Homework 6. Language bindings for TensorFlow

Motivation

Following up on the project, suppose you are trying to run an application server proxy herd on a large set of virtual machines. Your application uses machine learning algorithms and relies heavily on TensorFlow. You've built a prototype using Python and it runs as well as could be expected on large queries.

However, your application is unusual in that it handles many small queries, that create and/or execute models that are tiny by machine-learning standards. Although typical TensorFlow applications are bottlenecked inside C++ or CUDA code (e.g., in the Eigen or cuDNN libraries), when you benchmark your application you discover that it's spending most of its time executing Python code to set up your models.

Your boss suggests that a way to speed up performance would be to convert your application's Python code to some other language, and then use that instead of Python. You want to keep TensorFlow and the ability to prototype with Python, but you also want your application to run efficiently.

One other thing: your boss says that if a small query is initiated on a client, you should look for approaches that make it easy to answer the query directly on the client rather than shipping the problem off to an application server. Although this cannot be done for all the small queries (as some are initiated by servers or require resources available only on the servers), it can be done for a significant number of them.

Assignment

Review TensorFlow Architecture and TensorFlow in other languages, and consider three other languages that are plausible candidates for implementing your application: (1) Java (see bindings), (2) OCaml (see bindings), and (3) Kotlin (which does not have bindings yet, but one can use TensorFlow in Kotlin/Native).

You already know Java and OCaml from previous assignments. Familiarize yourself with Kotlin enough to write and test a simple method everyNth that accepts a list L and a positive integer N and returns a list containing every Nth element of L, starting with the (N−1)st element, then the (2N−1)st element, and so on until L's elements are exhausted so that the returned list's size is ⌊L.size / N⌋ (where ⌊*x*⌋ is the floor of *x*). The method should accept a list of any type, should return a list of the same type, and should consume O(N) time and space. The returned value should be immutable; any later changes to the input list should not affect the output list. Create a Makefile so that the command 'make check' compiles and runs your test successfully on the SEASnet GNU/Linux servers, using the Kotlin command-line compiler installed there.

Do some research on your three languages and support software as a potential platform. Your research should include an examination of the language and system documentation to help determine whether it would be effective. We want an alternative that supports event-driven servers well, such as the project servers using Python's asyncio.

Unlike the project, we are not expecting working prototypes, though prototypes are welcome.

Write an executive summary that compares the three alternate approaches to each other and to Python. The summary should be in 10-point font or larger and should be at most three pages. You can put references and appendixes in later pages, if you can't get under the page limit otherwise: the appendixes should contain any source code or diagrams. Your summary should focus on the technologies' effects on ease of use, flexibility, generality, performance, reliability; thie idea is to explore the most-important technical challenges in doing the proposed rewrite. The summary should be suitable for software executives, that is, for readers who have some expertise in software, particularly in managing software developers, but who are not experts in Java or OCaml or your chosen language. Please keep the resources for written reports and oral presentations in mind, particularly its rubrics and its advice for citations to sources that you consulted.

Submit

Submit a file hw6.pdf containing your summary. Submit your everyNthimplementation and test, along with any other prototype code, as a compressed tarball hw6.tar.gz in the usual way. Your tarball should include a Makefile such that the command make check builds and runs your test case. You can augment the makefile to build and run any other prototype code you submit.