CS 131 Homework 6:
Evaluating Dart for a Garage-Sale Buyer App

Abstract

We evaluate the use of the Flutter user interface toolkit, written in Dart, for a mobile application intended for people who buy things from garage sales. One of the features that this application has is to report the best deals in a garage sale based on a panoramic image of the items. We evaluate the features of Dart for the proposed app, comparing and contrasting them to the features of Ocaml, Java, and Python and finally giving a recommendation about its use.

1 Introduction

Our application, GarageGarner, previously had a web-based implementation, with most of the computation being done in a central server with only the user interface running locally on users' phones. In order to provide users with the best possible deal based on their panorama of the garage sale, the images are sent to the server, where all computations are performed. However, this uses too much bandwidth and has performance issues.

In order to solve these problems, we attempt to have the required machine learning algorithm run on the users' devices. Thankfully, newer phones have accelerators in the form of GPUs to make this possible from a hardware standpoint, and the TensorFlow Lite platform allows us to run our model. One proposal was to use the Flutter U/I Toolkit, written in Dart, with its $tflite$ plugin, to create the new application.

2 About Dart

Created by Google, Dart is a "client optimized language for fast apps on any platform."

Allegedly, Google engineers created Dart due to frustration with maintaining "massive JavaScript code bases" for Gmail and Google Maps. Hailed initially by some blogs as a "JavaScript killer," Dart received a fair amount of marketing, but didn't end up gaining as much traction as anticipated. Google's original plan to replace JavaScript with Dart in its browsers and in web application development were wholly unsuccessful. However, usage trends increased after Google released Flutter, a UI framework for iOS and Android, which made Dart a viable framework for mobile application development [2].

3 Advantages

3.1 Developer Friendliness

Most code editors and IDEs support Dart's configurable set of tools including profilers, loggers, debuggers, and a static analysis system, which has been called a powerful way to prevent errors in code. [1] In addition to Dart's flexible type system, these features save time for developers and cater to their preferences [1].

However, the largest selling point of Dart (and Flutter, for that matter) to developers is the **stateful hot reload** capability that it offers. A hot reload refers to the capability to immediately see code changes reflected in the prototype. From the Flutter documentation [3]:

> Flutter's hot reload feature helps you quickly and easily experiment, build UIs, add features, and fix bugs. Hot reload works by injecting updated source code files into the running Dart Virtual Machine (VM). After the VM updates classes with the new versions of fields and functions, the Flutter framework automatically rebuilds the widget tree, allowing you to quickly view the effects of your changes.

Thus, using this capability developers can quickly experiment with different ideas and see their potential effects without losing their current state. This conceivably saves a lot of developer time in the long run.

3.2 Flutter User Interfaces

The ease of creating user interfaces in Flutter is also a notable advantage of using Dart. Flutter has cross-platform flexibility,

meaning that a single codebase can be used for both Android and iOS versions of an application. Flutter employs a widget-based architecture, where widgets classes used to create user interfaces. Almost everything, including UI elements and layout structures, is a widget. This style is a departure from traditional native mobile development, as developers interact mostly with the widget interface provided by Dart, rather than those provided by Android or iOS [2] [4].

One advantage to widgets is that they enable high frame rates of up to 120 fps, given the hardware is compatible [4]. This is because widget objects are immutable, with their dynamic elements generally being stored in separate State objects. When a widgets needs o be changed, it is "rebuilt" into an entirely new instance. Ensuring the immutability of widget objects helps with performance, including frame rate in this case.

The nature widget architecture allow developers to create customizable layouts with flexible designs without having to use a separate API to design UI elements [1] [2]. One of the most appealing features is the number of aesthetic widget "catalogs," with the "Cupertino" catalog, which mimics iOS native design features, being one of the most popular. Flutter was created to make the lives of front-end developers easier, and it seems that, for many use cases, it delivers on that promise.

3.3 Compile Options

Dart provides a number of compile options. In terms of timing, we can use AOT (ahead-of-time) or JIT (just-in-time) compilation. JIT compilation capability is extremely useful in saving time during development; it's what enables the hot reload capability. Conversely, AOT is important for production-level builds of the application to provide quick start-ups and fast execution [2].

Additionally, while Dart normally compiles to 32-bit or 64-bit ARM machine code for mobile devices, it can also compile to x64 for desktop platforms or JavaScript for the web [1]. This means that code needs to be written in Dart only once to reach all platforms, although it's unlikely that anyone would really use Dart for a desktop application. More practically, the "compile to JavaScript" option makes web development in Dart more plausible, since almost all popular browsers today run on JavaScript.

3.4 Performance

The direct compilation to ARM compilation is beneficial for performance, as there's no compilation "bridge" required [2].

3.4.1 Asynchronous Programming

Dart can be also be used on the backend of applications to write HTTP clients and servers. Language features are important here, as server-side operations, especially I/O, can provide a performance bottleneck. We explored this concept in our previous investigation of asynchronous programming in Python with *asyncio*. Thankfully, Dart provides this exact same capability with event loops and the same async/await keywords.

3.4.2 Benchmark vs Node.js

We also found that Node.js is usually more performant in this case than Python. Upon further research, Debian.net performs benchmark testing with a wide variety of languages, and found that Dart as about as performant as, if not more, than Node [5].

