MDSC: Modelling Distributed Stream Processing across the Edge-to-Cloud Continuum

Daniel Balouek-Thomert, Pedro Silva, Kevin Fauvel, Alexandru Costan, Gabriel Antoniu, Manish Parashar

MDSC: Modelling Distributed Stream Processing across the Edge-to-Cloud Continuum

Daniel Balouek-Thomert
daniel.balouek@utah.edu
University of Utah, SCI Institute, USA

Pedro Silva
pedro.silva@hpi.de
Hasso-Plattner Institut, Uni. Potsdam

Kevin Fauvel
kevin.fauvel@inria.fr
Univ Rennes, Inria, CNRS, IRISA

Alexandru Costan
alexandru.costan@irisa.fr
Univ Rennes, Inria, CNRS, IRISA

Gabriel Antoniu
gabriel.antoniu@irisa.fr
Univ Rennes, Inria, CNRS, IRISA

Manish Parashar
manish.parashar@utah.edu
University of Utah, SCI Institute, USA

ABSTRACT

The growth of the Internet of Things is resulting in an explosion of data volumes at the Edge of the Internet. To reduce costs incurred due to data movement and centralized cloud-based processing, it is becoming increasingly important to process and analyze such data closer to the data sources. Exploiting Edge computing capabilities for stream-based processing is however challenging. It requires addressing the complex characteristics and constraints imposed by all the resources along the data path, as well as the large set of heterogeneous data processing and management frameworks. Consequently, the community needs tools that can facilitate the modeling of this complexity and can integrate the various components involved. In this work, we introduce MDSC, a hierarchical approach for modeling distributed stream-based applications on Edge-to-Cloud continuum infrastructures. We demonstrate how MDSC can be applied to a concrete real-life ML-based application - early earthquake warning - to help answer questions such as: when is it worth decentralizing the classification load from the Cloud to the Edge and how?

KEYWORDS

Computing Continuum, Stream Processing, Modelling

ACM Reference Format:
Daniel Balouek-Thomert, Pedro Silva, Kevin Fauvel, Alexandru Costan, Gabriel Antoniu, and Manish Parashar. 2018. MDSC: Modelling Distributed Stream Processing across the Edge-to-Cloud Continuum. In *Woodstock '18: ACM Symposium on Neural Gaze Detection, June 03–05, 2018, Woodstock, NY*. ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

With the proliferation of the Internet of Things (IoT), we have witnessed the development and popularization of new Machine Learning (ML) applications aiming to leverage distributed and heterogeneous data sources to automate decision-making. These applications bring new data processing challenges, in particular related to the need to process large volumes of streamed data under high-throughput and low-latency constraints.

Moving away from traditional centralized Cloud models, the use of resources spanning at the Edge of the infrastructure and in the Fog (i.e., between the Edge and Cloud), allows to distribute analytics while preserving low latency, high availability, and privacy. This aggregation of heterogeneous resources along the data path from the Edge to the Cloud, also referred to as the Edge-to-Cloud Continuum, or the Computing Continuum [9, 11], can be harnessed to support distributed analytics.

As IoT and ML applications are powered by pipelines of multiple data processing frameworks [5, 7, 13], the integration of the different software, libraries and platform components is a significant challenge on large and heterogeneous infrastructures. Understanding the dependencies between the application workflows to best leverage the underlying infrastructure is crucial for the end-to-end performance. However, we are currently missing models enabling this adequate mapping of distributed analytics pipelines on the Edge-to-Cloud Continuum.

To tap into the power of the Computing Continuum, we propose MDSC, an approach for **m**odeling **d**istributed **s**tream-based applications running across the Edge-to-Cloud **c**ontinuum. The objective of MDSC is to facilitate the definition and evaluation of alternative deployment options for such applications atop the Edge-to-Cloud Continuum, in a versatile way.

To enable this vision, MDSC leverages a set of simple but expressive architectural concepts (e.g., *layers*, *processing units*, *buses*), which aim to describe the essential components of the application and to make more explicit their inter-relations and functionalities. These abstractions further enable grouping for components of similar nature and allow to characterize the communication among those groups. Subsequently, this model facilitates the definition of alternative mappings onto the underlying edge/fog/cloud infrastructure.

