Enabling microservices management for Deep Learning applications across the Edge-Cloud Continuum

Zeina Houmani, Daniel Balouek-Thomert, Eddy Caron, Manish Parashar

Enabling microservices management for Deep Learning applications across the Edge-Cloud Continuum

Zeina Houmani*, Daniel Balouek-Thomert ‡, Eddy Caron*, Manish Parashar‡
*Inria Avalon team, LIP Laboratory, UMR CNRS - ENS de Lyon, University of Lyon, France
‡Scientific Computing Imaging (SCI) Institute, University of Utah, UT, USA
Corresponding author: zeina.houmani@ens-lyon.fr

***Abstract*—Deep Learning has shifted the focus of traditional batch workflows to data-driven feature engineering on streaming data. In particular, the execution of Deep Learning workflows presents expectations of near-real-time results with user-defined acceptable accuracy. Meeting the objectives of such applications across heterogeneous resources located at the edge of the network, the core, and in-between requires managing trade-offs between the accuracy and the urgency of the results. However, current data analysis rarely manages the entire Deep Learning pipeline along the data path, making it complex for developers to implement strategies in real-world deployments. Driven by an object detection use case, this paper presents an architecture for time-critical Deep Learning workflows by providing a data-driven scheduling approach to distribute the pipeline across Edge to Cloud resources. Furthermore, it adopts a data management strategy that reduces the resolution of incoming data when potential trade-off optimizations are available. We illustrate the system's viability through a performance evaluation of the object detection use case on the Grid'5000 testbed. We demonstrate that in a multi-user scenario, with a standard frame rate of 25 frames per second, the system speed-up data analysis up to 54.4% compared to a Cloud-only-based scenario with an analysis accuracy higher than a fixed threshold.**

***Keywords*-Cloud computing, Edge computing, Microservices, Task allocation, Real-time processing, Computing Continuum, Deep Learning.**

I. Introduction

Deep Learning has gained huge momentum in the industry over recent years with a growing market estimated at 44.3 Billion USD by 2027[1]. Deep Learning (DL) applications present a growing potential to extract knowledge from the analysis of streaming data, with applications in numerous domains including computer vision [1], speech recognition [2], and COVID-19 research [3].

Deep Learning applications, implemented as distributed analytics, are currently limited by Cloud-centric models that suffer crippling latency limitations when the amount and frequency of data increases. The computational ecosystem that supports these analytics has become highly heterogeneous and geographically distributed, bringing significant challenges associated with the complexity and sustainability of performing decision-making on sensor data [4], [5]. In particular, many Deep Learning applications require important decision-making to be delivered in a timely manner [6], requiring a novel design that enables trade-offs between the time and the quality of analysis.

Meeting the application's objectives when dealing with multiple data sources and resources of heterogeneous capabilities highlights the need for resource and data management solutions. Resource management aims at task allocation strategies and collaborative infrastructure designs [7], [8], while data management approaches often consist of customizing Deep Learning models to suit the resource-constrained systems while addressing the trade-off between QoS metrics [9], [10]. Existing work tends to approach these two aspects independently and rarely manages the entire Deep Learning pipeline, resulting in inefficiencies between the design and the deployment of data-driven applications.

Driven by an object detection use case, this paper presents a system for time-sensitive DL workflows. The architecture relies on heterogeneous resources close to data sources, in the core and along the data path. This continuum allows extracting insights from data at early stages, which helps in managing data-driven applications. By combining resource and data management solutions, the system aims at managing trade-offs between analysis makespan and accuracy of results to meet application performance. Furthermore, it offers means to developers to automatically distribute Deep Learning workflows across the Edge-to-Cloud continuum.

This paper makes the following contributions:

- A data management strategy based on data quality adaptation. It adjusts the resolution of data sources to manage latency-accuracy trade-offs.
- A data-driven workflow scheduling approach to distribute DL tasks on the computing continuum.

The rest of the paper is organized as follows. Section II presents the use case driving this work. Section III briefly discusses the related work. Section IV shows the system architecture, performance models, and system utility function. The data adaptation approach is proposed in Section V. Section VI presents the strategies for distributing DL tasks on the continuum. Section VII describes the experimental setup and results. Finally, Section VIII concludes this work.

[1]Global deep learning industry (2020), https://www.reportlinker.com/p05798338/Global-Deep-Learning-Industry.html

II. Motivating use case

For data scientists, the data analysis process consists of three main stages. **Pre-processing stage:** It is responsible for preparing incoming data for analysis. Additionally, this stage allows the characterization of incoming data. The extracted characteristics, such as data resolution, format, and size, contribute to a better selection of the analysis pipeline. **Analysis stage:** It corresponds to a set of Deep Learning models responsible for extracting features, detecting and recognizing objects in the incoming prepared data. These models can be of different types, such as *You Only Look Once* (YOLO) [11] and *Faster Region-based Convolutional Neural Networks* (Faster RCNN) [12]. These real-time detectors are based on convolutional networks that predict object boundaries and object scores at each detection. **Post-processing stage:** It processes the knowledge extracted from the previous stage. It can evaluate the resulting knowledge and make decisions whether to ignore it, visualize it, store it or urgently notify the end users about it.

Current data analysis systems only consider Deep Learning applications as a set of learning models making the use of management strategies complex in real-world deployments. Managing trade-offs for the entire pipeline on current heterogeneous infrastructures is challenging: ① tasks in each stage demand different computing requirements. Assigning resources to these tasks must consider their roles in the analysis. ② Deep Learning application deals concurrently with a high load of different resolutions. Assigning pipelines to data sources has an impact on the system performance as each data quality demands different computing needs.

This work is driven by an object detection use case (Figure 1). It represents a time-sensitive Deep Learning application that identifies and locates objects in an image or video (sequence of images). First, `Resize` service receives incoming frames and modifies their sizes to suit the input data size of the following task. This service belongs to pre-processing stage. Object detection task receives these frames and detects existing objects using two possible YOLOv4 models: `YOLOv4-416` for $416p$ frames and `YOLOv4-512` for $512p$ frames. These models are a part of the analysis stage. Finally, the frames and analysis results are sent to a `Draw` service responsible for marking the detected and identified objects on the frames and save them locally. This task belongs to the post-processing stage.