Overall, it seems that performance is one of Dart's better features. For GarageGarner, this may not be so important because the image-classification algorithm already presents a bottleneck. However, Dart's asynchronous concurrency may be very useful in this case.

4 Disadvantages

4.1 Lack of Developers

Due to its recent introduction, Dart lacks a large developer community. In fact, Dart has only recently been gaining users, mostly for mobile development with Flutter. The lack of developers poses a number of problems.

First, it may be harder to find Dart developers, than, for example, JavaScript developers. Also, during development, it's much easier to find external help and resources in more established languages rather than Dart. Although Dart's documentation is quite good, trying to find help for a specific problem would be hard.

4.2 Fewer Libraries

Another issue that comes with lack of adoption is the lack of external libraries and plugins that can be used. While Google has provided many packages and features, third party and open source libraries are always crucial. For Dart, these libraries will be scarce, and it may be hard to establish some specific tasks. Conversely, in Python or JavaScript, there are many more third-party libraries that directly address features developers wish existed.

Thankfully, for GarageGarner, our main third party desired feature is available in the form of the *tflite* plugin.

4.3 Integration of Native Features

One issue in Dart that's not necessarily related to developer adoption is the fact that Flutter developers may be missing out on access to certain native features in mobile applications. As mentioned earlier, Flutter developers can only access native

capabilities through widgets. Inevitably, this doesn't provide the same amount of access as other developers; there may be some lesser known features that aren't accessible through Dart widgets. This will also probably lead to later access for Dart developers when new features are implemented on iOS or Android.

For GarageGarner, this is probably not a huge issue, as we mostly just need access to cameras, which is already supported.

5 Dart vs Other Common Languages

Dart is a multi-paradigm language with a C-like syntax. It supports object-oriented, functional, and imperative programming paradigms. It is statically typed and supports type inference, making type declarations optional. However, there is also support for *dynamic* variables whose types are inferred at runtime. Dart's garbage collection

5.1 Ocaml

Compared to Ocaml, Dart has a similar typing system (static typing with type inference) and support for functional programming elements (e.g. lambda functions, higher order functions). The main difference is that while Ocaml is a pure functional language, Dart is a more general purpose language which is much easier to use for many developers across various platforms.

5.2 Java

Due to Dart's C-like syntax, Java and Dart are a little more similar; a Java developer would have an easier time adapting to Dart. The object-oriented concepts and syntax, like class declarations, are very similar. However, a few differences are that Dart supports optional parameters instead of overloading constructors, and that code can live outside of classes in Dart.

The JVM mostly supports JIT compilation, while Dart provides both JIT and AOT options (however, an experimental AOT option was introduced in Java 9). While Java compiles to JVM byte code which is interpreted, Dart compiles directly to JavaScript or machine code. Thus, in terms of native application development, Java is largely used for Android, while Dart provides more flexibility.

One important feature of Java is the JVM's support for multithreading, which we investigated previously. Meanwhile, Dart code runs in isolates. Each isolate has its own event loop and memory space, and isolates can communicate by messages. We can use isolates like threads, but really, they are more analogous to processes. Dart's single-threaded model does not allow shared memory or preemption.

5.3 Python

Out of all, Dart is probably most similar to Python. Both are newer languages with support for dynamic typing and OOP and without required type declarators or support for traditional multithreading. As discussed above, Dart's isolates provide an elegant solution to this problem.

Finally, Dart supports asynchronous programming with *futures* and event loops, a very similar approach to Python's *asyncio*. Some differences are that Dart is generally statically typed, and Flutter makes UI development in Dart much easier than Python. Garbage collection is also implemented differently in Dart with a generational mark-and-sweep algorithm similar to Java. This is made possible due to isolates, allowing quick construction and destruction of widgets, which, as discussed earlier, enables high frame rates.

6 tflite

Using TensorFlow provides us access to a number of pretrained models and runs on heavily optimized C++ and CUDA code. It is extremely difficult to create reliable, performant machine learning models from scratch, so TensorFlow is definitely preferable for our image processing algorithm. The *tflite* plugin for Flutter gives us access to TensorFlow functionality, which will run using the phone accelerators (GPUs).

7 Conclusion

Compared to Ocaml, Python, and Java, **Dart is the most suitable for GarageGarner**. Flutter provides the ability to easily make aesthetically pleasing, cross-platform user interfaces. The hot reload and compile options speed up development, and the support for asynchronous programming combined with *tflite* will allow us to deliver a high-performing application that performs image-classification natively on user devices.

The other languages mentioned aren't as well-suited to mobile development; using them for GarageGarner would likely necessitate multiple different languages and frameworks. Perhaps another suitable option for GarageGarner would be JavaScript using React Native and Node.js. This would provide the ability to make the entire application in one language, and TensorFlow support exists as well. JavaScript came up briefly in earlier discussions either would work well. Overall, Dart certainly seems like a strong choice for GarageGarner.

References

[1] Dart. https://dart.dev/

[2] The Fall and Rise of Dart, Google's "JavaScript Killer". https://insights.dice.com/2019/03/27/fall-rise-dart-google-javascript-killer/

[3] Flutter Hot Reload. https://flutter.dev/docs/development/tools/hot-reload
[4] Flutter Layouts. https://flutter.dev/docs/development/ui/layout
[5] https://benchmarksgame-team.pages.debian.net/benchmarksgame/fastest/dart.html