Handling Data Handoff of AI-based Applications in Edge Computing Systems

Handling Data Handoff of AI-based Applications in Edge Computing Systems

Domenico Scotece, *Member, IEEE*, Claudio Fiandrino, *Member, IEEE*, Luca Foschini, *Member, IEEE*

***Abstract*—Edge computing aims at better supporting low-latency applications. One of its key techniques is computation offloading, the process that outsources computing tasks from resourced-constrained mobile devices and moves them to edge data centers. In this paper, we tackle an emerging problem within the umbrella of computation offloading, i.e., migration of offloaded inference tasks of Artificial Intelligence (AI) trained models. Such context tailors migration aspects of *data-sensitive* services where *i)* the value of the updates is inversely proportional to the data age and *ii)* outage is highly detrimental to accuracy. To tackle this challenge, we propose Mobile Edge Data-handoff (MED) a framework able to relocate inference or online training tasks from one edge datacenter to another by moving only the necessary data to minimize any accuracy drop during the process. We implemented MED in a well-known edge computing emulator, openLEON, and experimentally verified its performance with an AI-based Industry 4.0 application that forecasts the gas flow in a chemical plant. For our experiments, we use a real, open-source dataset that contains sensors readings. Collected results show that MED, employing proactive data handoff algorithms, is able to minimize the packet loss during the handoff thereby providing guarantees on the inference accuracy.**

***Index Terms*—Multi-access Edge Computing, Industry 4.0, data handoff, computation offloading**

I. Introduction

In the recent years, edge computing has gained momentum and attracted the interest of both industry and academia [1]–[5]. Driven by the widespread diffusion of mobile and wearable devices and the tremendous growth of the Internet of Things (IoT), a number of new applications has emerged and calls for stringent requirements such as intensive computation and tight latency budgets. Examples of such applications are virtual and augmented reality (VR/AR), navigation, and gaming [6]. Although preliminary measurement studies on existing commercial edge platforms are yet not so promising in terms of latency improvement [1], edge computing is regarded as key to meet strict constraints by pooling computing resources closer to the end user and not in the far cloud. Such concept coincides with other paradigms, including fog computing, mist computing [7], and Multi-Access Edge Computing (MEC) [8] that work at different protocol stack layers. In this paper, we focus on the application layer and tackle a problem that is independent of the actual technology/standard used; hence, hereafter we will only use the term edge computing to refer to all those significant and ongoing efforts in this area.

Computation offloading is the process that moves away from the resource-constrained mobile devices part of the computing tasks onto the edge, for instance, to a micro data center deployed in close proximity of the network access that we call herein after edge data center (EDC). This is key to cut the latency and augment performances of the mobile devices by prolonging battery life time and by enabling the execution of applications that would be impossible to run on the mobile devices alone [9], [10]. In the presence of mobility, offloading is more complex because the process of detaching from the current service anchor (base station and computing server) and attach to the new anchor increases latency and that negatively impacts user Quality of Experience (QoE). To this end, migrating the service anchor in a way that follows the user movement has been proven effective [11], [12].

In this paper, we take the research on computation offloading and service migration in edge computing systems one step forward. In particular, we tackle an emerging problem within the umbrella of computation offloading, which we partially analyzed in [13], i.e., migration of offloaded inference tasks of Artificial Intelligence (AI) pre-trained models. This is a radically different problem than the one concurrently tackled in the literature (e.g., [11], [12]) and is becoming more and more important in the context of beyond 5G and 6G use cases like Industry 4.0 (I4.0) and unmanned mobility [14], [15]. Let us consider the example of precision sensing and actuation that is typical in I4.0 scenarios. Let us focus on smart factory applications that leverage AI for handling big data and machine learning algorithms to monitor and evaluate processes [16]. In this scenario, the value of the updates is inversely proportional to their age; hence, the capability to process the latest sensing readings is of the utmost importance [17]. However, sudden downtime in the EDC devoted to process such tasks might prevent the timely computation thereby leading to a drop of inference accuracy. Static over-provisioning or resource reservation are trivial solutions that can be obviously outperformed by more intelligent approaches. To this end, we seek a self-configuring mechanism, called Mobile Edge Data-handoff (MED), which fully aligns with the 6G vision of zero-touch orchestration and configuration, and we propose an intelligent data handoff migration solution that is easy to integrate in MEC systems. In a nutshell, upon detecting a possible system congestion that could lead to downtime, MED migrates *both data and the process status* to a nearby EDC so to reduce the amount of data loss and to always provide an acceptable accuracy level.

MED differs from related works in several ways. On

Dr. Fiandrino's work is supported by the Juan de la Cierva grant (IJC2019-039885-I).

Manuscript received August 9, 2022.

D. Scotece and L. Foschini are with the University of Bologna, Bologna, Italy. E-mail: {firstname.lastname}@unibo.it

C. Fiandrino is with IMDEA Networks Institute, Madrid, Spain. E-mail: {firstname.lastname}@imdea.org.

the one hand, the existing literature on service and data migration on MEC and fog computing has primarily focused on the migration aspects. Examples include optimization of live migration of Virtual Machines (VMs) [18], container migration such as [19] that propose to use containers in its live service migration framework by showing the associated advantages, or IBM Voyager [20] that proposed a live container migration service, which is vendor-agnostic, with consistency guarantees, and designed according to the Open Container Initiative (OCI) principles. On the other hand, several virtual machine (VM) placement algorithms have been proposed to optimize various metrics such as network overhead [21] and server congestion [22]. As better clarified later on, this wealth of work largely differs from ours because *i)* MED is designed to achieve the best of the two worlds, i.e., to align with the reactive schemes in relation to accuracy drop and be as efficient as proactive schemes in terms of completion time, *ii)* its design considers drop of accuracy reduction as a primary objective, and *iii) it enables to relocate inference from EDC-A to EDC-B by moving fresher data.* Actually, as we say later in the paper, what is important for inference is the fresher data. Finally, unlike our approach, several research works in the literature do not provide evidence of a real deployment experience nor in-the-field experimental evaluations [23].

We implement MED and extensively test an I4.0 application with an open-source dataset. While a number emulators for MEC exists [24]–[26], they model either network or computing mechanisms, but fail at modeling both components precisely thereby preventing to perform a fully-fledged performance evaluation that encompasses both mobile network and computing domains. For this reason, we implement our system in the openLEON testbed that allows to benchmark the performance of edge solutions end-to-end, from the EDC to the end mobile user or device [27]. openLEON uses srsLTE [28] to emulate the mobile network, which is LTE. Despite being well-known for higher latencies than 5G New Radio (RN) in standalone mode, many solutions can be applied for its reduction [29] and even make this technology usable for Virtual Reality (VR) [30]. The synopsis of contributions of this work is as follow:

1) We propose MED, a framework for service and data migration in MEC environments that resembles a proactive handoff algorithm. Moreover, MED enables data handoff for data-sensitive applications, especially for smart industry applications. We also show how to implement MED in Kubernetes.
2) We conduct extensive evaluations implementing MED in openLEON, a well-known emulator for edge computing and tested with a real I4.0 application.
3) The results demonstrate that MED reduces inference loss and data loss. Compared to the case of not performing the handoff that leads to unlimited data loss during outages, MED limits the number of packets lost (down to 2 packets in the worst case) without service interruption.

The remainder of the paper is organized as follows. We explain our motivation for this work in Section II. We introduce the models and the problem in Section III. The MED solution and a possible implementation blueprint are proposed in Sections IV. We evaluate its performances in Section V. After reviewing the related works in Section VI, we report a thorough retrospective discussion of our solution in Section VII. The conclusion of this paper is presented in Section VIII.

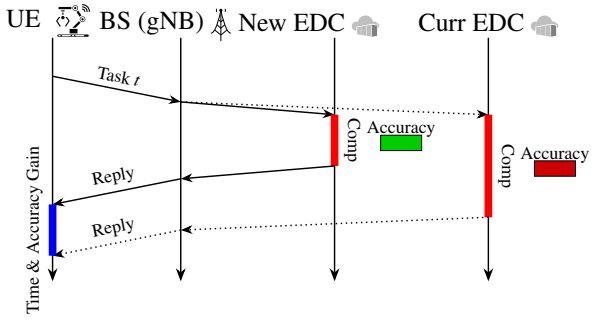

Figure 1. Task scheduling results at different edge node. The accuracy improves (green box) if the computation is moved from the current EDC of execution.

II. Motivating Example

As described in the 5G-PPP specifications, *factories of the future* will leverage the integration between the Cyber-Physical Systems (CPS) and the applications of the Internet of Things (IoT) [31]. For example, in process automation industries (e.g. automotive production, industrial machinery and equipment, consumer products) typical cycle times are around 100 ms and require high reliability (i.e., 99.9999)% [32]. Cloud computing in conjunction with edge computing can effectively handle those issues by guarantying high reliability very low latencies by offloading the computation at the edge of the network.

To work properly, an industrial machine that offloads data for computation to a EDC requires low latency and high availability. Task t, depicted in Fig. 1, is sent from an industrial machine (representing the User Equipment - UE) to an EDC (named current EDC) for processing. Industrial applications already use networks technologies such as wired Ethernet or wireless technologies like WiFi [33] that represent a starting point for more demanding, transformative automation using 5G technologies. Moreover, the factory owner has full control of deployment environment and industrial 5G networks can be designed and optimized differently in different shop floors. For instance, in the same industry center, there could be several shop floors that work in different sectors, and the workload is likely to vary over time. Typically, a node of one floor is connected to a single Base Station (BS) that interconnects the node with the closest EDC (e.g., on the same floor). In the event that the current edge node is running out of resources (e.g. high CPU consumption), it may not be able to meet the service requirements.

In this paper, we consider the Industrial Analytics scenario—is the use of analytics in IIoT systems [34]. The most representative example is predictive maintenance where analytics can be directly applied in the control loop of machines to adapt their behavior in order to avoid system outage. Generally, industrial analytics leverages machine learning algorithms to analyze large amounts of data gathered from industrial sensors. However, there are several challenges and

requirements posed by industrial analytics. The first and the most important is the *Correctness*. Industrial Analytics must satisfy a higher level of accuracy when providing analytic results [34].

Back to our industry center example, if the EDC located on the same floor of the node is predicted to be temporarily overloaded in the near future, it may be beneficial to move the computation to another "idle" EDC located on another shop floor. Indeed, by leveraging the computing load variations across the different EDCs, MED exploits un-utilized capacity when available. Figure 1 shows the drop of accuracy and the service total time if the computation would remain at the current EDC and thus with no handoff procedure into action. The data loss during server downtime leads to a drop of accuracy and, in turn, the predictive maintenance system will not work properly. To avoid this critical situation, it is important to promptly move the entire predictive maintenance system to the new EDC in order to lose as little of accuracy as possible. The gain in terms of accuracy and service time can be appreciated in Fig. 1, see the blue line at bottom left of the figure. Another different solution might be to collect data locally (i.e., in the sensor's memory) during the service downtime and upload those data once the system restores. However, considering a scenario where real-time data streams have to be processed by machine learning algorithms online (e.g., inference for alarm prediction) and if this is not possible, such data can be considered lost because it is no longer useful being outdated.

Therefore, to compute correctly the task t at the new EDC, both service and input data shall be available. Nowadays, services are commonly built as microservices (or containers) typically managed by an orchestrator engine such as Kubernetes [35]. Moreover, service autoscaling functionalities are included in Kubernetes and allow service migration purposes. The same could be applied to the input data if stored in a container. The problem occurs when moving AI-based applications. On the one hand, service migration solutions should support proactive data migration as well in order to grant negligible start-up time [36]. On the other hand, data migration strategies should be not proactive, but move data "just in time"; proactive migration would create too much unnecessary overhead because for inference purposes only the most recent data counts.

Finally, MED handoff process is characterized by three different modules: *triggering*, *decision*, and *execution*. The first module, as proved by our model in the next Section, works in a proactive way and is in charge of selecting the best time window to start the handoff according to different metrics. The *decision* module identifies the candidate EDC for the handoff and it works in a reactive way. Specifically, when the *triggering* module triggers the handoff, the *decision* module is turned on. Last, the *execution* module effectively performs the handoff and it works in two distinct ways, that is a proactive way for AI models and applications and a reactive way for data. Specifically, the choice to move data in a reactive way is justified by the recent trend of working with fresher data (Age of Information [17], [37]), our objective is to move only the last window of data that is the most important for AI applications. We call this process *data handoff*.

III. System Model and Problem Formulation

In this section, the system model is first presented, followed by the problem formulation. Finally, the migration condition problem is formulated.

A. System Model

We consider a set of $\mathcal{U} = \{1, 2, \ldots, U\}$ users that access computing resources available from a set $\mathcal{N} = \{1, 2, \ldots, N\}$ EDCs via mobile network technology. Specifically, each user $u \in \mathcal{U}$ can access computing resources of the EDC $n \in \mathcal{N}$ through a wireless link that provides the interconnection to a base station (eNB or gNB according to 4G and 5G terminology respectively) which is the entry point to the network of EDCs $\mathcal{N}$. Without loss of generality, we assume eNBs/gNBs to be interconnected with the EDCs via fiber optics or high capacity wireless links, e.g., millimeter-wave backhaul links [38]. The set $\mathcal{E} = \{1, 2, \ldots, E\}$ defines the set of links that interconnects the set of $\mathcal{N}$ EDCs, i.e., the network of edge data centers can be represented as a graph $G = (\mathcal{N}, \mathcal{E})$. The edge $e = \langle n, n^{'} \rangle$ where $\{e | n, n^{'} \in \mathcal{N}, \text{ and } e^{'} \in \mathcal{E}\}$ represents the link that interconnects EDCs n and $n^{'}$ and is characterized by a time-varying bandwidth $b_{n,n'}(t)$.

Let $c_n \in \mathcal{C}$ be a long-standing computing process that is currently in execution in EDC n:

$$x_c^n = \begin{cases} 1 & \text{if task } c \text{ is executed in EDC } n, \\ 0 & \text{otherwise.} \end{cases} \tag{1}$$

We assume that by default, c_n runs in background and when it becomes active, for its execution it requires a time-varying amount of CPU resources $r_{c_n}(t)$. The CPU load can be expressed in terms of floating-point operations per second (FLOPS) or instructions per seconds (IPS). The CPU load $L_n(t)$ of each EDC n is defined as:

$$L_n(t) = \sum_{c_n} x_c^n \cdot r_{c_n}(t). \tag{2}$$

Call L the maximum acceptable load when all the servers in EDC n run at full capacity. Then, it follows that $L_n(t) \leq L, \forall t \geq 0$.

For the set of computing processes $\mathcal{C}$, we consider tasks that are typical of I4.0 scenarios, like actuation, inference of previously off-line trained learning model, etc. All these tasks share a common aspect, i.e., for their execution, it is required to process both historical and fresh information coming from the sensors. Call $O_{c_n}^i$ the output of task c_n at time step i, let Γ_{c_n} the set of outputs generated by c_n: the past outputs are denoted with $\Gamma_{c_n,i}^- \subset \Gamma_{c_n}$. To make the notation easy to follow, from now on we refer always to task c_n, which allows to drop the corresponding index. In other words, $\Gamma_i^- = \{O_{i-1}, O_{i-2}, \ldots, O_{i-\kappa}\} \subset \Gamma$. Let us denote with t_i^E and t_i^S the end and start time at step i. The precedence constraint among the steps is defined by:

$$\begin{cases} t_i^S \geq \max_{h \in \Gamma_i^-} t_h^E & \Gamma_i^- \neq \emptyset, \\ t_i^S = 0 & \Gamma_i^- = \emptyset. \end{cases} \tag{3}$$

In other words, the step i can not start if the previous steps $i-1, i-2, \ldots, i-\kappa$ are not concluded. The task c_n starts its

execution at t_i^S upon receiving input data D_i from the mobile device. Without loss of generality, D_i consists of one or more data packets, e.g., information about sensor readings to be processed. For its execution at step i and to produce the output O_i to be included in Γ, i.e., $\Gamma \leftarrow \Gamma \cup O_i$, task c_n requires D_i and the historical data that produced the previous set of outputs Γ_i^-. We denote the historical data as the set $I_i^- = \{I_{i-1}, I_{i_2}, \ldots, I_{i-\kappa}\}$, hence $O_i = (D_i, I_i^-)$. This captures the case of inference on a previously trained learning model, e.g., the forecast of the status of the system in the next step. Over the set of outputs Γ in a time window $\Delta_t = (t, t+\tau]$, $\forall\tau$, further processing is computed and if some criteria are met, e.g., an important event is detected like an abnormal gas pressure in a room, a reply message R_i is sent back to the mobile device, e.g., to inform it about the actuation to be performed. For example, the computation over Γ is needed to determine the angle of rotation given the past actions that have been taken by the robot or to trigger an alarm because the gas pressure in a room has augmented too quickly.

B. Problem Formulation and Migration Condition

Our objective is to maximize the reliability of the system. Call $A(\Gamma)|_{\Delta t}$ the accuracy of the event detection that is computed over Γ in the window Δt. A is maximized when the computation takes place over all the outputs $O_i \in \Gamma = \Gamma^*$:

$$\max_{\Delta t} A(\Gamma). \tag{4}$$

In the presence of service downtime in the edge data center n, the mobile device can not send fresh inputs D and therefore some of the outputs are missing and the resulting set is not Γ^*, but $\Gamma^\dagger$. Also, the presence of sudden variations in the load $L_n(t)$ (the data center load is known to be a highly varying time-series [39]) might prevent the timely computation of O_i and therefore leading to $\Gamma^\dagger$ as well. Such conditions trigger the migration of the computation from EDC n to a nearby EDC $n^{'}$. To adhere to (4) for a successful migration, both Γ_i^- and I_i^- should migrate from EDC n to $n^{'}$. When not successful, the computation occurs on older values, which leads to a drop of accuracy. This problem, can therefore be casted to the problem of cache replacement when the server has no knowledge about incoming data arrival at the time of making the decision (of moving data in our case), which is NP-HARD [40].

The goal of the algorithm explained in the next Section is to understand when this migration should be triggered in order to minimize the start time t_i^S. Indeed, when migrating, the start time of the process in a new server is the most time consuming component. To understand if it is necessary to migrate a service, we resort to mechanisms that anticipate future computational load [39], [41], i.e., $L_n(t+1), L_n(t+2), \ldots, L_n(t+T)$.

IV. Efficient Data Handoff at the Edge

In this section, we illustrate the proposed MED algorithm for data migration tailored for specific AI inference type of tasks according to the discussed theoretical analysis. Note that, the following algorithm is specifically focused on data migration. MED designs the *execution* module to work in a proactive way for AI models and applications by following design guidelines proposed in our previous work [36], [42]. Moreover, MED proactively orchestrates AI applications by using standard solutions such as Open Source MANO and Kubernetes. Prior to an handoff occurrence, MED identifies neighbor EDCs and sends them business logic and trained model so that the AI model is ready to use in case of handoff. In this way, MED minimizes the downtime and service restart. Then, during the normal application workflow, MED strives to move data right before the handoff occurrence to reduce the accuracy drop of inference tasks.

Algorithm 1 Mobile Edge Data-Handoff

1: Fix parameters EDC $n \in \mathcal{N}$, and L^*
2: **At time** $t > 0$
3: Give the average CPU load of each EDC $L_n(t)$
4: Give $L_o(t)$ the average CPU load of the old EDC n_o
5: Give L^* the threshold
6: ***Decision***: Edge Selection (Periodically, select the new EDC n' based on average CPU load and run the service on it)
7: ***Execution***: Move the AI model and the set of computing processes C to neighbor EDCs
8: ***Triggering module***: handoff occurs
9: **if** $L_o(t) > L^*$ **then** ▷ The node n_o is running out of resources
10: ***Data Backup***: Create the archive $A \leftarrow \text{zip}(O_i)$
11: ***Send Backup***: Send the archive A to the node n'
12: ***Restore Backup***: Restore the archive A at the node n' and prepare the service (i.e., unzip(O_i) and restore both Γ_i^- and I_i^-)
13: ***Start Computation***: Start the set of computing processes C at the node n'
14: **end if**

A. MED Algorithm

In order to start the migration process from the current EDC n_o towards the new EDC n', MED relies on timely detection of system congestion or failures. It is really important to have historical data at the new EDC n' for ensuring an acceptable delay and accuracy. Most of I4.0 applications (e.g., predictive analysis, fault predictions, preventive maintenance and automation, etc.) regard AI as the enabler for smart industry and, to operate, those AI models need historical data for tasks like forecasting the failure of equipment That said, we claim the importance of having data, at least the last data, to the new EDC n' in order to promptly restart the computation. To ensure that it is important to: i) identify potential exhaustion of resources at the old EDC n_o early; ii) select the candidate EDC n'; and iii) migrate the historical data when the handoff starts. Note that, as we explain later, we save the historical data in a circular buffer and during the handoff we move only data in the buffer.

Therefore, the algorithm proposed in the MED framework is based on the following principles. First, as explained in Section III, it is very important to timely predict node congestion for multiple reasons. On the one hand, it allows us to properly select the new EDC n' based for instance on average load, and

to move data at the node n' when handoff happens. On the other hand, it enables us to greatly reduce the service interruption time for improving the service accuracy (fewer data are lost from the user perspective). Note that, decision and triggering modules are executed proactively before the handoff. The same is applies to the execution module only for the AI model and application to ensure that they are available at the new EDC. To efficiently manage the handoff, we define the parameter (line 1 Algorithm 1) L^* which represents the threshold value of CPU consumption to start the handoff process. The $L_n(t)$ value is defined as an average prediction of the CPU load of the EDC n_o at time t. Note that, the process to select the new EDC n', ***Decision*** (line 6 Algorithm 1), is performed periodically and in background and is not a core part of the MED algorithm. For the purpose of this work, we adopt a selective strategy based on the CPU load at the edge (we select the more "idle" EDC at time t). Moreover, a description of possible implementations of how to run the service at new EDC n' is described in Section IV-B. That said, when the average prediction of the CPU load of the current EDC $L_o(t)$ is greater than the threshold L^* (line 9 Algorithm 1 - this can be statistically inferred with existing forecasting algorithms) the migration process starts and involves four stages: ***Data Backup***, ***Send Backup***, ***Restore Backup***, and ***Start Computation***. The ***Data Backup*** step (line 10 Algorithm 1) creates an archive A, typically a tar o zip archive, containing all the historical data collected at the old EDC n_o (i.e., $A \leftarrow \text{zip}(O_i)$). Specifically, the archive contains both Γ_i^- and I_i^- sets respectively the generated output data and the input gathered data. The remaining steps (line 11, 12 and 13 Algorithm 1) involve directly the selected EDC n' and are respectively ***Send Backup*** step which sends the archive A from the EDC n_o to the EDC n', ***Restore Backup*** step that restores the archive A at the EDC n' i.e., unzip(O_i) and restore both Γ_i^- and I_i^-), and ***Start Computation*** that re-starts the connection with the user u.

Note that, the service now is hooked at the target EDC n' and we consider the service already available at the selected EDC n', as stated before. Finally, Fig. 2 shows the visual representation of the MED algorithm with emphasis on the Service Interruption time. Specifically, the Service Interruption time involves the steps ***Data Backup***, ***Send Backup***, and ***Restore Backup*** of the MED algorithm.

B. MED: Implementation Blueprint

As we have motivated in Section II, services and network functionalities can be implemented as software components executed on standard operating systems by leveraging the latest technologies in the field, such as Virtual Network Function (VNF) and software light virtualization (Docker Container). Specifically, MED leverages our previous work on the management of Docker containers including the management of the historical data that is implemented as a separated data volume container [36].

Kubernetes allows to efficiently manage services among different EDCs. In particular, Kubernetes defines a concept named Pod that represents a group of one or more application containers, see the example in Listing 1. Therefore, Kubernetes

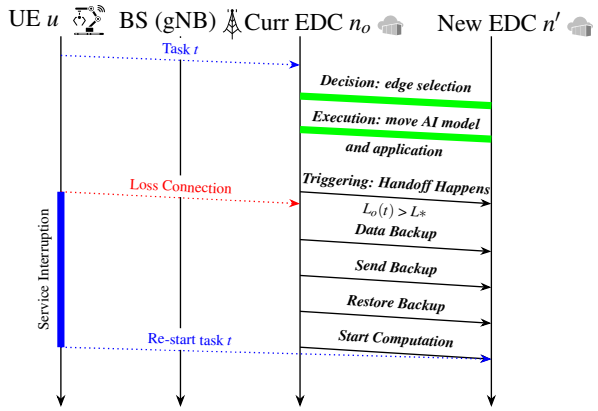

Figure 2. MED sequence diagram

proactively orchestrates Pods among the EDCs while MED defines when and where the handoff process happens. In this way, the AI application is always available to the neighbor EDC.

Listing 1. application/app-inference.yaml

```
apiVersion: v1
kind: Pod
metadata:
  name: app-inference
spec:
  containers:
  - name: <container-name>
    image: <container-image>
    ports:
    - containerPort: <container-port>
```

As motivated in Section II, input data should be moved in a reactive way. Once the handoff is started, MED moves historical data to the new EDC and starts the data volume container at the new EDC according to the Kubernetes primitives. A snippet implementation to automatically manage the historical data is presented below (Listing 2).

```
kubectl apply -f data-inference-EDC2.yaml
```

Listing 2. application/data-inference-EDC2.yaml

```
apiVersion: v1
kind: Pod
metadata:
  name: data-inference
spec:
  nodeName: EDC2 # schedule pod to EDC2
  containers:
  - name: mongo
    image: mongo:latest
    volumeMounts:
    - mountPath: /knowledge
      name: inference-volume
```

Here is very important to have a dedicated framework, such as MED, to guide Kubernetes for data migration, moving data

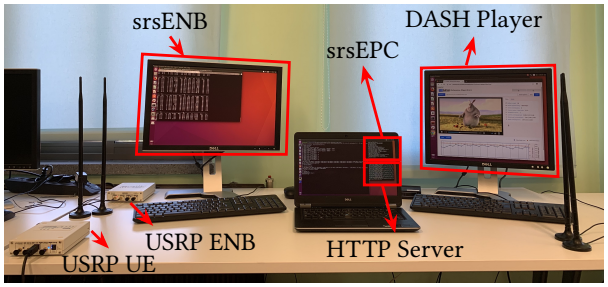

Figure 3. The openLEON platform

in the last possible window time could be essential in the loss of accuracy for AI-based applications.

V. Performance Evaluation

We now evaluate our proposed algorithm MED. First, we explain the methodology and the evaluation settings in the openLeon testbed (see Section V-A). Second, we analyze the dataset used in the analysis (i.e., Section V-B). Finally, we evaluate MED in terms of accuracy and performance, respectively, in Section V-C2 and V-C3.

A. Evaluation Methodology

Figure 3 shows the openLEON setup. We use a laptop equipped with an Intel i7-4600U processor at 2.1 GHz, 8 GB RAM and Linux Ubuntu 16.04 LTS to run the core network (`srsEPC` application from srsLTE version 19.0.6) and the data center network emulated with Containernet [43]. A desktop computer equipped with an Intel i7-6700 processor running at 3.4 GHz, 16 GB RAM and Linux Ubuntu 16.04 LTS, runs the BS application (`srsENB` application from srsLTE version 19.0.6). The LTE physical BS is an Ettus B210 connected with USB 3.0 to the desktop computer. The UE as well uses a Ettus B210 connected to a laptop with an Intel i7-4600U processor up to 2.1 GHz and 8 GB of RAM running the UE application (`srsUE` application from srsLTE version 19.0.6).

To assess the performance of the algorithms, we implement an I4.0 application on openLEON. Specifically, the objective of this application is to predict the gas flow rate in an industry plant for safety reasons. By leveraging a dataset whose details are hereafter reported (see Section V-B), we trained (splitting the dataset into a common 80/20 train-test components) offline a Long-Short Term Memory (LSTM) model (with 64 neurons in the hidden layer, ReLu activation function, and dropout layer set to 0.1 to avoid overfitting) and saved the resulting model in the form of a `.h5` file that is used for inference at later stage. The `.h5` model was then distributed to nearby EDCs (see Section IV). We create a client-server application that allows the mobile terminal to stream sensing readings to the edge server. Upon reception of the readings, the inference phase takes place. An alarm is fired with a message if the forecasts indicate that the flow rate differs from a standard value plus a given bound. Figure 4 shows the architecture when no data handoff occurs.

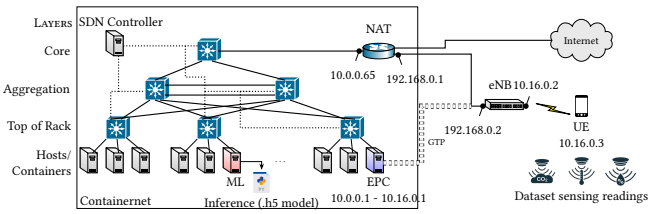

Figure 4. The architecture and the application

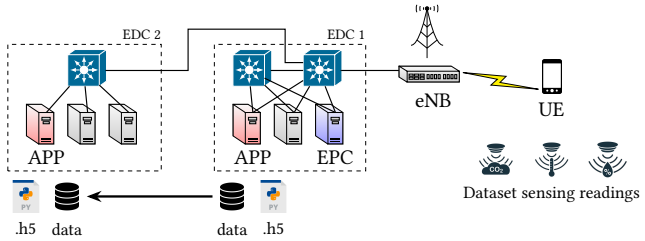

Figure 5. Data handoff for learning

B. Dataset Analysis

We leverage a publicly available IIoT dataset of a chemical detection platform that provides information on CO concentration (ppm), humidity (% r.h.), temperature (ºC), flow rate (mL/min), heater voltage (V), and the resistance of 14 gas sensors[1]. The dataset is composed of around 4 million instances separated into 13 text files. Each file corresponds to a different measurement day and consists of around 300 000 rows and each one includes in addition to the above mentioned indicators, the sample time t. We observe that the sampling frequency is 3.5 Hz.

First, we analyze the characteristics of the entire dataset in order to motivate the decisions made to implement the parameters mathematical model (e.g., the duration of the window - see Section III). Our first analysis is on the deviation between consecutive dataset entries. Specifically, we study the deviation between subsequent steps of the flow rate metric (we vary the distance or deviation step that defines how close are two samples, e.g., given the deviation step $\Delta = 1, 3, 5, 7$, we computed $i - j$ where i is the value of the current sample at time t and j is value of the sample at time $t - \Delta$). The longer is the outage duration, the higher is the number of samples lost. This is to effectively calculate the loss in precision when a fail happens at the EDC, in correspondence to a big event in the dataset there is a certain probability of losing very significant data. Figure 6 shows the cumulative distribution functions (CDF) for different deviation steps, i.e., 1, 3, 5, and 7 samples. The maximum deviation is obtained in correspondence of 5 as a deviation step and furthermore, we noted that using 7 or more as a deviation step the CDF does not change. This analysis reveals the region of interest in the dataset that if lost because of an outage, would negatively impact the inference accuracy. Consequently, although for the evaluation we run

[1]Available at: https://archive.ics.uci.edu/ml/datasets/Gas+sensor+array+temperature+modulation

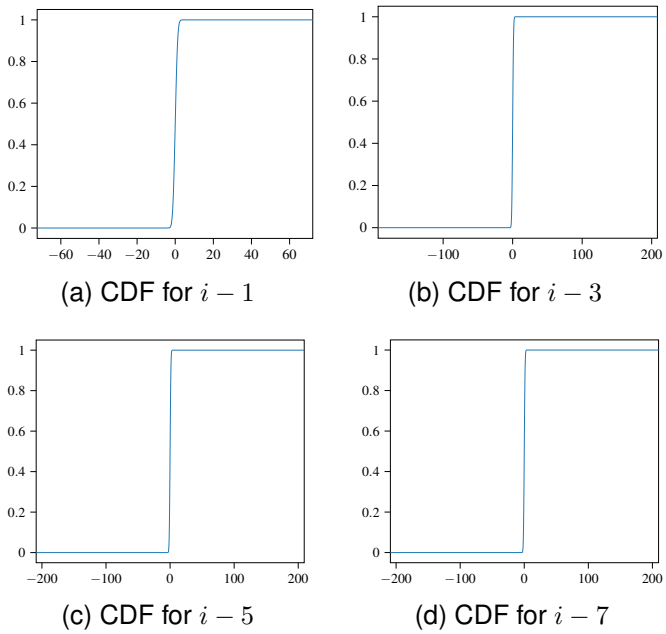

Figure 6. The Cumulative Distribution Functions (CDF) for different deviation steps computed over the entire dataset. We can appreciate that we reach the maximum deviation value after 5 steps.

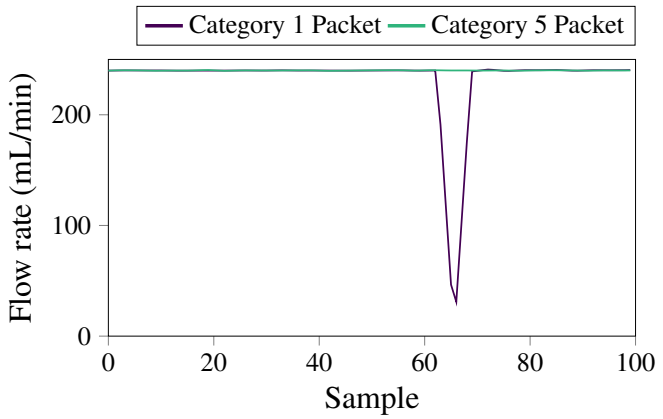

Figure 7. A comparison between a category 1 packet and category 5 packet

MED on the entire dataset, we specifically report results in the correspondence of the highest deviation values to highlight the worst case scenarios in terms of accuracy performance drops.

Finally, we identified five different packet categories. Each packet contains a number of samples (e.g., 100 samples) and we classified packets depending on the number of spikes inside it: category 1 packets contain at least a spike greater than 150, category 2 packets contain at least a spike between 50 and 150, category 3 packets contain at least a spike between 20 and 50, category 4 packets contain at least a spike between 5 and 20 and category 5 packets contain only minor spikes. The categories and the spike levels are obviously tailored to the specific dataset in use and for the flow rate metric. Specifically, if we lost a category 1 packet we probably lost an important behavior of our system, while if we lost a Category 5 packet probably we do not lost any information. Figure 7 shows the content of two different packets of different categories, e.g., a category 1 packet and a category 5 packet. Both contain 100 samples each.

Table I
TYPICAL PARAMETERS FOR MOST COMMON IIOT SENSORS [44]

Type	Sampling Frequency (Hz)	Resolution (Bit)
Inclinometers	10 to 800	16
Accelerometers	1 to 1600	16
Inertial module	1 to 6000	16
Magnetometers	10 to 100	16
Temperature	1 to 200	16
Humidity	1 to 10	16
Pressure	1 to 200	24

C. Experimental Results

1) Preliminary Results: Before delving into the actual results, we make a few clarifying statements on today management of IIoT sensors. In current settings, like those of an industry plant, it is extremely common that the sensor network is heterogeneous, with a wide range of IIoT sensors sharing completely different characteristics, e.g., sampling frequencies, duty cycles, etc... The sampling frequency defines how many samples a sensor can produce in an unit of time. Table I shows the range of operation of the typical today IIoT sensors, which is very large and ranges from 1 Hz to several kHz even for the same type of sensor, e.g., the temperature sensor. Note that, typically, the resolution (number of bits) of one sample is very low, and usually several samples from different sensors are aggregated together, e.g., in a JSON file, to reduce the network overhead.

We now perform a preliminary study that aims at highlighting the relation between the sampling frequency of IIoT sensors and the possible data loss during the service handoff. With the increase of the sampling frequency, the number of generated samples increases; accordingly, and consequently, the network overhead is reduced. However, the processing time increases as well and in case of system outage we may loss more data. Therefore, we simulate a system composed of a mix of sensors that have different sampling frequencies while we keep a fixed observed window set to 1 second. The results are shown in Fig. 8 and Fig. 9 respectively for processing time and for data exchanged. We observed that processing time and packet size grow exponentially if the number of samples increases. Therefore, the choice of the number of samples should be tailored to have a reasonable network overhead and less data loss in case of system outages.

2) Results about Inference Accuracy during Data Handoff: According to our model, we calculated the inference on a previously trained learning model to forecast the status of the system in the next step and subsequently detect anomalies. To ensure this, we calculated the average inference value for every single packet which is composed of 100 data samples. We chose to merge together 100 data samples in a single packet due to the high overhead cost to send one single sample at a time. The process calculates the inference value at step $t+1$ according to the entire historical data gathered at the edge. The historical data in our testbed is composed of ten input file $\{i_0, i_1, \ldots, i_9\}$

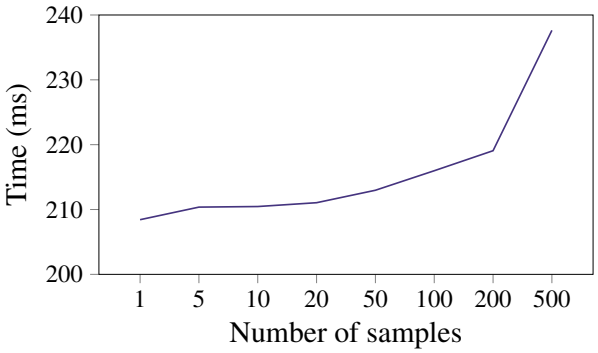

Figure 8. Processing times with different number of sample at a sampling frequency of 1 Hz

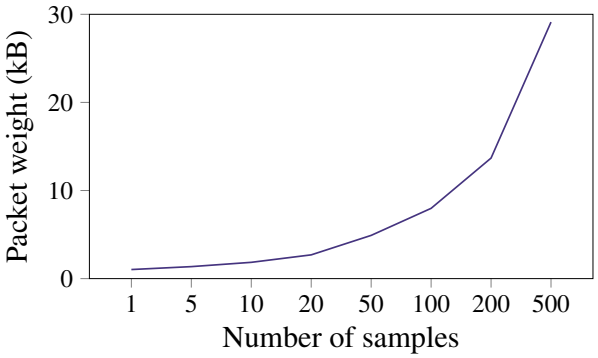

Figure 9. Data size in bytes for different number of sample at rate 1 s

and ten output file $\{o_0, o_1, \ldots, o_9\}$. Those file are put into two circular buffers (one for the input file and the other for the output file) where when a new file is produced the oldest file will leave the buffer. The output files are used for further processing; for instance, if some criteria are met, an important event is detected (e.g., decision process). Finally, let us note that we used circular buffers because are the ideal structures for cases where data production and consumption might happen at different rates and allow to use always fresh data. This choice is pretty standard in networks, e.g., for packet scheduling [45]. See Fig. 10 for a complete vision of our implemented circular buffer.

Previous studies on outages of edge computing platforms indicate that there are 2 min of outages every two weeks on average [46], [47]. By relating this information to our dataset, this is equivalent to losing 4 data packets with a 3 Hz sampling frequency. Our objective is to reduce the data loss because this translates into a lower inference accuracy. Depending on

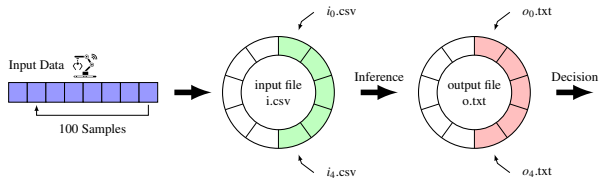

Figure 10. MED historical data representation in a circular buffer

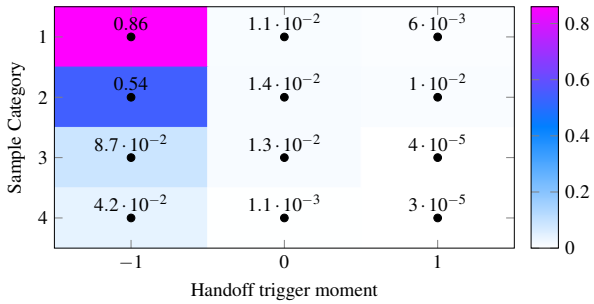

Figure 11. Drop of accuracy according to relation between sample category and handoff trigger moment.

the phenomenon under observation, the dynamics of the time series might change significantly or not in a short period of time. For example, as discussed in the previous section, for the dataset under analysis and for the gas flow rate metric, the time-series presents itself to be very regular with sudden spikes, see Fig. 7. Therefore, the loss of one or more packets may be totally insignificant if the packet does not features important events, like spikes, or severe if the packet does register important events. Precisely, Fig. 11 connects the drop of accuracy with the category of the sample and the handoff trigger timing. Specifically, is important to start the handoff process not in correspondence of a packet with an important spike. To better understand Fig. 11, let us explain the handoff trigger moment which is intended to be the time instant where the handoff is started compared to an important spike. The value -1 represents the packet before the important spike. The value 0 represents the exactly packet that contains the important spike and, finally, the value +1 represents the moment after the important spike. Therefore, the category shows the strictness of the accuracy drop, before, during, or after a data spike. Certainly, a category 1 packet that contains a very important spike has a big impact on the accuracy drop if we lose that packet due to the handoff process. While if we lose a category 4 packet during the handoff, the drop of accuracy is not that serious. The value reported in the Fig. 11 are the absolute error between the "real" average inference (value without the handoff) and the predicted average inference with the handoff. Let as note that category 1 packets present a significant spike but immediately after they work as they should without presenting intermediate steps as we noted in most of category 2 and 3 packets. This trend should be noted from Fig. 11 by comparing the drop of accuracy in the case of handoff trigger moment 0 and +1 for category 1 and category 2 packets. Finally, in that analysis we have omitted the category 5 packets due to non significant results.

3) MED Performance Evaluation: We widely assessed and validated the feasibility of our solution; this section presents a comprehensive selection of experimental results achieved in our testbed deployment scenario. The testbed evaluates the performance of the MED framework in terms of handoff procedure completion times by carrying out over 50 runs on the openLeon emulator.

The testbed calculated the performance of the MED framework by considering the migration of the accumulated historical

data from the old EDC to the new EDC to ensure service continuity. To the best of our knowledge, recent service migration solutions tailored explicitly to AI-based applications, are focused on service scheduling and orchestrating [48]. Moreover, no migration algorithms with the goal of reducing the loss of accuracy are presented in the literature. For this reason, we compare the MED framework with standalone Kubernetes service handoff algorithms both proactive and reactive, specifically, following the design guidelines mentioned in [36]. Furthermore, we created two prototype algorithms (i.e., one reactive, and the other proactive) that follow these principles. Specifically, we considered a tar archive that contains all the historical data that first is created at the old EDC, second is sent to the new EDC, and, finally, is restored at the new EDC. The results shown in Fig. 12 highlight the aforementioned steps of the MED algorithm: the ***Data Backup*** step that takes 9 ms, the ***Send Backup*** step that takes 18 ms, and the ***Restore Backup*** that takes 6 ms (line 10, 11 and 12 Algorithm 1). The last step, ***Start Computation***, depends on the specific application and does not count the service startup stage. Specifically, in the MED framework the service is already available at the new EDC when the handoff starts. This behavior is similar to the proactive service handoffs.

In the case of reactive handoff, the handoff process is started when the User u loses connection with the old EDC. In this kind of handoff, the migration involves all data to the new EDC which is around 63573 Bytes. ***Data Backup*** stage, ***Send Backup*** step, and ***Restore Backup*** step took around 50 ms, the problem in most of the reactive handoff systems lacks services already active at the new EDC. Therefore, ***Startup Service*** step is needed before the ***Start Computation*** stage. On the contrary, in the proactive handoff not only can we provide needed services at the new EDC before the handoff starts, but we can proactively move a portion of data at the new EDC. For this specific evaluation test, the algorithm proactively moves 75 % of data so that to move leftovers data when the handoff starts. This results in less execution time for each protocol stage. Figure 12 shows the reactive and proactive results and also displays a comparison with the MED framework. In conclusion, we proved that the MED framework takes inspiration from Kubernetes standalone proactive handoff systems, but rather with a different objective is the minimum loss of accuracy.

Finally, for the sake of completeness, we report in Fig. 13 the comparison in the drop of accuracy between our MED algorithm and the Kubernetes-based reactive and proactive algorithms during the handoff. The drop in accuracy is calculated considering different numbers of spikes present in the last data packet. As specified before, our sensor application sends 1 data packet containing 100 samples every 1 second. If we have no data spikes inside the packet, proactive algorithms, for sure, work better than reactive algorithms. This means that we can anticipate the migration without losing accuracy and gain on migration time. However, if we have unstable data (e.g., multiple spikes inside a data packet) and we use proactive solutions, we may incur some drop in accuracy. This happens when we lost an important packet. Using reactive solutions, such as MED, allows us to delay the migration as much as possible in order to gather fresher data. To conclude, the MED

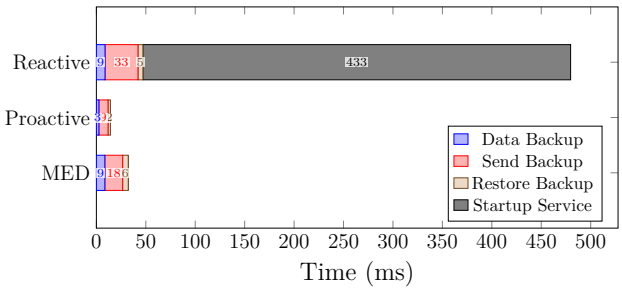

Figure 12. Comparative breakdown of the handoff procedure completion time of MED versus proactive and reactive schemes

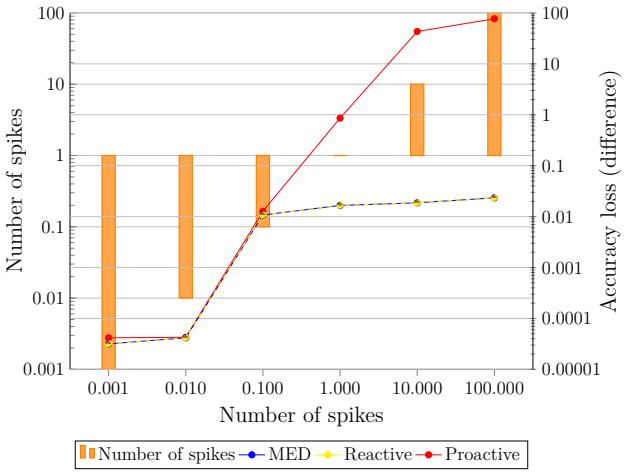

Figure 13. Comparison of MED and proactive handoff in terms of a drop of accuracy according to the data spikes

solution brings the best of both approaches. On the one hand, it offers a service downtime comparable to proactive approaches. On the other hand, the behavior in terms of gain in accuracy is similar to reactive solutions.

VI. Related Work

We now present related works in the areas of edge computation offloading and service migration to the edge. Furthermore, we also provide an overview of the research on data replication between datacenters.

A. Offloading Computing at the Edge

Offloading computation to the edge has attracted extensive research in the past years because of the inherent advantages it can offer (e.g., lower latency and jitter). For the IoT ecosystem, offloading computation at the edge also brings considerable advantages in terms of quality of service [49]. On the one hand, computation offloading at the edge has been applied to Industrial IoT [50] in order to drop the delay in typical industrial closed control loop applications. On the other hand, computation offloading oriented optimization is also proposed, such as the energy-efficient computation offloading

and resource allocation [51]–[53]. Specifically, MECO (Mobile-edge computation offloading) [52], is envisioned to be a promising solution for prolonging the battery life and enhancing the computational capacity of mobile nodes. In [54], P. Zhou et. al have proposed an handoff solution for 5G and MEC systems for computation-intensive applications. However, they do not take the importance of historical data or data in general into account but focusing on pure handoff aspects. Finally, another relevant scenario for computation offloading is edge-cloud collaboration. This scenario has been studied in [55], [56]. In a nutshell, edge computing and cloud computing will collaborate in several environments including smart cities, smart homes, industrial IoT, connected autonomous vehicles, and so on, to take full advantage of on-premises edge and cloud computing capabilities. Our proposal, even if is not directly related to computation offloading, leverages computation offloading at the edge to solve a sub-problem such as *moving the computation* from an edge data center to another one if this helps to guarantee an acceptable end-to-end delay.

B. Service Migration at the Edge

With the rise of the edge computing paradigm efforts of researches have been focused on its benefits and challenges. One of the uncleared challenges is service migration, which aims to guarantees service continuity across different edge nodes. Cloudlet [57] is one of the seminal examples of edge computing and proposed a mechanism for edge-enabled handoff management based on VM synthesis. In contrast, other seminal works have been focused on VM Live Migration [18] to guarantee low service downtime. Live migration ensures users to continue using the service "during" the migration procedure. This complex mechanism is guaranteed by two relevant algorithms such as pre-copy and post-copy. Pre-copy is the basic approach using by Clark et al. [18], where the VM state is transferred from source to destination while the VM is still running on the source. If some memory pages change during this process, they will be re-copied. On the contrary, post-copy migration defers the VM state transfer phase until after the VM's CPU state has already been transferred to the target and resumed there [58]. Ha et al. [59] proposed a mechanism called VM handoff that supports VM live migration for cloudlet-based applications in edge computing environments. The mechanism uses a parallelized computational pipeline approach to achieve high throughput by leveraging data reduction mechanisms so as to move fewer pieces of VM between edge datacenters. To build services for edge computing, container technologies have gained attraction because of their small memory footprint, rapid boot, and low I/O overhead. Machen et al. [19] have investigated the LXC containers live migration by proposing a layered framework that ensures low service downtime by splitting services into different layers and transferring only the necessary layers at destination host. The experimental results show the benefits of using the proposed approach with LXC containers against the standard KVM migration. Similarly, CloudHopper [60] is a proposed solution for container live migration across cloud providers. CloudHopper, with optimizations, can live migrate a heavy container between cloud providers within 30s while the procedure is completely transparent to the client. Unlike CloudHopper, the work proposed in [12] leverages the layered storage system of Docker containers to improve service migration performance across edge nodes by eliminating unnecessary transfers of portions of the application file system. Although VM/container live migration can be considered a starting point for service migration at the edge, none of the mentioned solutions are proactive, which is the strength of our approach. Our proposed model tries to move static data (heavy data) before the handoff starts so that it can move small data chunks in a "live migration" fashion during the handoff.

C. Data replication in Datacenter

In the last decade, with the rise of the cloud computing paradigm, a lot of research works have studied resource management. Specifically, the management of data in datacenters has followed different paths. The objective of DepSky [61] is to improve availability, integrity and confidentiality of information stored in the cloud through the encryption, encoding, and replication of the data. DepSky achieves these objectives by building a cloud services integration on top of a set of storage clouds, combining Byzantine quorum system protocols, cryptographic secret sharing, erasure codes and the diversity provided by the use of several clouds. Zhang et al. [62] present two online algorithms to minimize the cost of moving a huge amount of data across different datacenters for effective processing using a MapReduce approach. The difference between the above solutions and our work lies in the fact that the objective is not the same, and this work uses data replication as a means to boost offloading performance so that once a task is offloaded to a new EDC, it can be executed right away.

Our work is related to the VM placement problem [63]. Meng et al. [64] propose a VM placement algorithm that aims to improve network scalability. Specifically, the algorithm optimizes the placement of VMs on datacenters in a way that VMs with large mutual bandwidth usage are assigned to host machines in close proximity. Other works like [65] take into account the problem of workload consolidation to minimize energy consumption. Specifically in [65], the authors aim to pack the base station traffic load into fewer number of general purpose processors (VMs) and the problem is formulated as a bin-packing problem (NP-hard). In addition, the VM placement problem is addressed in [66], whereby the objective is to minimize the network congestion within the system. Finally, the closest to our work is [67] where the authors investigated the use of data replication in conjunction with the VM placement problem to decide both on which data should be replicated where and which VM must be migrated so as to minimize the network overhead. The major differences with our work are as follows: i) motivations to have data replicated to different edge datacenters, and ii) the way the data were moved. We claim that having data replicated to different edge datacenters can speed up the handoff process as mentioned in Section II. In addition, by being proactive in the migration process, it is possible to further reduce the migration time.

VII. Discussion

In this paper, we mainly focus on reducing the drop of accuracy for intelligent industrial applications during edge datacenters outages. For this purpose, we presented the MED solution for data handoff and we showed results about inference accuracy during the handoff according to our implementation and realistic dataset. Suppose that another industry plant creates a different dataset or uses a different implementation of the MED algorithm with different parameters. First, we analyzed a dataset that has a linear trend of data without multiple spikes in one or more consecutive packets [68]. In the case of different datasets, according to Fig. 11, packet categories might cease to make sense due to multiple spikes, for instance, multiple consecutive category 2 packets might result more critical in terms of accuracy drop than one miss of category 1 packet during the handoff. Therefore, according to the application, this might change the behavior of the MED algorithm described in Fig. 11. To address such a problem, we have though MED as completely agnostic to datasets and able to react to change given its performance results that make it similar to standalone proactive handoff algorithms. Second, in the experimental results, we considered 100 samples per packet. As explained in Section V, we evaluated which was the correct size of the packet beforehand. However, it creates a trade-off between loss of data and sending frequency. If we send much data, we certainly need to reduce the network overhead, but at the same time we stand to lose lot of data if the handoff occurs and this results eventually in loss of accuracy. Note that the choice of packet size depends also on datasets, namely, linear datasets tend to have more data in a packet compared to discontinuous datasets.

On the contrary, we can consider different parameters of the MED algorithm for instance the circular buffer size. For this case, the buffer size represents the number of historical data (they are considered as the number of file) that we used for calculating the inference. Here, we have a trade-off between accuracy and speed of execution, if we have all files containing all the history we have maximum accuracy. Besides, the inference algorithm will take more time and the handoff will move more data between EDCs. This implies that the system should be dimensioned according to the desiderata accuracy. Finally, MED provides steps for efficient data handoff at the edge, while we cannot provide precise size of the buffer, and consequently we can only achieve an approximate truthfulness and individual rationality.

VIII. Conclusions

In this paper, we focus on efficient data handoff at the edge. To address this problem, we introduce Mobile Edge Data-Handoff (MED) framework that is capable to handle AI-driven I4.0 applications. Specifically, the proposed MED algorithm is focused on migration of offloaded inference tasks of AI pre-trained models in edge computing environments. MED is able to relocate inference from one EDC to another EDC for preventing service outages. In order to evaluate MED, we implemented it in the openLEON emulator and demonstrated its performance with an AI-based Industry 4.0 application that forecasts the gas flow in a chemical plant. The results attest that the MED algorithm performs similarly to standalone proactive handoff algorithms and provides an high level of accuracy during the handoff.

Acknowledgment

Dr. Fiandrino is supported by the Juan de la Cierva Incorporation grant from the Spanish Ministry of Science and Innovation (IJC2019-039885-I). This work was partially supported by the European Union under the Italian National Recovery and Resilience Plan (NRRP) of NextGenerationEU, partnership on "Telecommunications of the Future" (PE00000001 - program "RESTART") CUP: J33C22002880001.

References

[1] M. Xu, Z. Fu, X. Ma, L. Zhang, Y. Li, F. Qian, S. Wang, K. Li, J. Yang, and X. Liu, *From Cloud to Edge: A First Look at Public Edge Platforms*, 2021, p. 37–53.

[2] Y. Su, W. Fan, Y. Liu, and F. Wu, "A truthful combinatorial auction mechanism towards mobile edge computing in industrial internet of things," *IEEE Transactions on Cloud Computing*, pp. 1–1, 2022.

[3] Z. Zhou, X. Chen, E. Li, L. Zeng, K. Luo, and J. Zhang, "Edge intelligence: Paving the last mile of artificial intelligence with Edge computing," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1738–1762, 2019.

[4] T. Qiu, J. Chi, X. Zhou, Z. Ning, M. Atiquzzaman, and D. O. Wu, "Edge computing in industrial internet of things: Architecture, advances and challenges," *IEEE Communications Surveys & Tutorials*, vol. 22, no. 4, pp. 2462–2488, 2020.

[5] M. Uddin, M. S. Kodialam, S. Mukherjee, T. V. Lakshman, and M. Uddin, "GLAMAR: Geo-location assisted mobile augmented reality for industrial automation," in *Proc. of ACM/IEEE Symposium on Edge Computing*, Nov 2020.

[6] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE Communications Surveys Tutorials*, vol. 19, no. 4, pp. 2322–2358, 2017.

[7] L. Lin, X. Liao, H. Jin, and P. Li, "Computation offloading toward edge computing," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1584–1607, 2019.

[8] T. Taleb, K. Samdanis, B. Mada, H. Flinck, S. Dutta, and D. Sabella, "On multi-access edge computing: A survey of the emerging 5G network edge cloud architecture and orchestration," *IEEE Communications Surveys Tutorials*, vol. 19, no. 3, pp. 1657–1681, 2017.

[9] C. Fiandrino, N. Allio, D. Kliazovich, P. Giaccone, and P. Bouvry, "Profiling performance of application partitioning for wearable devices in mobile cloud and fog computing," *IEEE Access*, vol. 7, pp. 12 156–12 166, Jan 2019.

[10] P. Mach and Z. Becvar, "Mobile edge computing: A survey on architecture and computation offloading," *IEEE Communications Surveys Tutorials*, vol. 19, no. 3, pp. 1628–1656, Third Quarter 2017.

[11] T. Taleb, A. Ksentini, and P. A. Frangoudis, "Follow-me cloud: When cloud services follow mobile users," *IEEE Transactions on Cloud Computing*, vol. 7, no. 2, pp. 369–382, 2019.

[12] L. Ma, S. Yi, and Q. Li, "Efficient service handoff across edge servers via docker container migration," in *Proceedings of the Second ACM/IEEE Symposium on Edge Computing*, 2017, pp. 1–13.

[13] D. Scotece, C. Fiandrino, and L. Foschini, "A practical way to handle service migration of ml-based applications in industrial analytics," in *GLOBECOM 2022 - 2022 IEEE Global Communications Conference*, 2022, pp. 123–128.

[14] H. Viswanathan and P. E. Mogensen, "Communications in the 6G era," *IEEE Access*, vol. 8, pp. 57 063–57 074, 2020.

[15] G. Wikström, J. Peisa, P. Rugeland, N. Johansson, S. Parkvall, M. Girnyk, G. Mildh, and I. L. Da Silva, "Challenges and technologies for 6G," in *Proc. of 6G Wireless Summit (6G SUMMIT)*, 2020, pp. 1–5.

[16] S. Wang, J. Wan, D. Zhang, D. Li, and C. Zhang, "Towards smart factory for industry 4.0: a self-organized multi-agent system with big data based feedback and coordination," *Computer Networks*, vol. 101, pp. 158–168, 2016, industrial Technologies and Applications for the Internet of Things. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1389128615005046

[17] C. Sönmez, S. Baghaee, A. Ergişi, and E. Uysal-Biyikoglu, "Age-of-information in practice: Status age measured over tcp/ip connections through wifi, ethernet and lte," in *Proc. of IEEE BlackSeaCom*, 2018, pp. 1–5.

[18] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield, "Live migration of virtual machines," in *Proceedings of the 2nd Conference on Symposium on Networked Systems Design & Implementation - Volume 2*, ser. NSDI'05. USA: USENIX Association, 2005, p. 273–286.

[19] A. Machen, S. Wang, K. K. Leung, B. J. Ko, and T. Salonidis, "Live service migration in mobile edge clouds," *IEEE Wireless Communications*, vol. 25, no. 1, pp. 140–147, 2018.

[20] S. Nadgowda, S. Suneja, N. Bila, and C. Isci, "Voyager: Complete container state migration," in *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, 2017, pp. 2137–2142.

[21] N. Tziritas, M. Koziri, A. Bachtsevani, T. Loukopoulos, G. Stamoulis, S. U. Khan, and C. Xu, "Data replication and virtual machine migrations to mitigate network overhead in edge computing systems," *IEEE Transactions on Sustainable Computing*, vol. 2, no. 4, pp. 320–332, 2017.

[22] T. Z. Emara and J. Z. Huang, "Distributed data strategies to support large-scale data analysis across geo-distributed data centers," *IEEE Access*, vol. 8, pp. 178 526–178 538, 2020.

[23] S. Wang, J. Xu, N. Zhang, and Y. Liu, "A survey on service migration in mobile edge computing," *IEEE Access*, vol. 6, pp. 23 511–23 528, 2018.

[24] Y. Zeng, M. Chao, and R. Stoleru, "EmuEdge: A hybrid emulator for reproducible and realistic edge computing experiments," in *Proc. IEEE International Conference on Fog Computing*, June 2019, pp. 1–6.

[25] R. Mayer, L. Graser, H. Gupta, E. Saurez, and U. Ramachandran, "EmuFog: Extensible and scalable emulation of large-scale fog computing infrastructures," in *Proc. IEEE Fog World Congress*, Oct 2017, pp. 1–6.

[26] J. Hasenburg, M. Grambow, E. Grünewald, S.Huk, and D. Bermbach, "MockFog: Emulating fog computing infrastructure in the cloud," in *Proc. IEEE International Conference on Fog Computing*, June 2019, pp. 1–6.

[27] C. Fiandrino, A. B. Pizarro, P. J. Mateo, C. A. Ramiro, N. Ludant, and J. Widmer, "openLEON: An end-to-end emulation platform from the edge data center to the mobile user," *Computer Communications*, vol. 148, pp. 17 – 26, 2019.

[28] I. Gomez-Miguelez, A. Garcia-Saavedra, P. D. Sutton, P. Serrano, C. Cano, and D. J. Leith, "srsLTE: An open-source platform for LTE evolution and experimentation," in *Proc. of ACM WiNTECH*, 2016, pp. 25–32.

[29] Y. Li, Z. Yuan, and C. Peng, "A control-plane perspective on reducing data access latency in LTE networks," in *Proc. of ACM MobiCom*, 2017, p. 56–69.

[30] Z. Tan, Y. Li, Q. Li, Z. Zhang, Z. Li, and S. Lu, "Supporting mobile VR in LTE networks: How close are we?" *Proc. ACM Meas. Anal. Comput. Syst.*, vol. 2, no. 1, Apr. 2018.

[31] 5G-PPP. White paper: 5G and the factories of the future. [Online]. Available: https://5g-ppp.eu/wp-content/uploads/2014/02/5G-PPP-White-Paper-on-Factories-of-the-Future-Vertical-Sector.pdf

[32] Qualcomm, "Ultra-Reliable Low-Latency 5G for Industrial Automation," https://www.qualcomm.com/media/documents/files/read-the-white-paper-by-heavy-reading.pdf, 2021, [Online; accessed 31-January-2022].

[33] D. Lopez-Perez, A. Garcia-Rodriguez, L. Galati-Giordano, M. Kasslin, and K. Doppler, "IEEE 802.11be extremely high throughput: The next generation of wi-fi technology beyond 802.11ax," *IEEE Communications Magazine*, vol. 57, no. 9, pp. 113–119, 2019.

[34] I. I. Consortium, "Industrial Analytics: the Engine Driving the IIoT Revolution," https://www.iiconsortium.org/pdf/Industrial_Analytics-the_engine_driving_IIoT_revolution_20170321_FINAL.pdf, 2021, [Online; accessed 31-January-2022].

[35] Kubernetes, "Production-Grade Container Orchestration," https://kubernetes.io/, 2022, [Online; accessed 31-January-2022].

[36] P. Bellavista, A. Corradi, L. Foschini, and D. Scotece, "Differentiated service/data migration for edge services leveraging container characteristics," *IEEE Access*, vol. 7, pp. 139 746–139 758, 2019.

[37] I. Kadota, M. S. Rahman, and E. Modiano, "WiFresh: Age-of-information from theory to implementation," in *Proc. of IEEE ICCCN*, 2021, pp. 1–11.

[38] E. Arribas, A. Fernández Anta, D. R. Kowalski, V. Mancuso, M. A. Mosteiro, J. Widmer, and P. W. H. Wong, "Optimizing mmwave wireless backhaul scheduling," *IEEE Transactions on Mobile Computing*, vol. 19, no. 10, pp. 2409–2428, 2020.

[39] A. Mozo, B. Ordozgoiti, and S. Gómez-Canaval, "Forecasting short-term data center network traffic load with convolutional neural networks," *PLOS ONE*, vol. 13, no. 2, pp. 1–31, 02 2018.

[40] J. Chen, V. C. S. Lee, K. Liu, and J. Li, "Efficient cache management for network-coding-assisted data broadcast," *IEEE Transactions on Vehicular Technology*, vol. 66, no. 4, pp. 3361–3375, 2017.

[41] B. Liu, J. Guo, C. Li, and Y. Luo, "Workload forecasting based elastic resource management in edge cloud," *Computers & Industrial Engineering*, vol. 139, p. 106136, 2020.

[42] D. Scotece, C. Fiandrino, and L. Foschini, "On the efficiency of service and data handoff protocols in edge computing systems," in *2021 IEEE Global Communications Conference (GLOBECOM)*, 2021, pp. 1–6.

[43] M. Peuster, J. Kampmeyer, and H. Karl, "Containernet 2.0: A rapid prototyping platform for hybrid service function chains," in *Proc. of IEEE NetSoft*, 2018, pp. 335–337.

[44] STMicroelectronics, "Motion and environmental sensors for industrial applications," https://www.st.com/content/st_com/en/campaigns/industrial_sensors.html, 2021, [Online; accessed 14-June-2021].

[45] N. Budhdev, R. Joshi, P. G. Kannan, M. C. Chan, and T. Mitra, "FSA: Fronthaul slicing architecture for 5G using dataplane programmable switches," in *Proc. of ACM MobiCom*, 2021, p. 723–735. [Online]. Available: https://doi.org/10.1145/3447993.3483247

[46] Ponemon, "Data Center Downtime at the Core and the Edge: A Survey of Frequency, Duration and Attitudes," https://www.vertiv.com/490a6d/globalassets/documents/reports/ponemon/vertiv-ponemon-data centerdowntimesurveyreport_321796_0.pdf, 2021, [Online; accessed 31-October-2021].

[47] UptimeInstitute, "Annual outage analysis 2021: The causes and impacts of data center outages," https://uptimeinstitute.com/annual-outage-analysis-2021, 2021, [Online; accessed 31-October-2021].

[48] Y. Mao, Y. Fu, W. Zheng, L. Cheng, Q. Liu, and D. Tao, "Speculative container scheduling for deep learning applications in a kubernetes cluster," *IEEE Systems Journal*, vol. 16, no. 3, pp. 3770–3781, 2022.

[49] A. Yousefpour, G. Ishigaki, R. Gour, and J. P. Jue, "On reducing iot service delay via fog offloading," *IEEE Internet of Things Journal*, vol. 5, no. 2, pp. 998–1010, 2018.

[50] S. Mubeen, P. Nikolaidis, A. Didic, H. Pei-Breivold, K. Sandström, and M. Behnam, "Delay mitigation in offloaded cloud controllers in industrial iot," *IEEE Access*, vol. 5, pp. 4418–4430, 2017.

[51] X. Tao, K. Ota, M. Dong, H. Qi, and K. Li, "Performance guaranteed computation offloading for mobile-edge cloud computing," *IEEE Wireless Communications Letters*, vol. 6, no. 6, pp. 774–777, 2017.

[52] C. You, K. Huang, H. Chae, and B. Kim, "Energy-efficient resource allocation for mobile-edge computation offloading," *IEEE Transactions on Wireless Communications*, vol. 16, no. 3, pp. 1397–1411, 2017.

[53] J. Zhang, W. Xia, F. Yan, and L. Shen, "Joint computation offloading and resource allocation optimization in heterogeneous networks with mobile edge computing," *IEEE Access*, vol. 6, pp. 19 324–19 337, 2018.

[54] P. Zhou, B. Finley, X. Li, S. Tarkoma, J. Kangasharju, M. Ammar, and P. Hui, "5G MEC computation handoff for mobile augmented reality," 2021.

[55] K. Kaur, S. Garg, G. S. Aujla, N. Kumar, J. J. P. C. Rodrigues, and M. Guizani, "Edge computing in the industrial internet of things environment: Software-defined-networks-based edge-cloud interplay," *IEEE Communications Magazine*, vol. 56, no. 2, pp. 44–51, 2018.

[56] H. Guo and J. Liu, "Collaborative computation offloading for multiaccess edge computing over fiber–wireless networks," *IEEE Transactions on Vehicular Technology*, vol. 67, no. 5, pp. 4514–4526, 2018.

[57] K. Ha, Y. Abe, Z. Chen, W. Hu, B. Amos, P. Pillai, and M. Satyanarayanan, "Adaptive VM handoff across cloudlets." Technical Report CMU-CS-15-113, CMU School of Computer Science, Tech. Rep., 2015.

[58] M. R. Hines, U. Deshpande, and K. Gopalan, "Post-copy live migration of virtual machines," *ACM SIGOPS Oper. Syst. Rev.*, vol. 43, no. 3, p. 14–26, Jul. 2009.

[59] K. Ha, Y. Abe, T. Eiszler, Z. Chen, W. Hu, B. Amos, R. Upadhyaya, P. Pillai, and M. Satyanarayanan, "You can teach elephants to dance: Agile VM handoff for edge computing," in *Proc. of ACM/IEEE SEC*, Oct. 2017, p. 12.

[60] T. Benjaponpitak, M. Karakate, and K. Sripanidkulchai, "Enabling live migration of containerized applications across clouds," in *Proc. of IEEE INFOCOM*, 2020, pp. 2529–2538.

[61] A. Bessani, M. Correia, B. Quaresma, F. André, and P. Sousa, "Depsky: Dependable and secure storage in a cloud-of-clouds," *ACM Trans. Storage*, vol. 9, no. 4, Nov. 2013.

[62] L. Zhang, C. Wu, Z. Li, C. Guo, M. