A Big-data Analytics Framework with Efficient Support for Dynamic Languages

Luca Salucci
Università della Svizzera Italiana
luca.salucci@usi.ch

Walter Binder
Università della Svizzera Italiana
walter.binder@usi.ch

Daniele Bonetta
Oracle Labs
daniele.bonetta@oracle.com

1. The Problem

Over the last years, several different frameworks have emerged in the field of big-data analytics. Recent frameworks tend to expose a developer-friendly API via dynamic languages such as Python. Despite the benefits offered by such high-level languages in terms of programming model and developer productivity, their adoption has been limited because of the considerable runtime overhead due to inefficiencies in the integration of the dynamic language support.

In this paper we highlight the advantages of hosting multiple language runtimes in a single shared virtual machine and we have shown that it is possible to effectively employ our technique in the context of Big-data analytics. As a case study we applied this approach to the Spark runtime, which runs on top of the JVM, to integrate it with Java based implementations of Python and JavaScript, namely ZipPy [2] and GraalJS [1].

2. A shared VM for multiple language runtimes

Our modified runtime is capable of executing Python and JavaScript on top of the same JVM that executes Spark, unlike the existing integration of Spark and Python (named PySpark) which relies on spawning external processes running Python and communicate with them using Operating System channels such as pipes, preventing from exploiting the shared heap of the JVM. On the contrary in our solution multiple threads are spawned within the JVM and each of them has access to an instance of the guest language engine, implemented in Java, thus allowing the threads to execute the guest language code, for instance Python and JavaScript.

Another major characteristic of our solution is that it enables *cross-boundaries* JIT compilation, consisting on the possibility to perform type specialization on the guest language and to overcome limitations caused by the dynamic type system of the guest language. ZipPy and GraalJS make use of self-optimizing Abstract Syntax Tree (AST) intepreters [3] and use them to perform JIT compilation of the guest-language code and produce highly-specialized machine code based on the type detected at runtime. We argue that in the context of big-data analytics using dynamic languages this kind of type specialization optimization can be particularly effective.

Integrating Python and JavaScript as two different *guest languages* for Spark required to handle the generation of their ASTs and their execution from threads, as well as the conversions of data to and from the guest language.

We implement some common benchmarks for big-data analysis, we measured the performance of our proposed solution in the context of a single machine and we compared them against the performance of PySpark showing that our runtimes are able to outperform it and to get performance similar to the original Spark runtime using Scala.

3. Future Work

The work presented in this paper limits to the case of a single machine, thus it would be interesting to move to a cluster and perform further evaluations to check if we are still able to get improvements over PySpark even running ZipPy/Spark and JS/Spark on a cluster.

Hosting the host and the guest language on the same shared VM is a first step toward tighter integration between big-data analytics frameworks and the dynamic language they support; since in ZipPy/Spark and JS/Spark multiple languages run on the same VM it would be possible perform cross language optimizations and we intend to explore in this direction.

References

[1] Oracle GraalJS. URL `http://www.oracle.com/technetwork/oracle-labs/program-languages/overview/index-2301583.html`.

[2] ZipPy, a fast and lightweight Python implementation. URL `https://bitbucket.org/ssllab/zippy`.

[3] T. Würthinger, A. Wöß, L. Stadler, G. Duboscq, D. Simon, and C. Wimmer. Self-optimizing ast interpreters. *in Proc. of DLS '12*, 48(2):73–82, Oct. 2012. ISSN 0362-1340. .