Active-Code Replacement in the OODIDA Data Analytics Platform

Gregor Ulm, Emil Gustavsson, Mats Jirstrand
Fraunhofer-Chalmers Research Centre for Industrial Mathematics

FRAUNHOFER CHALMERS
RESEARCH CENTRE FOR INDUSTRIAL MATHEMATICS

Problem

OODIDA (On-board/Off-board Distributed Data Analytics) is a data analytics platform for the automotive domain. It targets fleets of reference vehicles and is intended for rapid prototyping. Multiple assignments can be carried out concurrently. An assignment consists of a task for the selected client devices and a supplementary task that is carried out on the cloud. The former is the on-board task, the latter the off-board task. A simple example is the detection of outliers in a bounded data stream on each client, which the cloud simply forwards to the user.
Our system can be deployed automatically. However, redeploying either the cloud or client application is disruptive as ongoing assignments have to be terminated and the corresponding part of the system restarted. Thus, we explored approaches for updating part of the system without interrupting ongoing workflows.

Contribution

We describe the design and implementation of active-code replacement in the OODIDA platform. This feature has been fully implemented and now further improves the suitability of OODIDA for rapid prototyping of algorithmic methods. Active-code replacement has been very useful in practice. With this feature, even the most complex use cases of OODIDA, such as federated learning [3], can be implemented *ad hoc* by a user of the system.

Solution

Context Figure 1 shows OODIDA in context. This system is distributed on user (nodes u_i and f_i), cloud (nodes b and a), and client hardware (nodes c_j and a_j). White nodes represent external applications creating assignment specifications or performing tasks, while shaded nodes are used for handling messages and distributing tasks. Node f is a user front-end for assignment creation and node u the local user node that communicates with the central cloud node b, which turns a global assignment into tasks for individual nodes. Cloud node a performs off-board tasks. Node c is a local client node running on an embedded device. It communicates with an external application a, also running on the client, that performs computational tasks.

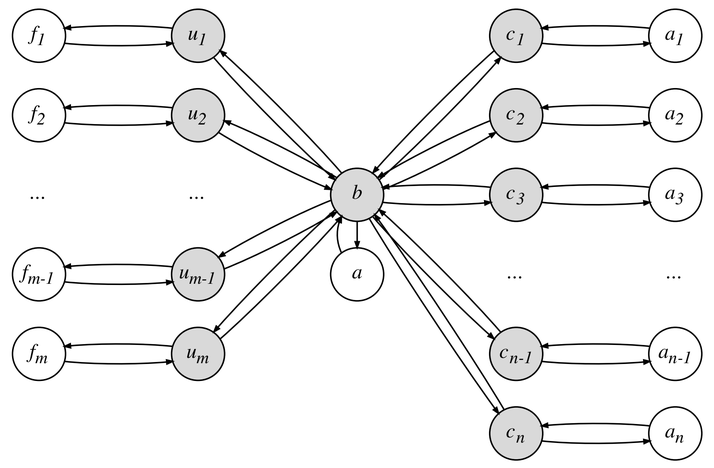

Figure 1: Schematic overview of the OODIDA platform, showing how the user (f_i and u_i), cloud (b and a), and clients (c_j and a_j) interact. Assignments are issued with the help of front-end nodes f_i. The goal of active-code replacement is to update software on nodes a and a_j at runtime.

Idea The user can supply a custom Python module for the cloud and/or the client. Distributing custom code piggybacks on the mechanism for sending assignments. Once custom code has been distributed, a custom on-board or off-board method can be used in an assignment specification.

Implementation Details The user front-end f performs static and dynamic checks on the Python source code. Among others, there are limitations on the used types and the libraries that are called. Functions also have to produce a return value within a certain amount of time. If those checks are passed, the provided source code gets encoded, and turned into the payload of an assignment.

Afterwards, the code is sent to node b, which forwards it to the specified subsets of clients. On the client, the original Python module gets gets decoded and saved. Once deployment has succeeded, the user receives a status update. When a user assignment calls custom code, the corresponding OODIDA code lazily loads that Python module anew with each iteration. This makes it possible to update custom code while an assignment is ongoing. For instance, the user can adjust parameters of an algorithm on an ongoing assignment, which is very helpful for exploratory work in data analytics.

Evaluation

Experiment In order to quantify the effect of active-code replacement, we set up OODIDA in an idealized environment where user, cloud, and client machines are connected via Ethernet. We first deployed the system from scratch and afterwards deployed a moderately-sized Python module.

Results A full deployment of the cloud and client installation take 23.6s and 40.8s, respectively. In contrast, deploying a custom Python module to the cloud and client takes 20.3ms and 45.4 ms, respectively. All times are the averages of five runs.

Discussion The quantitative difference between active-code replacement and a regular deployment amounts to three orders of magnitude. However, this comparison understates the problem of a full redeployment, which cannot be performed by a mere user of our system. Furthermore, there are organizational barriers that make an extension of any part of OODIDA a long-winded process. On the other hand, there are deliberate limitations on active-code replacement, which prevent the deployment of some algorithms, for instance because they require a missing library. Thus, both approaches are complementary rather than competitive.

Related Work

Active-code replacement is an extension of the OODIDA platform [5], which originated from `ffl-erl`, a framework for federated learning in Erlang/OTP [4]. The latter was a research system for exploring distributed data analytics with high levels of concurrency.

In terms of descriptions of systems that perform active-code replacement, Polus by Chen et al. [1] deserves mention. A significant difference is that it replaces larger units of code instead of isolated modules. It also operates in a multi-threading environment instead of the highly concurrent message-passing environment of OODIDA. We also noticed a similarity between our approach and Javelus by Gu et al. [2]. Even though they focus on updating a stand-alone Java application as opposed to a distributed system, their described "lazy update mechanism" likewise only has an effect if a module is indeed used. This mirrors our approach of only loading a custom module when it is needed, albeit our approach is more targeted towards rapid prototyping, as outlined in the Solution section.

Acknowledgements

This research was supported by the project On-board/Off-board Distributed Data Analytics (OODIDA) in the funding program FFI: Strategic Vehicle Research and Innovation (DNR 2016-04260), which is administered by VINNOVA, the Swedish Government Agency for Innovation Systems. Adrian Nilsson and Simon Smith assisted with the implementation.

References

1. Chen, H., Yu, J., Chen, R., Zang, B., Yew, P.C.: Polus: A powerful live updating system. In: 29th International Conference on Software Engineering (ICSE'07). pp. 271–281. IEEE (2007)
2. Gu, T., Cao, C., Xu, C., Ma, X., Zhang, L., Lu, J.: Javelus: A low disruptive approach to dynamic software updates. In: 2012 19th Asia-Pacific Software Engineering Conference. vol. 1, pp. 527 – 536. IEEE (2012)
3. McMahan, H.B., Moore, E., Ramage, D., Hampson, S., et al.: Communication-efficient learning of deep networks from decentralized data. arXiv preprint arXiv:1602.05629 (2016)
4. Ulm, G., Gustavsson, E., Jirstrand, M.: Functional federated learning in Erlang (ffl-erl). In: Silva, J. (ed.) Functional and Constraint Logic Programming. pp. 162–178. Springer International Publishing, Cham (2019)
5. Ulm, G., Gustavsson, E., Jirstrand, M.: OODIDA: On-board/off-board distributed data analytics for connected vehicles. arXiv preprint arXiv:1902.00319 (2019)