The execution of time-sensitive applications presents expectations of near-real-time latency with user-defined acceptable accuracy. In this use case, the analysis accuracy corresponds to the number of detected faces and labels assigned. Meeting these objectives motivates the need for a system supporting DL pipelines with data and resource management solutions. It relies on the microservices paradigm to deploy the pipeline on the distributed resources. This paradigm provides the capacity to monitor each task individually to detect events and maintain system performance. In this work, the terms tasks and microservices are used interchangeably. The proposed system is presented in Section IV.

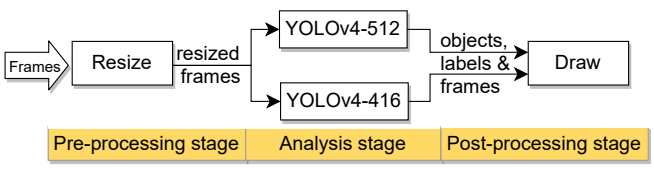

Figure 1: Workflow of an object detection Deep Learning application showing the stages, dataflow, and tasks.

III. Related work

A. Edge-enhanced Data Analytic Systems

Cloud-centric systems suffer crippling latency limitations when the amount and frequency of data increase [13]. Therefore, Edge computing [14] has widely emerged to complement and extend the Cloud [15]. Model optimization and Machine Learning inference are currently the main vectors driving the use of resources at the edge [16]. Many works attempted to adopt Edge-based infrastructure designs to optimize analytics systems, where some systems focus on reducing energy consumption [17], guaranteeing deadlines [18] and meeting real-time requirements [19]. In [19], Kong *et al.* ensure a real-time analysis in consideration of accuracy by adopting an Edge-based system. However, the system only considers the inference time of Deep Learning models and not the entire analysis pipeline. In addition, due to limited resources, the proposed system will fall short in dealing with several high-resolution video streams.

Cloudlet resources are widely used in Edge Computing. They are characterized by a high bandwidth network relative to the Edge and offer lower latency than the Cloud [14]. Gigasight [8] reduces latency and bandwidth from Edge/Cloudlet to Cloud by running analysis on the Cloudlet resources and sending only the results to the Cloud. In [18], Zamani *et al.* adopt a federated resources model that exposes in-transit and edge capabilities to participant sites.

The Microservices paradigm has gained great popularity in recent years, exploring the benefits of modular, self-contained components for highly dynamic applications [20]. This paradigm serves large-scale data analytics systems. In [21], Khoonsari *et al.* showed that microservices allow for efficient horizontal scaling of analyzes on multiple computational nodes, enabling the processing of large datasets. In addition, Taneja *et al.* adopted microservices in [22] due to multiple reasons such as their deployability on hybrid Fog-Cloud environments and their technological independence.

B. Scheduling Strategies For Deep Learning

Scheduling approaches for Deep Learning can be classified into three main categories. First, *job scheduling* techniques are designed to schedule prediction and training jobs on the Deep Learning cluster workers [23]. Second, *single*

task scheduling techniques for Deep Learning models with a guarantee of specific performance targets [24]. Lastly, *workflow scheduling* techniques are designed to distribute the entire application on the available resources [13]. This category is not yet well discussed in the literature for Deep Learning applications. The contribution of this work belongs to the last category.

The functional partitioning of Deep Learning workflow and its distribution across Edge-Cloud resources has been considered in [13]. The proposed scheduling approach is goal-driven; the scheduling decisions made are motivated by the system's goal of satisfying the real-time requirements. These approaches do not consider the difference in the computing and network requirements of Deep Learning tasks, nor the dynamic impact of incoming data on the application performance. The data-driven scheduling approach proposed in this work examines task heterogeneity and makes allocation based on the tasks' categories and their dependencies.

C. Configuration Adaptation For Edge-based Systems

Analyzing concurrently multiple data streams with limited resources forces resource-quality trade-offs [25]–[27]. As the incoming data are processed concurrently, resources available to each data stream are often unknown. Online/offline configurations adaptation is currently a promising solution to address the issue of limited resources [9], [26], [28]–[30].

In [28], Wang *et al.* adopt an offline configuration adaptation and bandwidth allocation strategies to address the issue of limited resources between IoT devices and edge nodes. Similar to the approach presented in this work, the adaptation is triggered periodically. Systems in [9], [29] adopt an online configuration adaptation algorithms for video analytics in Edge computing. The configurations targeted are frame rate and resolution. However, these systems only focus on the performance of the analysis stage and not on a complete workflow. Additionally, they only target Edge-based video analytics applications. In [30], they present an Edge Network Orchestrator for Mobile Augmented Reality (MAR) systems. It boosts the performance of an Edge-based MAR system by optimizing the edge server assignment and video frame resolution selection for MAR users. In this work, we argue that adopting a data quality adaptation strategy for Microservice-based Deep Learning applications built on a 3-tiers environment will optimize the analysis latency with respect to accuracy constraints.

IV. Model and Architecture

A global system overview is presented in Figure 2. The system's architecture consists of three levels. Each has a set of management services following the microservice paradigm and communicates via APIs.

Workflow management level. It is responsible for categorizing and scheduling the submitted pipeline (P) across the resources (see Section VI). The pipeline is composed of three stages: Pre-processing (Pr), Analysis (A) and Post-processing (Po) stage. Each stage s is composed of a set of microservices $M = \{m_i\}$. The analysis stage has Z learning models. These models can be of different types (Faster RCNN, YOLO, etc.) or the same type but support different input quality (YOLO416, YOLO512, etc.). Let $D = \{d1, d2, ..., dZ\}$ represent the set of models and $Q = \{q1, q2, ..., qL\}$ the set of data quality supported.

