current programming, or asynchronous blocking. Functions never block, instead they return a D̂effered.t type [7]. This is a value that will eventually be replaced with a result.

Another library is libevent. This is an ocaml wrapper for the libevent API, which provides a mechanism to execute a callback function when an even occurs [8]. It replaces the event loop found in most event driven network servers, such as Python's asyncio.

Ocaml-cohttp is a third option. This OCaml library is a portable HTTP processor for creating HTTP daemons. Its implementations combine Lwt, an asynchronous library for UNIX bindings, and Async [9].

3.2 Ease of Use and Flexibility

Async is relatively easily to install and cited for reliably dispatching requests.

There is less information about the flexibility Mirage and OCaml-Libevent. These are open-source options available on Github.

3.3 Performance and Reliability

OCaml offers compilers. This gives the language high performance, and advantage over Python which uses an interpretor.
The async library avoids performance compromises and preemptive threads, making it a great boon to asynchronous task handling. Dispatching requests over the message queue with multiple processes and servers has been reportedly fast[10].

3.4 Technical Challenges

In regards to Tensorflow, the fact that OCaml is in experimental mode implies it would be technically challenging in some uses cases when reverting from Python to OCaml.

Multi-threading is not possible in OCaml currently. While this is hte case, it is not necessary because libraries such as Lwt and Async currently exist.

Ocaml-Libevent's has issues involving memory leaks and API inconsistencies between lwt and async. More collaboration and respondence to these issues would contribute to a better source for Ocaml Event handling.

3.5 Evaluation

Using multiple processes on Ocaml is unlikely to work out well. OCaml does not have all the bindings of Tensorflow and is currently in experimental mode. There is not a lot of web-driven support for the language, making it uncompetitive for writing networking servers. Especially since most of the support comes from only Async and source from unofficial

github repos, it is questionable how feasible event-driven server herding on Ocaml is. As result, it unlikely a good option against Python.

4 F#

F# is a functional programming language often used for scientific computing. Tensorflow currently supports .NET bindings for F# [11]. For the most part, bindings have been completed by open-source users. The developers of Tensorflow have not released an official binding of their own.

4.1 Support of Event-Drive Servers

F#'s asynchronous programming model allows for programmers to write code that does not block the executing thread, but can still be written in sequential style[12]. In essence, F# has a event-based programming model.

One option to make HTTP requests using the Service library[13].

The EventStore API is also available. This API allows users to make connections to a stream. EventStore is an implementation of a state machine using GraphDB, SQL, Client, and ElasticSearch.

4.2 Ease of Use and Flexibility

F# is relatively flexible to handle. One programmer documents his experiences translating F# to Javascript. Since this is true, and python and Javascript are both sticky languages, it is likely not difficult to translate Python server herd implementation to F#.

There is also ease of use in creating a web server with F#. Programmers an use HTTPListen library, self-hosted WCF service, and NancyFx framework[14]. Unfortunately, NancyFx does not have natural support for F# and the code is not simple to understand or implement.

4.3 Performance and Reliability

The ûpdateLoop function in F# makes it easy to initiate new requests quickly. The asynchronous workflow of the language forces the body of this function to wait 0.5 seconds and asynchronously call and update server.

4.4 Technical Challenges

This is a relatively new language that is still garnering support. F# does not have the official support of TensorFlow, only the bindings developed from outside users.

4.5 Evaluation

Since the TensorFlow bindings are no longer in experimental mode and the API appears to have stability, F# would be a good choice to replace the Python TensorFlow. Its flexibility and speed adds to this argument.

5 Conclusion

We have reviewed several potential languages to replace TensorFlow's Python backend to increase efficiency. Java has a lot of support for asynchronous handling, and currently is supported by TensorFlow bindings, but has some instability. While Java is slow to start, it is much faster than Python with increased computing time. When compared to only Python, I would recommend Java as a suitable option. OCaml is much faster than python and has most of the support of TensorFlow bindings, but less so and it is still under experimental mode. For this reason and more research supporting OCaml's limitations with asynchronous servers, I would not recommend OCaml. F# on the otherhand has good support and almost complete binding from Tensorflow. The language has shown to be flexible among different asynchronous server handling calls.

Notes

[1]GitHub. (2018). tensorflow/tensorflow. [online] Available at: https://github.com/tensorflow/tensorflow/tree/master/tensorflow/java.

[2]Netty.io. (2018). Netty: Home. [online] Available at: http://netty.io/.

[3]Playframework.com. (2018). Play Framework - Build Modern & Scalable Web Apps with Java and Scala. [online] Available at: https://www.playframework.com/.

[4]NGINX. (2018). NGINX — High Performance Load Balancer, Web Server, & Reverse Proxy. [online] Available at: https://www.nginx.com/.

[5]Beltran, V. (2018). Evaluating the Scalability of Java Event-Driven Web Servers. Barcelona (Spain): Computer Architecture Department, Technical University of Catalonia (UPC).

[6]GitHub. (2018). LaurentMazare/tensorflow-ocaml. [online] Available at: https://github.com/LaurentMazare/tensorflow-ocaml/.

[7]Realworldocaml.org. (2018). Chapter 18. Concurrent Programming with Async / Real World OCaml. [online] Available at: https://realworldocaml.org/v1/en/html/concurrent-programming-with-async.html.

[8]GitHub. (2018). ygrek/ocaml-libevent. [online] Available at: https://github.com/ygrek/ocaml-libevent.

[9]GitHub. (2018). mirage/ocaml-cohttp. [online] Available at: https://github.com/mirage/ocaml-cohttp.

[10]https://discuss.ocaml.org/t/7-years-after-is-ocaml-suitable-to-write-networking-servers/639

[11]GitHub. (2018). migueldeicaza/TensorFlowSharp. [online] Available at: https://github.com/migueldeicaza/TensorFlowSharp.

[12]Petricek, T. (2018). Asynchronous client/server in F# (QCon 2012). [online] Tomasp.net. Available at: http://tomasp.net/blog/qcon-async-fsharp.aspx/.

[13]http://gorodinski.com/presentations/event-driven-services-with-fsharp-and-eventstore/#/14

[14]Tihon, V. (2018). Three easy ways to create simple Web Server with F#. [online] Sergey Tihon's Blog. Available at: https://sergeytihon.com/2013/05/18/three-easy-ways-to-create-simple-web-server-with-f/.