Data Analytics Service Composition and Deployment on IoT Devices

Jianxin Zhao, Tudor Tiplea, Richard Mortier, Jon Crowcroft, Liang Wang
Computer Laboratory, University of Cambridge, UK
firstname.lastname@cl.cam.ac.uk

CCS CONCEPTS

• **Human-centered computing → Ubiquitous and mobile computing**; • **Computing methodologies** → *Machine learning*; • **Software and its engineering** → *Software notations and tools*;

ACM Reference Format:
Jianxin Zhao, Tudor Tiplea, Richard Mortier, Jon Crowcroft, Liang Wang. 2018. Data Analytics Service Composition and Deployment on IoT Devices. In *MobiSys '18: The 16th Annual International Conference on Mobile Systems, Applications, and Services, June 10–15, 2018, Munich, Germany*. ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3210240.3223570

1 INTRODUCTION

Machine Learning (ML) techniques have begun to dominate data analytics applications and services. Recommendation systems are the driving force of online service providers such as Amazon. Finance analytics has quickly adopted ML to harness large volume of data in such areas as fraud detection and risk-management. Deep Neural Network (DNN) is the technology behind voice-based personal assistance, self-driving cars [1], image processing [3], *etc*. Many popular data analytics are deployed on cloud computing infrastructures. However, they require aggregating users' data at central server for processing. This architecture is prone to issues such as increased service response latency, communication cost, single point failure, and data privacy concerns.

Recently computation on IoT and mobile devices has gained rapid growth, such as personal data analytics in home [5] and DNN application on a tiny stick [6].HUAWEI has identified speed and responsiveness of native AI processing on mobile devices as the key to a new era in smartphone innovation [4].

Many challenges arise when moving ML analytics from cloud to IoT devices. One widely discussed challenge is the limited computation power and working memory of IoT devices.Personalising analytics models on different IoT devices is also a very interesting topic [7]. However, one problem is not yet well defined and investigated: the deployment of data analytics services. Most existing machine learning frameworks such as TensorFlow and Caffe focus mainly on the training of analytics models. On the other, the end users, many of whom are not ML professionals, mainly use trained models to perform inference. This gap between the current ML systems and users' requirements is growing.

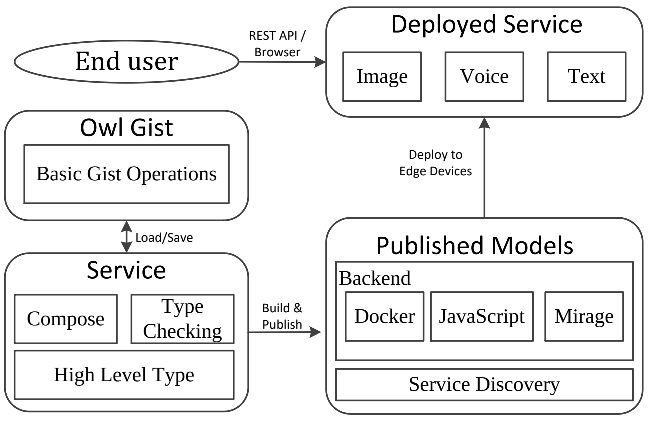

Figure 1: Zoo System Architecture

Another challenge in conducting ML based data analytics on IoT devices is model composition. Training a model often requires large datasets and rich computing resources, which are often not available to normal users. That's one of the reasons that they are bounded with the models and services provided by large companies. To this end we propose the idea *Composable Service*. The idea is that many services can be constructed from basic ML ones such as image recognition and recommendation to meet new application requirements. We believe that modularity and composition will be the key to increasing usage of ML-based data analytics.

Existing service deployment systems such as Clipper [2] and TensorFlow Serving [8] do not support type-safe service composing, nor do they offer flexible cross platform automatic deployment solutions. To this end, we present the design of the Zoo system to address the aforementioned two challenges. It provides concise Domain-specific Language (DSL) to enable composition of different data analytics services, and also deploys services to multiple backends. More detailed background can be seen in the authors' previous work [11].