Infrastructure level. Its design is based on the Edge-to-Cloud computing continuum. The set of resources is given by $R = \{r_k\}$ which represents the set of Edge (E), Fog (F), and Cloud (C) nodes. In this work, the terms Cloudlet and Fog refer to the same type of resources. Cloudlet resources provide computational capabilities greater than Edge resources and less than Cloud resources. The capacity of the Cloudlet-Cloud network link is a thousand times higher than the capacity of the Edge-Cloudlet link. The system supports multiple data sources located at the Edge, each generating data in the default frame rate $(fs = 25fps)$ and resolution $(fr = 512 \times 512)$. The set of data sources in the system are denoted by $U = \{u_1, u_2, ..., u_k\}$. A frame generated by a data source is considered as a job J to be processed by the application. Let N be the total number of jobs generated by a data source in a time slot t.

Data management level. It selects the data quality distribution for data sources providing a system makespan and accuracy that meet the developer's needs. This level consists of three components. First, a discovery component responsible for the pipeline discovery. Second, performance models that estimate the makespan and accuracy of the assigned pipeline. Third, a data quality adaptation component that selects for data sources the qualities providing the best-estimated performance.

The remainder of this section presents the analytical models of the end-to-end latency and accuracy of K data sources, as well as the formulation of the system's objective.

A. End-to-end Latency Model

The end-to-end latency of a job J from a data source u corresponds to the time taken to complete its analysis pipeline. It is presented as follows:

$$T_J^u = \mu \cdot T_{reduce} + T_{Pr} + T_A + T_{Po} \quad (1)$$

T_{reduce} refers to the time required to reduce the job's quality before the processing starts. μ is a binary that indicates whether a data adaptation was needed for u or not. Details about data adaptation will be presented in Section V. T_{Pr}, T_A, and T_{Po} correspond to the time spent in the preprocessing, analysis, and postprocessing stages, respectively.

Total time cost (T_s) of a processing stage is the sum of the response time of its microservices. It is presented as $T_s = \sum_{i=1}^{n} RT(m_i)$ where $RT(m_i) = \alpha \cdot T_D + Trans(m_j, m_i) + T_E$. $RT(m_i)$ represents the response time of microservice m_i. α is a binary variable that indicates

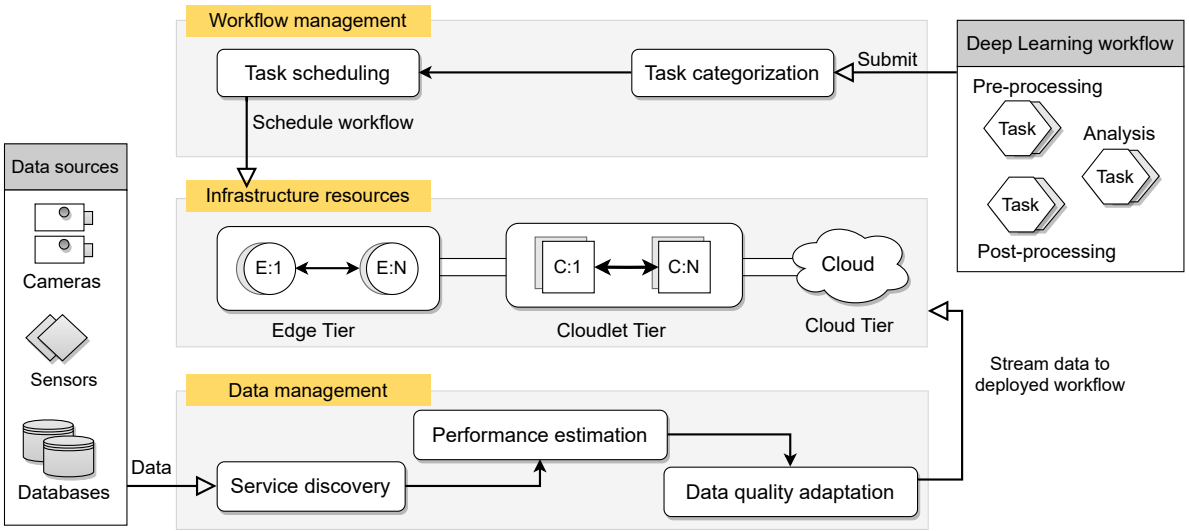

Figure 2: Global overview of the system design. It consists of workflow management, infrastructure, and data management level.

whether a microservice discovery happened or not while processing a job. T_D represents the discovery time of the microservice. $Trans(m_j, m_i)$ represents the time needed to transfer the data input of microservice m_i from its predecessor m_j. Let r_j and r_i be the resources where m_j and m_i are deployed respectively. For clarity, $Trans(m_j, m_i)$ is replaced by $Trans(r_j, r_i)$. T_E corresponds to the execution time of the microservice.

The transfer time of the data is formulated as:

$$Trans(r_j, r_i) = \begin{cases} \frac{S}{w(r_j, F)} + \frac{S}{w(F, r_i)} & \text{if } r_j \in E,\ r_i \in C \\ 0 & \text{if } r_i = r_j \\ \frac{S}{w(r_j, r_i)} & \text{otherwise,} \end{cases} \tag{2}$$

S refers to the data size (in Mbits) to be sent over the network. It is given by $S = \gamma fr^2$. γ is the number of bits required to represent the information carried by one pixel. $w(producer, consumer)$ refers to the bandwidth (Mbits/s) of the network link between the data producer and consumer. If they were deployed on resources $\{r_i, r_j\} \in \{E, C\}$, the transferred data must pass through the Cloudlet tier.

The execution time of a microservice is presented as $T_E = Load/C$ where $Load$ refers to the data to be analyzed by m_i and C to the number of data that can be processed per second. In stage A of the pipeline, the throughput depends on the chosen Deep Learning models and the incoming data quality of the data source. As experimentally proved in [28] and [29], with high data quality, the models provide a slower analysis speed than that with low quality. In addition, it [31] showed that with the same data, some models perform faster than others.

Thus, the average end-to-end latency of K data sources in a time slot t is presented as follows:

$$T_t = \frac{1}{K} \sum_{i=1}^{K} \left(\frac{1}{N} \sum_{j=1}^{N} T_j^i \right) \tag{3}$$

B. Analysis Accuracy Model

Analysis accuracy of DL models corresponds to the metric F1 score. It is a weighted average of the *Precision* and *Recall* evaluation metrics. To identify their True Positives (TP) and False Positives (FP), the *Intersection over Union (IoU)* metric is used. *IoU* is a number from 0 to 1 that specifies the amount of overlap between the predicted and ground truth (i.e., actual) bounding box. If $IoU \geq 0.5$ and the label is correct, the analysis is TP. However, the analysis is FP if the label was false or the $IoU < 0.5$ and the label is true.