The contributions of this paper are:

- **a hierarchical approach for modeling distributed stream-based applications** on the Edge-to-Cloud Continuum – MDSC (Section 3);
- an illustration of how this model can be applied to a complex ML-based real-life application for Earthquake Early Warning – MDSC EEW (Section 4);
- **a large-scale experimental validation** of MDSC EEW through various experiments on an Edge-to-Cloud infrastructure emulated atop the Grid'5000 testbed (Sections 5 and 6);

Woodstock '18, June 03–05, 2018, Woodstock, NY

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/10.1145/1122445.1122456

2 BACKGROUND AND CHALLENGES

State-of-the-art stream-based analytics applications are mostly built on the premise that data ingestion, management, and processing are done in centralized locations by leveraging general purpose analytics frameworks such as Apache Flink [13] or Apache Spark [34]. At the same time, legacy single-machine ML libraries start to receive support for execution in distributed settings, based on dataflow models [8, 23]. Efforts from major Cloud providers enable ML capabilities when consuming data from devices at large scale but do not offer the ability to orchestrate application components across hybrid Edge-Cloud infrastructures [2, 5, 7]. *Edge Analytics systems* take advantage of the processing power of machines located at and near the Edge of the network in order to accelerate and to scale up analytics applications with minimum resource usage.

This decentralization of stream-based data analytics systems leads to multiple challenges:

Expressive and versatile models for both the application and the infrastructure. To effectively leverage the computing power near the network edges, powerful and expressive models are necessary to help describe the application by considering the possible configurations and limitations of the underlying infrastructure.

Sustained real-time processing Many Edge analytics systems (ML-enabled or not), such as home assistants and health care systems, require real-time or near real-time processing latency. Achieving it is non-trivial when processing huge volumes of data streamed from heterogeneous sources.

Effective processing distribution considering constrained resources Complex trade-offs needs to be considered when deploying stream-based analytics pipelines across the Continuum: which parts should be executed at the Edge and which in the Cloud?

Given their different scopes, addressing all these challenges within a single framework is a non-trivial endeavour. There is already work on processing distribution [17] and ML on constrained resources [18]. Therefore, in this paper, we focus on the first challenge: provide *expressive and versatile models for the application and for the infrastructure*, as an effective means to enable real-time processing in analytics applications deployed on the Edge-to-Cloud Continuum (the second challenge).

3 MDSC: AN APPROACH FOR MODELLING DISTRIBUTED STREAM-BASED ANALYTICS

In this section, we introduce MDSC, a modelling approach which aims to help developers organize the different components of processing frameworks used by analytics applications (ML-based or not), separate their concerns and roles, and provide a set of good practice guidelines for distributing data processing on the Computing Continuum. MDSC can be summarized as a composition of *layers*, consisting of *units*, connected by *buses*.

3.1 Modeling Infrastructures as Layers

Processing Layers A layer is an abstract representation of a subset of computing resources of an infrastructure. The main objective of a layer is to describe data processing and characterize the data flow among resources. The strategy for grouping resources is mainly related to the location of the resources and their processing objective. For example, in an infrastructure composed of Edge, Fog and Cloud, there could be 3 different layers, one for each part of the infrastructure. Each layer groups resources with different characteristics.

Processing units are responsible for data processing (e.g., data transformation, aggregation, or filtering) by means of user-defined logic. Key components of processing units are *libraries*, *data connectors* and *user-defined logic*, that can be extended by system designers. The objective of modeling processing units is to have a straightforward and compact view of the available processing resources of a layer, of their composition and interactions. For example, in a ML application, a processing pod could define the Python Standard Library and scikit-learn [22] as libraries, a Docker image containing an off-the-shelf third party software as user-defined logic, and an implementation of a Kafka connector class as a data connector.

Helper Units provide an environment from which processing units can benefit in order to process data. helper units may be ingestion systems, key-value stores, load-balancing servers, or proxy servers. Notice that a unit does not necessarily map to an entire machine. It can represent management resources and can be encapsulated by a virtual machine, a container, or a process.

Buses They connect layers or units inside a layer. Units inside of the same layer share the same network, however this is not necessarily the case for units from different layers. Helper units have the responsibility of pulling or receiving data through the network and making it accessible for processing units.