2 SYSTEM DESIGN

The Zoo system is implemented on Owl [9], an open-source scientific computing library in OCaml language. Owl provides a full stack support for numerical methods, scientific computing, and advanced data analytics such as ML and DNN on OCaml.

Initially, Zoo system is designed to make it convenient for developers to share their OCaml code snippets via Github Gist. We further extend it to address the composition and deployment challenges. Fig. 1 shows the workflow of Zoo. It consists of two parts: *development* on the left side and *deployment* on the right. *Development* concerns the design of interaction workflow and the computational functions of different services. A normal Gist script will be

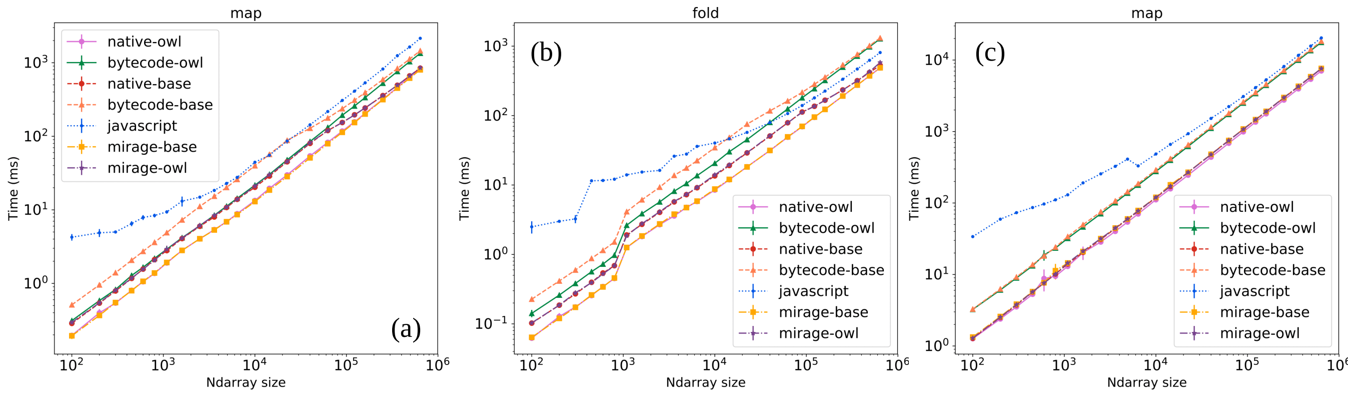

Figure 2: Performance of map and fold operations on ndarray on laptop (a-b) and RaspberryPi (c).

loaded as a module in OCaml. Functions from different Gists can be imported and composed to make new functions, which can then be saved to another Gist. *Deployment* takes a Gist and creates models in different backends. These models can be deployed to IoT devices. Service deployment and development are separated.

After showing the general workflow, the rest of this section will briefly introduce some major components in the Zoo system.

Gist is a core abstraction in Zoo. It is the centre of code sharing. However, to compose multiple analytics snippets, Gist alone is insufficient. For example, it cannot express the structure of how different pieces of code are composed together. Therefore, we introduce another abstraction: `service` in Zoo.

We observe that most current popular deep learning models can generally be categorised into three fundamental types: `image`, `text`, and `voice`. To make sure the type matches in service composition, we use generalised algebraic data types (GADTs) in OCaml. Type checking in OCaml ensures type-safe and meaningful composition of high level services.

Recognising the heterogeneity of IoT device deployment, one key principle of Zoo is to support multiple deployment methods. Zoo supports deploying services as Docker containers. Each container provides RESTful API for end users to query. Another backend is JavaScript. Using JavaScript to do analytics aside from front end development begins to attract interests from academia [10] and industry, such as Tensorflow.js and Facebook's Reason language. We also initially explore using MirageOS as an option. Mirage is an example of Unikernel, which builds tiny virtual machines with a specialised minimal OS that host only one target application. Deploying to Unikernel is proved to be of low memory footprint, and thus quite suitable for resource-limited IoT devices.