It has been experimentally observed in [28], [29], [31] that the data quality and the chosen DL model impact the accuracy of the results. As showed in [28], [29], the relationship between accuracy and data quality is formulated as a concave exponential function of three coefficients $\mathcal{E}(fr, d) = \alpha 1_d - \alpha 2_d \times e^{-fr/\alpha 3_d}$. It reflects that a higher quality produces a better analytics accuracy, and the analytics accuracy gain decreases at a high quality. In this work, $\{\alpha 1, \alpha 2, \alpha 3\}$ are constant coefficients of a DL model d.

Due to the data-driven discovery mechanism, changing the data quality by the data management strategy during a time slot will cause a change in the selected Deep Learning model. Let ϕ_J^i and β_J^x be two binary variables that indicate whether model d_i and data quality q_x are selected for data source u_k to process its job J. So, $d_J'^k = \sum_{y=1}^{Z} \phi_{J,k}^y d_y$ is the Deep Learning model of job J from data source u_k and $q_J'^k = \sum_{x=1}^{L} \beta_{J,k}^x q_x$ is its data quality with $\sum_{y=1}^{Z} \phi_{J,k}^y = 1$ and $\sum_{x=1}^{L} \beta_{J,k}^x = 1$. Hence, the average accuracy of a data source u_k in time slot t is $\frac{1}{N} \sum_{j=1}^{N} \mathcal{E}(q_j'^k, d_j'^k)$. During

a time slot, $\sum_{y=1}^{Z} \phi_k^y$ and $\sum_{x=1}^{L} \beta_k^x$ can be greater than 1, which indicates that a data source can use multiple Deep Learning models and have multiple data qualities. The average accuracy of K data sources in a time slot t is presented as follows:

$$a_t = \frac{1}{K} \sum_{i=1}^{K} \left(\frac{1}{N} \sum_{j=1}^{N} \mathcal{E}(q_j'^i, d_j'^i) \right) \tag{4}$$

C. System Objective

This work aims to reduce the data analysis latency of Deep Learning applications under a long-term accuracy constraint. To achieve this objective, a latency-accuracy trade-off utility function is needed. It is formulated as $U_{u,t} = a_{u,t} - \Theta T_{u,t}$ where Θ trades off between the latency cost and accuracy. The total utility for K data sources is presented as:

$$U_t = \frac{1}{K} \sum_{i=1}^{K} U_{i,t} \tag{5}$$

Maximizing this utility during runtime, with respect to the accuracy constraint, will increase the system efficiency. As in [28], the long-term utility augmentation will become limited. So, the system goal is formulated as follows:

$$\begin{aligned} Goal: \quad & \max_{\beta,\phi} \quad \lim_{t\to+\infty} \frac{1}{T} \sum_{t=1}^{T} (U_t) \\ & \text{subject to} \quad \lim_{t\to+\infty} \frac{1}{T} \sum_{t=1}^{T} a_t \geq a_{min} \end{aligned} \tag{6}$$

The accuracy constraint ensures that the tradeoff is only possible if the long-term average accuracy exceeds the minimum threshold a_{min} (by default, $a_{min} = 50\%$).

The system aims to adapt the quality of incoming data to optimize this utility (see Section V).

V. Data Quality Adaptation Strategy

The system contains a set of distributed and limited resources of different computing and network capacity. Analyzing data generated by multiple data sources on these resources requires adopting a latency-accuracy tradeoff solution. This section presents a data quality adaptation strategy for DL applications. This strategy is responsible for specifying the distribution of data qualities on the existing data sources. It estimates whether the system can handle the original data qualities of all data sources or a quality reduction is required. Reducing data quality can negatively affect the analysis accuracy. So, this strategy controls the quality of generated data while guaranteeing a system accuracy higher than a fixed threshold a_{min}. The microservices responsible for applying this strategy are deployed on the Edge.

Deep Learning applications have dynamic data content. Therefore, adapting data quality once during analysis is inefficient as the performance varies depending on the content. For that reason, the adaptation strategy is triggered periodically and when new data sources join the system.

Let $B = \{b_1, ..., b_m\}$ be the set of possible quality distributions among data sources. Each distribution b_i is formulated as $\{\beta \cdot q_l \mid 1 \leq \beta \leq k \text{ and } q_l \in Q\}$, where β represents the number of data sources having the data quality q_l and k is the total number of data sources in the system. q_l might corresponds to their original or reduced data qualities. In the object detection use case $Q = \{512p, 416p\}$. For n=3, a possible data quality distribution can be $b_1 = \{(2) \cdot Q_{512p}, (1) \cdot Q_{416p}\}$.

The proposed strategy selects from B the distribution with the fastest analysis latency and accuracy that does not fall below the a_{min} threshold. However, in specific use cases, the selected data quality distribution may not be the fastest: The system compromises between latency and accuracy in case there is another distribution that provides a latency gain less than $10ms$ but an accuracy loss greater than or equal to 20%. The strategy is presented in Algorithm 1. In

Algorithm 1: Select the quality distribution with a latency-accuracy trade-off.

Result: quality distribution with the optimal trade-off.
begin
1. initialization;
2. **for** *b* ***in*** *B* **do**
3. L ← getEstimatedLatency(b) ; // model (3) in IV-B
4. A ← getEstimatedAccuracy(b) ; // model (4) in IV-B
5. **if** $A \geq a_{min}$ **then**
6. add({b, L, A}, list);
7. **if** *isEmpty(list) == TRUE* **then**
8. return $\emptyset$
9. best ← getMinLatency(list);
10. **for** *x* ***in*** *list* **do**
11. **if** $\triangle(x[L], best[L]) < 10$ ***and*** $\triangle(x[A], best[A]) \geq 20\%$ **then**
12. best ← x;
13. return(best);

steps 2-6, it calculates the estimated latency and accuracy of each possible data quality distribution. Then, in step 5, it filters those with unacceptable accuracy. If no configuration provides acceptable accuracy, the data source can't join the system at that time period (steps 7 & 8). Otherwise, the preferred data distribution among the acceptable configurations is the one with the fastest analysis (step 9). In steps 10-12, it checks if there is another distribution that matches the use case presented above. If so, it will be selected as the preferred quality distribution in the system.