3.2 Modeling Applications as UDL Components

The main role of UDL (*user-defined logic*) components of processing units is to expose their data processing logic, i.e., the procedures defined by the user to transform the data. Consequently, UDL components are responsible for mapping the application itself and its placement among the different processing units and layers. For example, UDL components can be encapsulated in a Java program, a standalone binary, a container, or in the processing operators of a workflow/dataflow or stream processing engine, such as Flink.

3.3 Features

MDSC has two main features which help the application and infrastructure modeling process.

Organization is crucial when designing complex systems because of the amount of different technologies that need to work together, and their consequent configurations and characteristics. With MDSC, the concepts of *layers and buses* and *processing/helper units* help the designers organize and arrange together the different frameworks and technologies at their hands. While conceptually simple, these abstractions are powerful in terms of expressiveness (as illustrated in Section 4).

Separation of concerns is enhanced by organization. MDSC aims at grouping similar roles and functionalities together so that the interactions as well as the interference of the frameworks and technologies become clearer. We argue that besides helping the modeling process, these features allow for a compact and easy to understand way of abstracting an application, especially in environments shared by designers with different backgrounds (as it is often the case of ML applications).

4 MODELING AND DEPLOYING A REAL-LIFE ML APPLICATION ON THE EDGE-TO-CLOUD CONTINUUM

In this section we illustrate the use of MDSC to model an ML-based *Earthquake Early Warning* system.

4.1 DMSEEW: An Algorithm for Distributed ML-Based Earthquake Early Warning

Earthquake Early Warning (EEW) systems provide earthquake alerts before the shaking damage of a seismic event reaches sensitive areas, giving governments and communities a time window of seconds to minutes to take protective actions. Traditionally, EEW is executed in a fully centralized fashion with data from sensors being sent to Clouds.

In a previous work [16], we proposed moving part of the sensor data processing towards the Edge to speed-up detection while further enhancing network usage reduction, fault tolerance, and idle machines usage. The Distributed Multi-Sensor Earthquake Early Warning (DMSEEW) takes sensor-level class predictions (normal activity, medium earthquake or large earthquake) based on the data gathered by each sensor, aggregates them using a bag-of-words representation, and use it to predict the final earthquake category. A high-level illustration of DMSEEW and its important processing steps are presented in Figure 1.

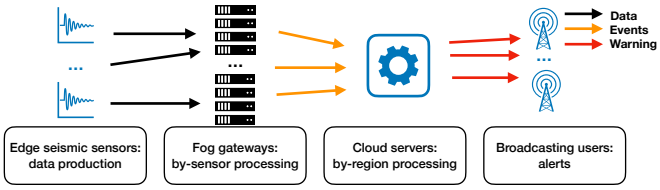

Figure 1: Earthquake Early Warning use-case (DMSEEW).

4.2 Modeling the DMSEEW Hierarchical Processing

By-sensor classification To perform the by-sensor classification, it is necessary to gather measurements streamed by each sensor and apply the WEASEL+MUSE [27] classifier on those data. To gather and process the data without interrupting the stream, we use *sliding windows* configured to hold and then process 30 seconds of data every 1 second. The processing performed at the moment that the window data is evaluated can be summarized as *sorting* the data points and *running* the WEASEL+MUSE classifier on the resulting time-series. Hence, each sensor produces a stream of seismic measurements which is processed and transformed into a stream of predictions.

By-region prediction Sensors are organized into regions, with the objective of predicting the magnitude of eventual seismic events in each of those regions. As sensors generate streams of by-sensor predictions, it is necessary to filter those streams by region, then, for each stream containing only predictions of a given region, to predict the final event category. For doing that, we use 1-second *tumbling windows* (i.e., processing abstractions that store the last 1 second of data before processing it) to gather predictions from a given region and we calculate the final prediction using the 1NN classifier on a normalized frequency vector of predictions.

4.3 Deploying DMSEEW on the Edge-to-Cloud Continuum

MDSC EEW is a model of DMSEEW mapped on the hybrid infrastructure, based on MDSC (Figure 2). In summary, it uses 4 layers: (i) **Edge**, where the data are produced; (ii) **Fog**, where data are pre-processed (by-sensor classification); (iii) **Cloud**, where the partially processed data are gathered and finally processed (by-region prediction); and (iv) **Bus**, connecting the above layers.

The Edge layer holds seismic sensors which act as processing units, grouped by region. Their only processing capability is sending the data to a predefined gateway server, located in the Fog. Each group within the Edge layer has one bus which models the LAN that connects all sensors isolated from the outside world and forwards sensor data to other networks. Note that in this example we consider 50 sensing regions; for each region we define a separate group of 20 processing units inside the Edge layer.

The Fog layer is essentially composed of gateway machines. They aggregate data coming from a set of sensors in the Edge and process them. For each of the 50 sensing regions we consider one fog-level gateway with an MQTT server, which ingests data coming from the sensors, and a Java program, that pulls data from the MQTT server, processes them and sends the results to the Cloud. Hence, we model every MQTT server as a helper unit and every Java program as a processing unit. Furthermore, processing units have access to different libraries such as Java JDK, Eclipse Paho, and WEASEL+MUSE. They describe the by-sensor classification step as user-defined logic and use connectors to the MQTT servers and to the Kafka cluster.

The Cloud layer is mainly composed of an Apache Kafka cluster, an Apache Zookeeper server, modeled as helper units, and an Apache Flink cluster, whose Task Managers are modeled as processing units. Apache Kafka brokers ingest the data, Apache Zookeeper manages Kafka metadata, and Flink processes the data pulled from Kafka and pushes the results back to Kafka. processing units rely on many libraries to process the data, such as Java SDK, the Flink API, and the ML library LIBSVM [6]. These libraries are used by the user-defined logic component implementing the by-region prediction step (cf. Section 4.2).

Buses model the intra- and inter- processing layers communication. Each Edge-level group of processing units has its own LAN and can communicate with the corresponding Fog-level helper unit through high speed region LANs. The Fog layer communicates with the Cloud layer using Internet or a dedicated network.

5 EVALUATION METHODOLOGY

To illustrate the versatility of the MDSC modeling, we used it to enable a comparative performance analysis of the Edge-based implementation of DMSEEW with various cloud-only implementations. The goal of this evaluation is to show that: *a)* MDSC is able to gracefully scale up to thousands of sensors and tens of processing nodes; *b)* MDSC isolates performance bottlenecks at various levels, from application to infrastructure; *c)* MDSC is able to identify trade-offs between Edge-to-Cloud and Cloud-only deployments;

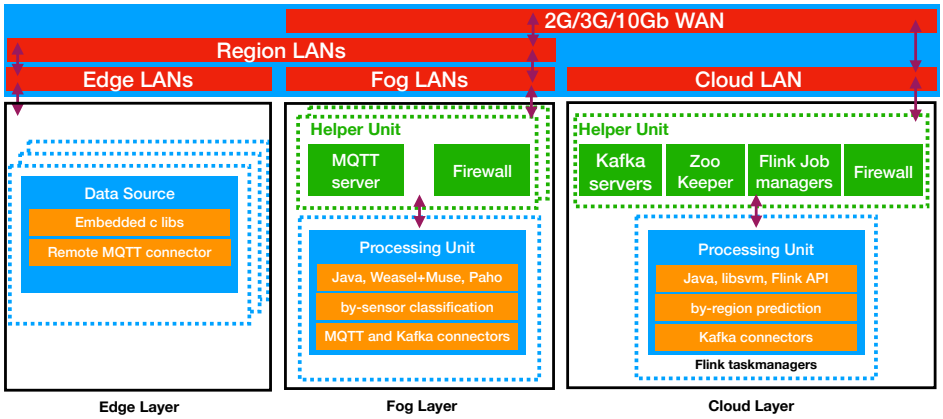

Figure 2: Overview of MDSC EEW: applying MDSC for modelling the EEW use-case.

5.1 Baselines

The default Edge-to-Cloud MDSC implementation of EEW, using the WEASEL+MUSE predictor, is further denoted as WME and will be compared to three different baselines: *WEASEL+MUSE EEW Cloud-only* (WME-CO), *Random EEW* (RE), and *Random EEW Cloud-only* (RE-CO). They are all based on MDSC but differ in the layer where the by-sensor data processing is performed and the prediction library used for that purpose, as shown in Figure 3.

WME-CO The by-sensor data processing step is performed on the Cloud, i.e., the Fog machines only forward the data produced by the sensors to the Cloud. The prediction library used is the same as MDSC, i.e., WEASEL+MUSE.

RE The by-sensor data processing is performed on the Fog, as in MDSC, using an integer pseudo-random generator as predictor. This baseline acts as a lower bound for Edge computing scenarios, since, in practice, there is no processing done for calculating the predictions.

RE-CO The by-sensor data processing is performed uniquely on the Cloud and the by-sensor and the by-region predictions are calculated using an integer pseudo-random generator. This baseline acts as a lower bound for Cloud-only scenarios.

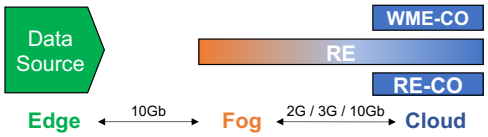

Figure 3: Baselines summary according to the layers used for by-sensor and by-region processing.

5.2 Cyber-Infrastructure

We implemented the architecture described in Section 3 and deployed it on the *gros* cluster from the Grid'5000 experimental test bed [15]. We rely on the prototype of an experimental platform [26, 29] able to set up the environment, execute the experiments and gather metrics and results. This platform was designed based on a methodology ensuring the repeatability of the experiments, so that the comparison of MDSC with the baselines remains fair. In absolute numbers, the resulting distributed EEW system has

	Bandwidth	Minimum latency	Jitter
2G	0.3 Mbps	300 ms	700ms (normal)
3G	2Mbps	100 ms	400ms (normal)
10Gb	10Gbps	0ms	0ms

Table 1: Fog-to-Cloud network configurations used for the evaluation of MDSC EEW

1,000 sensors emulated by virtual machines deployed on 25 host machines. Each set of 20 sensors composes an Edge layer and it is connected to one of the 50 implemented Fog layers. All Fog layers communicate and transfer their data to one Cloud layer containing a 5-machine Apache Kafka cluster, a 5-machine Apache Flink cluster and one Apache Zookeeper server. We describe in detail in Table 1 the values of bandwidth, minimum latency and jitter for each type of network configuration.

The end-to-end latency measures in milliseconds the time elapsed between the production of an event by a sensor and the prediction of the corresponding seismic event on the Cloud. This metric contains the by-sensor window processing time (maximum of 30s plus the classifier execution time) and the by-region window processing time (maximum of 1s plus the processing time). It also contains Edge-to-Fog and Fog-to-Cloud transfer times.

5.3 Datasets

Initially, our objective was to use real seismic data[1] from the American Incorporated Research Institutions for Seismology (IRIS)[1], as in [16]. Nevertheless, with its size of 58MB from 897,060 data points, it is not large enough for stressing the infrastructure of the scenarios proposed in this work. Hence, we artificially generate data using the IRIS format. We generate 12,000 messages for each of the 1,000 sensors distributed in 50 regions with a frequency of 100Hz. Hence, each sensor produces 100 messages per second for 2 minutes (resulting in a total of 12,000,000 messages to be processed).

6 EVALUATION RESULTS

We first assess the performance of the lower bound RE and RE-CO baselines. Then, we compare the WME and WME-CO scenarios, highlighting issues related to network and library constraints.

[1] https://figshare.com/articles/EarthquakeEarlyWarningDataset/9758555

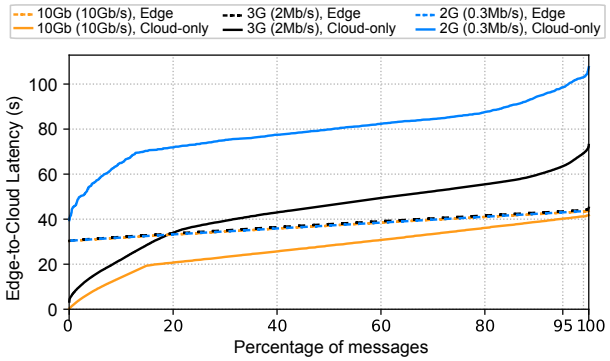

Figure 4: Latency resulting from processing one seismic event on lower bound scenarios RE and RE-CO.