Zoo provides a minimal DSL for service composition and deployment. A user can simply use "`[$gid # n] $> s $@ backend`" in a script to get a service of name `n` from gist `gid`, compose it with service *s*, and deploy the new service in the format of a chosen `backend`.

Developers would modify and upload their scripts several times. As such, each version of a script is assigned a unique id in Gist. The naming scheme of a Gist is "`gid?vid?tol`", where a user can specify a version of a Gist using `vid`, and how often should the local Gist cache be updated using `tol`.

3 EVALUATION

As part of our evaluation, we compare the performance of different backends in Zoo. Since the Owl library contains functions that are implemented in C, it cannot be directly supported by `js-of-ocaml`, the tool we use to convert OCaml code into JavaScript. Therefore in Owl we have also implemented a "base" library in pure OCaml that shares the core functions of the Owl library. We compare two kinds of executables in OCaml: bytecode and native. Specifically, we observe the representative operations: `map` and `fold` operations on ndarray.

Fig. 2(a-b) show the performance of these two operations on ndarray. We use simple functions such as plus and multiplication on 1-d (size $< 1,000$) and 2-d arrays. The log-log relationship between total size of ndarray and the time each operation takes keeps linear. For both operations, the C/OCaml implementation of Owl is faster than `base`, and native executables outperform bytecode ones. The performance of Mirage executables is close to that of native code. Generally JavaScript runs the slowest, but note how the performance gap between it and the others converges when the ndarray size grows. For fold operation, JavaScript even runs faster than bytecode when size is sufficiently large. Fig. 2(c) shows similar conclusions on RaspberryPi 3 Model B.

4 ACKNOWLEDGMENTS

This work is funded in part by the EPSRC Databox project (EP/N028260/2), NaaS (EP/K031724/2) and Contrive (EP/N028422/1).

REFERENCES

[1] Mariusz Bojarski, Del Testa, et al. 2016. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316* (2016).
[2] Daniel Crankshaw, Xin Wang, Guilio Zhou, Michael J Franklin, Joseph E Gonzalez, and Ion Stoica. 2017. Clipper: A Low-Latency Online Prediction Serving System.. In *NSDI*. 613–627.
[3] Google. 2017. Google Cloud Vision API. https://cloud.google.com/vision. Accessed Nov. 11, 2017.
[4] Huawei-News. 2017. HUAWEI Reveals the Future of Mobile AI at IFA 2017. http://consumer.huawei.com/en/press/news/2017/ifa2017-kirin970/. [Online; accessed 10-Nov-2017].
[5] Richard Mortier, Jianxin Zhao, Jon Crowcroft, Liang Wang, Qi Li, Hamed Haddadi, Yousef Amar, et al. 2016. Personal Data Management with the Databox: What's Inside the Box?. In *Proceedings of the 2016 ACM Workshop on Cloud-Assisted Networking*. ACM, 49–54.
[6] Movidius. 2017. Movidius Neural Compute Stick. https://developer.movidius.com/. Accessed Nov. 11, 2017.

[7] Sandra Servia Rodríguez, Liang Wang, Jianxin R. Zhao, Richard Mortier, and Hamed Haddadi. 2018. Privacy-preserving Personal Model Training. *Internet-of-Things Design and Implementation (IoTDI), The 3rd ACM/IEEE International Conference on* (2018).
[8] TensorFlow Serving. 2017. TensorFlow Serving. https://www.tensorflow.org/serving/. [Online; accessed 14-Mar-2018].
[9] Liang Wang. 2017. Owl: A General-Purpose Numerical Library in OCaml. *CoRR* abs/1707.09616 (2017). http://arxiv.org/abs/1707.09616
[10] Liang Wang, Sotiris Tasoulis, Teemu Roos, and Jussi Kangasharju. 2016. Kvasir: Scalable provision of semantically relevant web content on big data framework. *IEEE Transactions on Big Data* 2, 3 (2016), 219–233.
[11] Jianxin Zhao, Richard Mortier, Jon Crowcroft, and Liang Wang. 2018. Privacy-preserving Machine Learning Based Data Analytics on Edge Devices. *AAAI/ACM Conference on Artificial Intelligence, Ethics, and Society* (2018).