After selecting the data quality distribution, the system randomly maps qualities to data sources. This mapping switches periodically between data sources.

VI. Data-driven workflow scheduling and discovery approaches

Several challenges exist when scheduling the Deep Learning pipeline on the continuum: ① tasks in the pipeline are heterogeneous and have different resource requirements; ② the system has limited resources; ③ system resources have different computing and network capacity; ④ in a continuum, resources can become unavailable. For challenges ① and ③, this work adopts a scheduling approach based on a task categorization. Its purpose is to distribute Deep Learning workflow across the edge of the network, the core, and along the data path with regards to the microservices' functionalities and data inputs. For challenge ②, the system adopts a requirement adjustment algorithm to efficiently using the limited resources. The 4th scheduling challenge is beyond the scope of this paper. Each component in the scheduling approach is deployed as microservice and located on the Edge tier. The scheduling approach is triggered when a new workflow is submitted. When triggered, the categorization microservice receives the workflow description. After examining each task, It sends the description with the categorization results to the reservation microservice. The latter allocates the resources and triggers the scheduling microservice. If the available resources are not enough, it triggers the adjustment microservice.

Assigning a data source to a scheduled pipeline matching the resolution of generated data is done via a data-driven microservice discovery mechanism running on the Edge.

A. Tasks Categorization

Tasks in Deep Learning pipelines demand different CPU, memory, storage, and bandwidth requirements. Based on our observations in deploying and running DL applications, tasks can be classified into three categories: Non-Intensive (NI), Low-Intensive (LI), and High-Intensive (HI) tasks.

NI tasks refer to the pre-processing and post-processing tasks that do not manage their own database. Database microservices are considered HI due to their storage demand. In our application use case, `Resize` microservice is considered NI and `Draw` microservice as HI. Classic learning models (i.e., shallow models) are LI. Concerning Deep Learning models, their characteristics have an impact on their performance, such as the number and type of parameters and neural network layers. In this work, we only consider the data resolution to categorize Deep Learning models. For each learning model in the analysis stage (YOLO, Faster RCNN, etc.), the implementation with the lowest resolution is considered LI. For example, in the object detection use case, `YOLOv4-416` is considered as LI and `YOLOv4-512` as HI.

In practice, users need to specify in the description of the submitted workflow the type of each task (pre-processing, shallow, etc.). The system will then automatically assign each type to a category, as mentioned above.

B. Resource Reservation And Scheduling Algorithms

A naive approach for assigning tasks to resources is to explore all the possibilities in a brute-force manner which creates a search space of exponential complexity. This work adopts the following pruning techniques to minimize the search space. First, all tasks within the same category have the same resource assignment. Second, intensive tasks can only be deployed on the Fog and Cloud. HI tasks have a priority to be assigned to the Cloud and LI to the Fog.

The scheduling approach consists of two parts. The first part aims to reserve resources for intensive tasks. It gives HI tasks a higher placement priority than LI tasks. The resource reservation for HI tasks is presented in Algorithm 2. This algorithm takes as input the number of HI tasks in the submitted workflow. In step 1 and 2, it retrieves the available resources in Cloud and Cloudlet tiers, respectively. In step 3, it counts the number of HI tasks that can be placed on the Cloud. This depends on the capacity of the Cloud tier and the fixed requirements of the HI tasks. Steps 4-16 check whether the computing requirements of HI tasks can be fully guaranteed at the Cloud or they must be distributed across Cloudlet-Cloud tiers. In addition, they decide whether the available resources can handle the entire computing requirements of HI tasks or a "requirements adjustment solution" is needed. For the first case (steps 4-6), it checks if the capacity of the Cloud tier is greater than the requirements of HI tasks. If so, it reserves all required resources on the Cloud. If not (steps 7-16), it checks whether Fog resources

Algorithm 2: Resource reservation for HI tasks.

Data: countHI
Result: List of resources reserved for HI microservices

```
List Reserve_HI(countHI)
    begin
1       capCloud ← getFreeResources('cloud');
2       capFog ← getFreeResources('fog');
3       cloudHI ← countCloudHI(capCloud);
4       if cloudHI ≥ countHI then
5           entry ← reserve(countHI, 'cloud');
6           updateReservedList(entry);
7       else
8           remainHI ← countHI - cloudHI;
9           fogHI ← countFogHI(capFog);
10          if fogHI ≥ remainHI then
11              entry ← reserve(cloudHI, 'cloud');
12              updateReservedList(entry);
13              entry ← reserve(remainHI, 'fog');
14              updateReservedList(entry);
15          else
16              Adjust_res('HI');
17      return(getReservedList());
```

can handle the requirements of the remaining tasks. If so, the reserved resources for HI tasks will be distributed across Cloudlet-Cloud tiers. However, if the resources available in the Cloudlet are also not enough, it triggers Algorithm 4, which will be presented in Section VI-C.

After reserving the resources for HI tasks, the system repeats the same logic to reserve those for LI tasks. The system checks first whether their computing requirements can be guaranteed at the Fog before checking the Cloud.

The second part of this scheduling approach is to distribute the pipeline on the continuum (see Algorithm 3). This algorithm takes the list of resources reserved for intensive tasks and the workflow. For each task, it checks whether they are intensive or not. If so, it searches for the resources reserved for its category (steps 4 and 5). However, if not, it looks for the remaining free resources (steps 6 and 7). As NI tasks do not require a lot of computing power, they can be easily placed on system resources without prior reservation. Among discovered resources for each task, it selects those located near its predecessor (step 8). A predecessor of a task corresponds to its preceding task in the analysis pipeline or a data source in case it is the entry task. If several resources are located on the same infrastructure level, the selection of the resource is random. After the resource is selected, it deploys the task and makes it ready for production (step 9).

Algorithm 3: Scheduling tasks on the continuum.