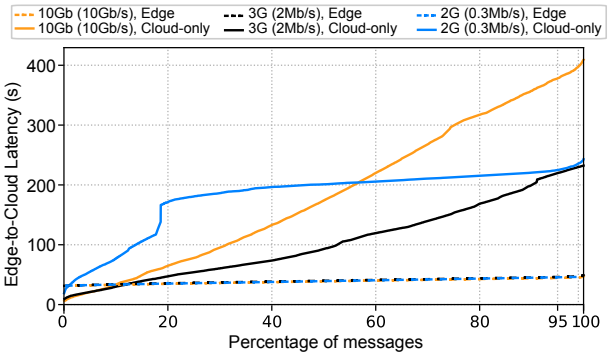

Figure 5: Latency resulting from processing one seismic event on WME and WME-CO scenarios.

6.1 Random EEW: Lower Bounds

We start by presenting the performance of RE and RE-CO, the lower bound scenarios that are used as a basis of comparison to the other scenarios throughout this section. The main characteristic of those scenarios is that predictions are calculated through the generation of pseudo-random integers whenever the processing windows are ready to be evaluated. Due to that, the main performance bottleneck of this scenario is the network, as observed in Figure 4.

The behavior of RE experiments is very similar because, in Edge computing scenarios, messages are aggregated in the Fog and only the predictions are sent to the Cloud, generating less data load on the network. Consequently, even on network constrained scenarios, the latencies only vary between 30 and 45 seconds. The difference between the amount of data transferred from the Fog to the Cloud is illustrated in Table 2. There are more than 13 times less data being sent to the Cloud compared to RE-CO scenarios.

On RE-CO experiments, the impact of network constraints is shown by the translation of the latency curves as the network characteristic changes. Observe that the RE-CO 10Gb scenario has better latency than the Edge scenarios, showing that, when the network is not constrained, processing everything on the Cloud can be a better option in terms of latency.

WME-2G/3G/10Gb	WME-CO-3G	WME-CO-2G	WME-CO-10GB
9,100	1,647,126	11,695,871	12,000,000

Table 2: Amount of messages transferred between Fog and Cloud on different network configurations for WME and WME-CO experiments. The same behavior is observed on RE and RE-CO experiments.

6.2 MDSC EEW: Performance Overview

We can now discuss the performance of WME (the vanilla MDSC implementation of EEW on Edge-to-Cloud) using WME-CO, RE and RE-CO as baselines. We show that despite the amount of computing resources available, the latency of WME is directly affected by the choice of the classification library used in DMSEEW. The extent of the performance degradation caused by that library is further discussed using the results of WME-CO.

We illustrate the WME and WME-CO scenarios in Figure 5. The main observation from this chart is the gap between the latencies of the WME and WME-CO experiments. While WME's latencies vary between 40 and 50 seconds, 90% of WME-CO latencies are above 100 seconds.

Starting with WME-CO, we observe very high latencies even for 10Gb scenarios. While the lower bound performance (cf. Figure 4) indicated that 10Gb scenarios provide better latencies than Edge computing ones, in Figure 5 that scenario has latencies close to 400 seconds. We identify, however, a similar behavior of IE compared to the lower bound RE, but with slightly higher latencies, varying between 32s and 50s.

The behavior of WME-CO observed in Figure 5 can be explained by the existence of two bottlenecks: the network, in 2G and 3G scenarios, as discussed in Section 6.1, and the interaction between WEASEL+MUSE, the ML library used for predicting the sensor-wise magnitudes of events, and Apache Flink. The implementation of WEASEL+MUSE [28] is a prototype, hence, it is not optimized in terms of execution time complexity, and it is not thread-safe. Because of the latter, it is not possible to share the same instance of the model among processing window instances running in the same Flink Task Manager. Since there will be one new open window per sensor every 1 second and for 30 seconds, that procedure demands more memory than the available one on each Task Manager, slowing down the data processing due to garbage collection and swapping.

7 RELATED WORK

A rich set of tools, libraries and frameworks are currently proposed at the intersection of Edge computing and ML. In this section, we provide an overview of the state of the art of the efforts from the converging communities of Big Data and Artificial Intelligence at the network edge.

Stream processing with ML libraries. In the Big Data community, the baseline approach for ML tasks like prediction and classification is to periodically transfer all the raw data from the Edge to the Cloud. Back-end stream processing frameworks like Apache Spark [34] and Apache Flink [13] perform advanced analytics once the data is collected. They are supported by rich dedicated libraries like MLlib [20] and FlinkML [3] respectively. These frameworks are greedy in terms of resources and cannot be executed on Edge nodes *per se*. Instead, some dedicated systems such as Azure

IoT Edge [4], and the AWS IoT Greengrass [5] can be used to offload the ML computations on the Edge nodes.

ML at the Edge. Several studies [14, 21] identified the main vectors driving a shift towards using resources at the Edge of the network, near the data sources, building what is called the *Edge intelligence* [32]: ML training and ML inference at the Edge.

Fast inference was initially achieved on single Edge devices. Instead of transmitting raw contextual data in the network, [24] only transmits the inferred knowledge using incremental ML on the Edge nodes. DeepDecision [25] and MCDNN [19] take the offloading decision based on constraints such as the size of input data, the model to be executed, the tradeoffs between the inference accuracy, network latency and bandwidth. Most of these approaches are application specific. When applied to different contexts, the overhead of the adaptation increases the inference latency particularly on high-dimensional input data and on mobile devices [21].

Edge-enhanced architectures Distributed deep learning has motivated the use of computing hierarchies by splitting neural network [31] or using sparse updates [30] to aggressively reduce communication costs. However, these works are either not resilient/sensitive to topology changes [33] or rely on unsuitable resource management for service delivery [12]. Previous work from authors have explored Edge-enhanced infrastructures for Urgent Science and large-scale application[10].

Conclusion Different from these works, our architecture for ML executions on the Continuum: *(i)* is general enough to be adapted to any application, in any context, without loss of efficiency, and *(ii)* holistically considers the deployment issues of ML on Edge/Cloud nodes (i.e., communication and computation constraints).

8 CONCLUSION AND FUTURE WORK

In this paper we present MDSC, an approach for modeling distributed stream-based applications on Edge-to-Cloud continuum infrastructures. It uses interconnected layers and units to model the core components of an application, such as libraries, data processing and management tools. We illustrate the versatility of MDSC by modeling of a complex real-life ML application, that was deployed on an Edge-to-Cloud infrastructure emulated atop Grid'5000.

As future work, we are looking into developing the description of the helper units to reflect their internal configuration and libraries, which can have an impact on the application performance. Another interesting research direction is the design of scheduling algorithms that help designers decide where to deploy processing units on the different available layers of a hybrid infrastructure.

REFERENCES

[1] [n.d.]. Incorporated Research Institutions for Seismology. https://www.iris.edu/. [Online; accessed 25-May-2020].

[2] 2015. Google Cloud IoT Edge. https://cloud.google.com/iot-edge/. [Online; accessed 15-April-2020].

[3] 2019. Apache FlinkML. https://ci.apache.org/projects/flink/flink-docs-release-1.4/dev/libs/ml/. [Online; accessed 15-April-2020].

[4] 2020. AI Toolkit for Azure IoT Edge. https://github.com/Azure/ai-toolkit-iot-edge. [Online; accessed 15-April-2020].

[5] 2020. AWS IoT Greengrass. https://aws.amazon.com/greengrass/. [Online; accessed 15-April-2020].

[6] 2020. LIBSVM – A Library for Support Vector Machines. https://www.csie.ntu.edu.tw/~cjlin/libsvm/. [Online; accessed 25-May-2020].

[7] 2020. Microsoft Azure IoT Edge. https://azure.microsoft.com/en-us/services/iot-edge/. [Online; accessed 15-April-2020].

[8] M. Abadi and al. 2016. TensorFlow: A system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. 265–283. https://www.usenix.org/system/files/conference/osdi16/osdi16-abadi.pdf

[9] D Balouek-Thomert and al. 2019. Towards a computing continuum: Enabling edge-to-cloud integration for data-driven workflows. *The International Journal of High Performance Computing Applications* 33, 6 (Nov. 2019), 1159–1174. https://doi.org/10.1177/1094342019877383

[10] Daniel Balouek-Thomert and al. 2020. Harnessing the Computing Continuum for Urgent Science. *SIGMETRICS Perform. Eval. Rev.* 48, 2 (Nov. 2020), 41–46. https://doi.org/10.1145/3439602.3439618

[11] P Beckman and al. 2020. Harnessing the computing continuum for programming our world. *Fog Computing: Theory and Practice* (2020), 215–230.

[12] Hyunseok C. and al. 2014. Bringing the cloud to the edge. In *2014 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE, 346–351.

[13] P. Carbone and al. 2015. Apache Flink: Stream and Batch Processing in a Single Engine. *IEEE Data Engineering Bulletin* 38 (01 2015).

[14] J. Chen and X. Ran. 2019. Deep Learning With Edge Computing: A Review. *Proc. IEEE* 107, 8 (2019), 1655–1674.

[15] F. Desprez and al. 2013. Adding Virtualization Capabilities to the Grid'5000 Testbed. In *Cloud Computing and Services Science*. Communications in Computer and Information Science, Vol. 367. Springer International Publishing, 3–20. https://doi.org/10.1007/978-3-319-04519-1_1

[16] K. Fauvel and al. 2020. A Distributed Multi-Sensor Machine Learning Approach to Earthquake Early Warning. In *Proceedings of the Thirty-Fourth AAAI Conference on Artificial Intelligence*.

[17] E. Gibert Renart and al. 2019. Distributed Operator Placement for IoT Data Analytics Across Edge and Cloud Resources. In *CCGrid 2019 - 19th Annual IEEE/ACM International Symposium in Cluster, Cloud, and Grid Computing*. Larnaca, Cyprus, 1–10. https://doi.org/10.1109/CCGRID.2019.00060

[18] C. Gupta and al. 2017. Protonn: Compressed and accurate knn for resource-scarce devices. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 1331–1340.

[19] S. Han and al. 2016. MCDNN: An Approximation-Based Execution Framework for Deep Stream Processing Under Resource Constraints. In *14th Annual International Conference on Mobile Systems, Applications, and Services*. 123–136.

[20] X. Meng and al. 2016. MLlib: Machine Learning in Apache Spark. *J. Mach. Learn. Res.* 17, 1 (2016), 1235–1241.

[21] M. G. Sarwar Murshed and al. 2019. Machine Learning at the Network Edge: A Survey. *arXiv* (07 2019).

[22] F. Pedregosa and al. 2011. Scikit-learn: Machine learning in Python. *Journal of machine learning research* 12, Oct (2011), 2825–2830.

[23] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-Learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.

[24] K. Portelli and C. Anagnostopoulos. [n.d.]. Leveraging Edge Computing through Collaborative Machine Learning. In *2017 5th International Conference on Future Internet of Things and Cloud Workshops*. 164–169,.

[25] X. Ran and al. 2018. DeepDecision: A Mobile Deep Learning Framework for Edge Video Analytics. In *IEEE INFOCOM 2018 - Conference on Computer Communications*. 1421–1429.

[26] Daniel Rosendo and al. 2020. E2Clab: Exploring the Computing Continuum through Repeatable, Replicable and Reproducible Edge-to-Cloud Experiments. In *Cluster 2020 - IEEE International Conference on Cluster Computing*. Kobe, Japan, 1–11. https://doi.org/10.1109/CLUSTER49012.2020.00028

[27] P. Schäfer and U. Leser. 2017. Multivariate Time Series Classification with WEASEL+MUSE. *ArXiv* (2017).

[28] P. Schäfer and U. Leser. Last accessed on May 2020. WEASEL MUSE Artifact. https://github.com/patrickzib/SFA.

[29] P. Silva and al. 2019. Towards a Methodology for Benchmarking Edge Processing Frameworks. In *2019 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. 904–907.

[30] Zeyi Tao and Q. Li. 2018. eSGD: Communication Efficient Distributed Deep Learning on the Edge. In *HotEdge*.

[31] S. Teerapittayanon, B. McDanel, and H. T. Kung. 2017. Distributed DNN over the Cloud, the Edge and End Devices. *CoRR* abs/1709.01921 (2017). arXiv:1709.01921

[32] X. Wang and al. 2020. Convergence of Edge Computing and Deep Learning: A Comprehensive Survey. *IEEE Communications Surveys Tutorials* (2020).

[33] Z. Wen and al. 2018. ApproxIoT: Approximate Analytics for Edge Computing. In *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*. 411–421. https://doi.org/10.1109/ICDCS.2018.00048

[34] M. Zaharia and al. 2016. Apache Spark: A Unified Engine for Big Data Processing. *Commun. ACM* 59, 11 (oct 2016), 56–65.