Research statement

Marcel Wagenländer
Imperial College London
marcel.wagenlander19@imperial.ac.uk

Abstract

I aim to develop ML systems that separate training processes, specifically model and data pre-processing, from the underlying computational resources. There is a focus on performance because training large models is resource-intensive and training times are long. Optimizing performance leads to ML systems being highly intertwined with the resources, such as model-specific distributions to resources. Flexibility for adaptations is missing, e.g. to move the training to different resources.

The goal is to decouple ML frameworks from hardware to enable device-independent computations. By introducing new abstractions in the ML stack between the expression of training and the execution on hardware. Clear boundaries enable new features and easier optimizations of components. E.g. systems can scale the number of resources during training and improve cluster utilization. This approach enhances the efficiency of ML training, reducing both time and environmental impact. Additionally, ML systems can scale the model size up more easily to better performance on tasks.

Machine learning (ML) has remarkable achievements in, e.g., computer vision, recommender systems, and conversational artificial intelligence. People can search for objects in photos due to object recognition, generate images from text with generative AI, and send voice commands to their devices. On the ML theory side, it is due to milestones such as stochastic gradient descent (SGD), the ResNet or Transformer models, and the Adam optimizer. On the ML systems side, it was enabled by huge increases in computational power and DL frameworks, such as Tensorflow or PyTorch. The frameworks allow the ML scientists and engineers to express the training and allocate the computations to the resources.

Focus on performance. To be able to scale up ML training for huge datasets and large models a focus on performance is required because otherwise, the training times are too long to see models converge. Therefore, ML frameworks became highly intertwined with the hardware to optimize the execution of the computations. While the performance benefits are great for training times, the execution of the training is very rigid. The current practice is to write device-specific code describing the ML training, as shown in Fig. 1. The data pre-processing and model definition are expressed for each device. It includes the communication between devices, i.e. which devices communicate with which other devices and how. Additionally, the descriptions need to define which data samples and model parameters are required.

It entails multiple issues (1) The distribution of computations to multiple accelerators are *model-specific* to get the last bit of performance out of the hardware. The decision of how to distribute the computations for ML training is made on the same level of abstraction as the definition of the ML model and training. (2) Even though ML training can be very long-running, ML frameworks *don't account for resource changes*. More resources can become available but cannot be incorporated and taken advantage of. (3) Optimizing the execution for certain resources, can be a problem to *reproduce the results* of a training with different resources. It needs to be evaluated if the hyper-parameters used work with the different resources. For instance, there can be the issue of not being able to use the same batch size because of less memory being available.

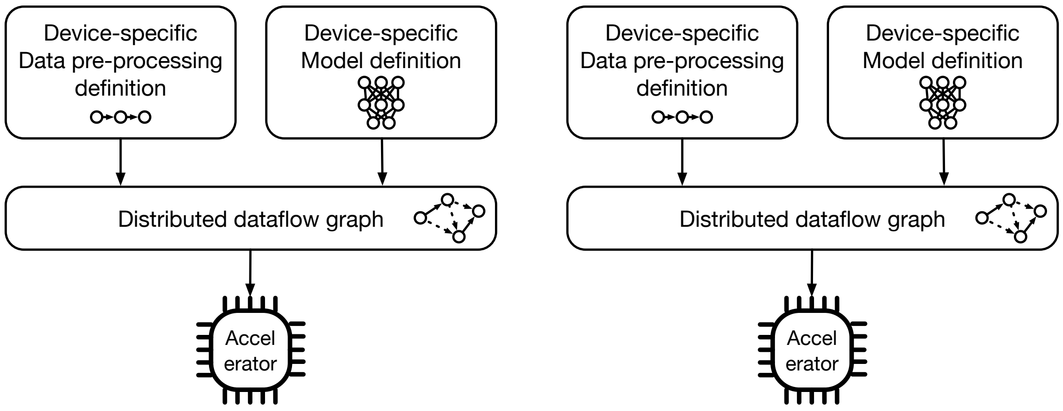
Device-specific Data pre-processing definition
Device-specific Model definition
Device-specific Data pre-processing definition
Device-specific Model definition
Distributed dataflow graph
Distributed dataflow graph
Accel erator
Accel erator

Figure 1: Current ML stack

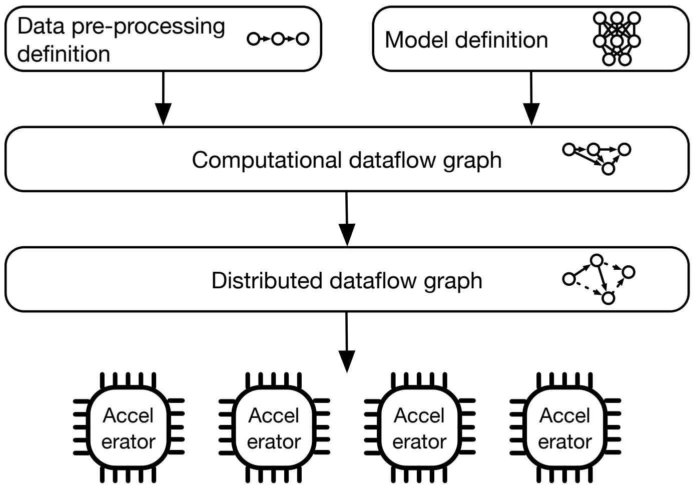
Data pre-processing definition
Model definition
Computational dataflow graph
Distributed dataflow graph
Accel erator
Accel erator
Accel erator
Accel erator

Figure 2: Proposed ML stack

Disentangle ML Frameworks From Resources. While performance is key for ML training, there should not be such a device-specific definition of the training. As shown in Fig. 2, I propose to disentangle the ML framework from the resources by introducing new abstractions through new layers in the ML stack. An ML engineer should design model training without needing to tailor it to specific computational resources. The definition of data pre-processing and model is translated into a united computational dataflow graph. A dataflow graph that is device-independent, meaning only expressing the mathematical computations. Using the computational dataflow graph, the resources are considered and a distributed dataflow graph is created. It allows the optimization of dataflow graph mapping to the ML accelerators. With clear abstraction layers, everyone can focus on their expertise, e.g. improving the model structure or optimizing the distribution of computations.

Given the high cost of accelerators, organizations must keep the utilization of accelerators high, while ensuring ML jobs to achieve their training goals in terms of e.g. model accuracy and training throughput. Users submit ML jobs to ML schedulers, which allocate them to resources. An emerging requirement for ML schedulers is that they must change the accelerator allocation of long-running ML jobs *dynamically at runtime*. This has several reasons: (1) *elasticity*—to maintain high cluster utilization, ML jobs must claim extra accelerators resources when they become available, (2) *redeployment*—ML jobs may have to release GPU resources and move to other resources to reduce fragmentation, support hardware maintenance, or handle preemption by higher priority jobs, and (3) *failure recovery*—ML jobs may lose accelerators at runtime due to failure and must continue training with a subset of resources after recovering from checkpoints.

Parameters are adapted *dynamically* during training. For example, many models only reach high accuracy if the learning rate is decreased as the model converges, the batch size can be set dynamically based on real-time gradient metrics, and the communication strategy between workers can be adapted to the current training loss. Similarly, system parameters can be updated to react to changes in exploitable levels of parallelism and resource availability. For example, the number of workers can be changed according to the observed resource utilization, thus improving the utilization of expensive accelerators. *KungFu* [1] is a distributed ML training library that is designed to adapt configuration parameters at runtime. The key idea is to support *Adaptation Policies* written by users, which change hyper- and system parameters during training based on real-time monitored metrics.

Many large-scale DL jobs are run in the cloud. With long training times, there is an economic incentive for distributed training on transient virtual machines (VM), which can yield cost reductions. It should be possible for distributed DL jobs to run *exclusively* on transient VMs. Systems face challenges such as workers may be added and removed from the training when transient VMs become available and are revoked, and as a cluster changes configuration parameters of the DL job are affected. Therefore, it is hard to deploy distributed DL jobs currently. *Spotnik* [2] is a distributed ML system for transient VMs that explores *adaptive collective communication* by using a novel communication layer that handles dynamic changes to the membership in the cluster.

The design of current DL frameworks, such as PyTorch, TensorFlow, or JAX are ill-suited to allow DL schedulers to change GPU resources at runtime. DL jobs executed by current frameworks lack a property that we term *device-independence*: DL jobs are tightly coupled to accelerators at deployment time, preventing DL schedulers from changing the allocation at runtime. Additionally, changing the number of accelerators of a DL job with multi-dimensional parallelism can affect model convergence. The idea is to create a new *state management library* that externalizes the training state, i.e. the model and data pre-processing state, from a DL job in the DL framework and then transforms the state at runtime in response to resource changes. Thereby, *Tenplex* (unpublished) enables DL jobs with multi-dimensional parallelism to support changes to accelerator resources during training.

Future trajectory. Having made good progress in exploring the concept of disentangling ML frameworks and resources, there are still features possible that can be achieved by disentangling. One of the features is failure recovery. Long-running machine learning jobs face the issue of losing a lot of training progress in case of failure which failure recovery can mitigate. A failure of an accelerator implies a resource change with the addition of a potential loss of training state. The idea is to use the capability of dynamically changing the resources during runtime and a replication of the training state. I will investigate whether to maintain state in persistent storage or through replication among workers. Additionally, I will evaluate and compare different approaches to see which approaches have the least overhead what the overhead depends on, and what the bottlenecks are.

Another interesting directory is to explore ML training in which the model learns its structure. During the training, connections between artificial neurons are added and removed. It has implications for the ML system when the computations and therefore the computations dataflow graph changes. The question is how can a ML system support the computation changes with a minimum degradation of performance. Does the proposed disentangled ML stack work or must it be extended to properly support it? I want to find out if the computational graph should become more fine-grained and incorporate the learning of connections between neurons. Additionally, the exploration includes comparing it to enabling and disabling parameters and whether a combination of both has the best performance. Sparse ML models are of research interest and there might be techniques used for sparse models which can be applied to structure-changing ML models. Better support for model structure changes would help ML scientists better explore the space.

References

[1] Luo Mai, Guo Li, Marcel Wagenländer, Konstantinos Fertakis, Andrei-Octavian Brabete, and Peter Pietzuch. Kungfu: Making training in distributed machine learning adaptive. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pages 937–954.

[2] Marcel Wagenländer, Luo Mai, Guo Li, and Peter Pietzuch. Spotnik: Designing distributed machine learning for transient cloud resources. In *12th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 20)*. USENIX Association, July 2020.