Data: listsIntensive, workflow
Result: Mapping DL pipeline to system resources

```
Void Scheduling(listsIntensive, workflow)
    begin
1       for task in workflow do
2           predecessor ← getPredecessor(task);
3           category ← getCategory(task);
4           if category = 'LI' OR category = 'HI'
              then
5               list ← getReserved(category,
                  listsIntensive);
6           else
7               list ← getFreeResources(category);
8           res ← selectResource(list, predecessor);
9           deployTask(task, res);
```

C. Requirements Adjustment Algorithm

As Algorithm 2 has shown, it is possible that the system cannot handle the computing requirements of intensive tasks. Therefore, Algorithm 4 is used to reduce the resource requirements of intensive tasks when a full guarantee of required resources is not possible. It maximizes the use of system resources while ensuring a minimum threshold equal to 50% of their fixed requirements. This algorithm is only triggered by Algorithm 2 if a new workflow is submitted.

Algorithm 4 takes the category of the task to be placed and the list of resources reserved for intensive tasks. In steps 2 and 3, it gets the remaining free resources on the Cloudlet-Cloud tiers and selects the one with the maximum remaining capacity. If the remaining capacity selected is greater than or equal to the minimum threshold, the resource is reserved (steps 3-5). However, if not, the algorithm attempts to reach the minimum threshold by adjusting the computing capacity of the other reservations on the same selected resource (steps 6-13). In step 7, it gets the remaining capacity needed to reach the minimum threshold. The reservations on the selected resource that can handle a resource adjustment are those that remain above the minimum threshold even if their computing capacity is reduced (step 8). In steps 9 and 10, it reduces the remaining capacity needed evenly from the reservation list. After adjustment, the list of reserved resources is updated (steps 11-13).

Algorithm 4: Resource adjustment for tasks.

Data: listLI, listHI, category
Result: Adjust required resources of intensive tasks

```
Void Adjust_res(listLI, listHI, category)
    begin
1       listFree ← getFreeResources();
2       res ← getMAX(listFree, category);
3       if res ≥ 50% × requiredResources then
4           entry ← reserve(res, category);
5           updateReservedList(category, entry);
6       else
7           remain ← getRemain(res);
8           listReservations←checkReservations(remain,
              listLI, listHI);
9           part ← (remain / size(listReservations));
10          newList ← Reduce(part,
              listReservations);
11          updateReservedList(newList);
12          entry ← reserve(res, remain);
13          updateReservedList(entry);
```

D. Data-Driven Microservices Discovery

Current discovery mechanisms are goal-based, designed to achieve the overall goal of the system. However, using these mechanisms in the proposed system is not efficient due to the following reasons: ① producers and consumers of incoming data can be designed by different entities; ② static connections between tasks in DL applications are not efficient as new tasks can be added, and others removed; ③ analysis tasks are designed for different data resolutions and offer different latency and accuracy guarantees.

In this work, a data-driven discovery mechanism is used to overcome these limitations. It consists of discovering available microservices depending on the data characteristics

such as resolution, type, and format. Details about this mechanism are presented by Z. Houmani *et al.* in [32]. In the proposed system, the discovery assigns data sources to pipelines supporting their data resolution. During the adaptation process, the system triggers the discovery for each supported resolution to select the distribution providing the maximum utility.

VII. Experimental setup and results

The evaluation aims to show the impact of the proposed system on the performance of the object detection use case (Figure 1) when dealing with high load. The use case is representative of the general Deep Learning problem as it deals with tasks of different categories and requires leveraging limited and heterogeneous resources to achieve real-time performance. This section presents the evaluation methodology, the results and then discusses some takeaways.

A. Experimental Setup

The system evaluation is performed on the large-scale platform *Grid'5000* [33]. It represents a distributed testbed designed to support experimental-driven research in parallel and distributed systems. The experimental setup of the Edge-to-Cloud continuum consists of a total of 14 nodes in the *nova* cluster. Each node is originally equipped with 2 processors of 8 cores each, 64GB memory, and 598GB storage. Among the reserved nodes, there are 11 Edge nodes, 2 Fog, and 1 Cloud node. The original node capacity is not entirely allocatable (see Table I). The setting of the network connections between each type of node are given in Table II. The emulation of the continuum on *Grid'5000* is achieved via the framework E2Clab [34]. It allows specifying the communication constraints and separating Edge, Fog, and Cloud nodes into independent virtual networks.

The system has 19 data sources on the Edge. They generate 25 frames per second of resolution $512p$. The frames are from the COCO2017 validation dataset [35] of size 1GB. The models `YOLOv4-512` and `YOLOv4-416` are pre-trained on the COCO dataset with an accuracy AP_{50} equals to 64.9% and 62.7%, respectively.

Table I: Resource capacity of Edge, Fog and Cloud nodes.

Layer	Cores	RAM (GiB)	Storage HDD
Edge	1	2	2GB
Fog	4	32	20GB
Cloud	8	64	500GB

Table II: Delay and bandwidth of network connections between nodes.

Layer	Average Delays(ms)	Uplink Bandwidth
Edge-Edge	1	1Gbps
Fog-Fog	1	1Gbps
Cloud-Cloud	1	10Gbps
Edge-Fog	4	30Mbps
Fog-Cloud	5	10Gbps

Based on the proposed data-driven scheduling approach, the `Resize` microservice is deployed on the Edge, `YOLOv4-416` on the Fog, and `YOLOv4-512` and `Draw` on the Cloud. Each task reserves the entire capacity of the node on which it is deployed. Regarding the two services placed on the Cloud, they share the resources by half.

The `YOLOv4`, and `Draw` microservices have one instance each and `Resize` microservice has 11 instances. These microservices constitute two possible pipeline configurations: Cloud-only and Fog-only. Data source assigned to the Cloud-only pipeline uses the `YOLOv4-512` model for the analysis stage. However, with Fog-only pipeline, the data source uses the `YOLOv4-416` model.

The system evaluation consists of four experiments running for 20 minutes. Their purpose is to measure the average makespan and accuracy of the application during runtime with and without the latency-accuracy tradeoff. Experiments 1 and 2 use all system data sources and experiments 3 and 4 use only one. In experiments 1 and 3, the quality of generated data is $512p$. All data sources are assigned to the Cloud-only pipeline. However, in the experiment 2, the data adaptation strategy was used once. Among all data sources, the data quality of only 1 data source is reduced to $416p$. So, unlike the rest, this data source is assigned to the Fog-only pipeline. In experiment 4, the data adaptation strategy is applied to the single data source used.

B. Evaluation Results And Discussion

Figure 3 shows the average system makespan variation with 19 data sources with and without using the data adaptation. In experiment 1, without adaptation, the system takes up to around 5.7 hours to analyze the data generated by 19 data sources during a 20 minutes test. However, in experiment 2, when applying the adaptation strategy on one data source, the average system makespan was reduced to around 2.6 hours. The gain in the average system makespan between experiments 1 and 2 is up to 54.4% (≈3.1 hours).

Figure 4 shows the variation of average system F1-score with and without the data adaptation strategy. Results show that system accuracy decreased from 71.19% to 63.82% after reducing the quality of one data source from 512×512 to 416×416. The system accuracy remains higher than the default accuracy threshold fixed to 50%. The obtained accuracy depends mainly on the models used. In this work, the accuracy results are for the pre-trained YOLOv4 models.

Figure 5 shows the average system makespan of a single data source with and without using the data quality adaptation strategy. In experiment 3, without any data adaptation, the system takes up to around 1 hour to analyze all the data generated by the single data source. However, when applying the data adaptation strategy in experiment 4, the average system makespan is around 1.3 hours.

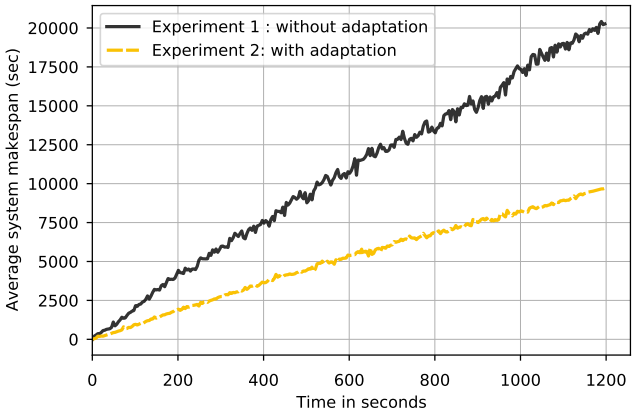

Figure 3: With 19 data sources, the system makespan with data adaptation in experiment 2 is lower than in experiment 1, where no tradeoff solution is used.

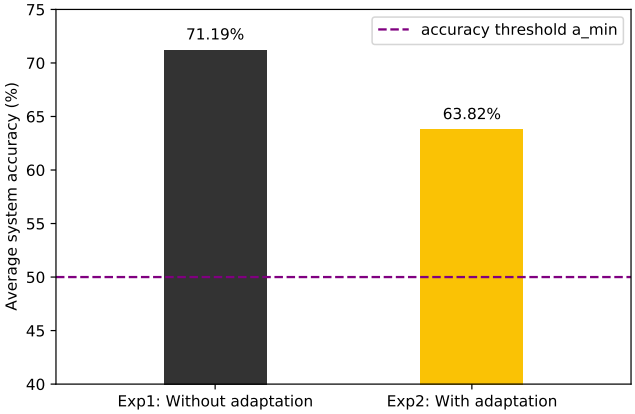

Figure 4: With 19 data sources, average system accuracy decreased in experiment 2 (with data adaptation) compared to experiment 1 (without data adaptation). Despite this, it remains higher than a fixed threshold equals to 50%.

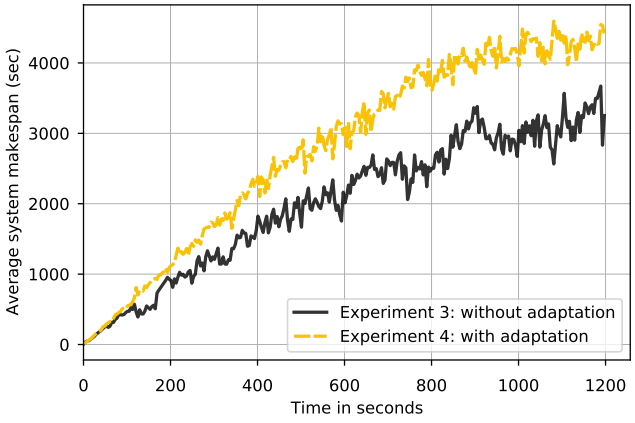

Figure 5: With a single data source, the average system makespan with data adaptation in experiment 4 is higher than in experiment 3, where no tradeoff solution is used.

During the system evaluation, we observed that when the Fog-Cloud network is constrained due to high load, the Edge-only pipeline can be an interesting approach, as it is shown in experiment 2. However, if no high load in the system as in experiments 3 and 4, the Cloud of 10Gbps bandwidth performs better in terms of makespan than the Edge-only configuration. This indicates that, for Deep Learning applications, applying the data adaptation strategy to maximize the utility function is only possible when the Fog-Cloud network performance is constrained. Otherwise, the Cloud-only pipeline configuration is the better choice in terms of average system makespan.

VIII. Conclusion

This paper proposes a system that support time-critical Deep Learning workflows in an Edge-to-Cloud environment. It distributes the workflow automatically across the continuum based on the categories of the tasks. In addition, it reduces the resolution of incoming data when potential latency-accuracy trade-off optimizations are available. The evaluation of an object detection use case on *Grid'5000* showed a gain in average system makespan reaching up to 54.4% compared to a Cloud-only pipeline configuration in a multi-user scenario.

As future work, an interesting direction is designing a resource assignment tool for Deep Learning workflows that considers the characteristics of the load, tasks, and resources to guarantee performance constraints.

Acknowledgments

This research is supported in part by the NSF under grants numbers OAC 1640834, OAC 1835692, and OCE 1745246.

References

[1] A. Voulodimos *et al.*, "Deep learning for computer vision: A brief review," *Computational intelligence and neuroscience*, vol. 2018, 2018.

[2] Z. Zhang *et al.*, "Deep learning for environmentally robust speech recognition: An overview of recent developments," *ACM Transactions on Intelligent Systems and Technology (TIST)*, 2018.

[3] C. Shorten, T. M. Khoshgoftaar, and B. Furht, "Deep learning applications for covid-19," *Journal of Big Data*, vol. 8, no. 1, pp. 1–54, 2021.

[4] D. Balouek-Thomert *et al.*, "Towards a computing continuum: Enabling edge-to-cloud integration for data-driven workflows," *The International Journal of High Performance Computing Applications*, vol. 33, no. 6, pp. 1159–1174, 2019.

[5] P. Beckman *et al.*, "Harnessing the computing continuum for programming our world," *Fog Computing: Theory and Practice*, pp. 215–230, 2020.

[6] D. Balouek-Thomert, I. Rodero, and M. Parashar, "Harnessing the computing continuum for urgent science," *SIGMETRICS Perform. Eval. Rev.*, vol. 48, no. 2, p. 41–46, Nov. 2020.

[7] Z. Chen *et al.*, "Computation offloading and task scheduling for DNN-based applications in cloud-edge computing," *IEEE Access*, 2020.

[8] M. Satyanarayanan *et al.*, "Edge analytics in the internet of things," *IEEE Pervasive Computing*, vol. 14, no. 2, pp. 24–31, 2015.

[9] J. Jiang *et al.*, "Chameleon: scalable adaptation of video analytics," in *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, 2018, pp. 253–266.

[10] B. Taylor *et al.*, "Adaptive deep learning model selection on embedded systems," *SIGPLAN Not.*, vol. 53, no. 6, p. 31–43, Jun. 2018.

[11] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, "Yolov4:optimal speed and accuracy of object detection," *arXiv preprint arXiv:2004.10934*, 2020.

[12] S. Ren *et al.*, "Faster r-cnn: Towards real-time object detection with region proposal networks," *arXiv preprint arXiv:1506.01497*, 2015.

[13] M. Ali *et al.*, "Res: Real-time video stream analytics using edge enhanced clouds," *IEEE Transactions on Cloud Computing*, pp. 1–1, 2020.

[14] G. Carvalho *et al.*, "Edge computing: current trends, research challenges and future directions," *Computing*, pp. 1–31, 2021.

[15] A. Yousefpour *et al.*, "All one needs to know about fog computing and related edge computing paradigms: A complete survey," *Journal of Systems Architecture*, 2019.

[16] M. G. S. Murshed *et al.*, "Machine learning at the network edge: A survey," *CoRR*, vol. abs/1908.00080, 2019.

[17] C. Liu *et al.*, "A new deep learning-based food recognition system for dietary assessment on an edge computing service infrastructure," *IEEE Transactions on Services Computing*, vol. 11, no. 2, pp. 249–261, 2018.

[18] A. R. Zamani *et al.*, "Deadline constrained video analysis via in-transit computational environments," *IEEE Transactions on Services Computing*, 2020.

[19] X. Kong *et al.*, "Real-time mask identification for covid-19: An edge computing-based deep learning framework," *IEEE Internet of Things Journal*, 2021.

[20] N. Dragoni *et al.*, "Microservices: yesterday, today, and tomorrow," *Present and ulterior software engineering*, pp. 195–216, 2017.

[21] P. Emami Khoonsari *et al.*, "Interoperable and scalable data analysis with microservices: applications in metabolomics," *Bioinformatics*, 2019.

[22] M. Taneja *et al.*, "Smartherd management: A microservices-based fog computing–assisted iot platform towards data-driven smart dairy farming," *Software: practice and experience*, vol. 49, no. 7, pp. 1055–1078, 2019.

[23] H. Wang, Z. Liu, and H. Shen, *Job Scheduling for Large-Scale Machine Learning Clusters*. New York, NY, USA: Association for Computing Machinery, 2020, p. 108–120.

[24] M. Zhang, C. Krintz, and R. Wolski, "Stoic: Serverless teleoperable hybrid cloud for machine learning applications on edge device," in *2020 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*. IEEE, 2020.

[25] A. H. Jiang *et al.*, "Mainstream: Dynamic stem-sharing for multi-tenant video processing," in *2018 USENIX Annual Technical Conference (USENIX ATC 18)*. Boston, MA: USENIX Association, Jul. 2018, pp. 29–42.

[26] H. Zhang *et al.*, "Live video analytics at scale with approximation and delay-tolerance," in *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. Boston, MA: USENIX Association, Mar. 2017, pp. 377–392.

[27] A. R. Zamani *et al.*, "Submarine: A subscription-based data streaming framework for integrating large facilities and advanced cyberinfrastructure," *Concurrency and Computation: Practice and Experience*, vol. 32, no. 16, p. e5256, 2020.

[28] C. Wang *et al.*, "Joint configuration adaptation and bandwidth allocation for edge-based real-time video analytics," in *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications*, 2020, pp. 257–266.

[29] W.-J. Kim and C.-H. Youn, "Lightweight online profiling-based configuration adaptation for video analytics system in edge computing," *IEEE Access*, vol. 8, pp. 116 881–116 899, 2020.

[30] Q. Liu *et al.*, "An edge network orchestrator for mobile augmented reality," in *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*, 2018, pp. 756–764.

[31] J. Hui, "Object detection: speed and accuracy comparison (Faster R-CNN, R-FCN, SSD, FPN, RetinaNet and YOLOv3)," 2018, http://www.lighterra.com/papers/videoencodingh264/.

[32] Z. Houmani *et al.*, "Enhancing microservices architectures using data-driven service discovery and QoS guarantees," in *2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID)*. IEEE, 2020, pp. 290–299.

[33] Desprez and al., "Adding virtualization capabilities to the grid'5000 testbed," in *Cloud Computing and Services Science*, I. I. Ivanov, M. van Sinderen, F. Leymann, and T. Shan, Eds. Cham: Springer International Publishing, 2013, pp. 3–20.

[34] D. Rosendo *et al.*, "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, Sep. 2020, pp. 1–11.

[35] T. Lin *et al.*, "Microsoft COCO: common objects in context," *CoRR*, vol. abs/1405.0312, 2014.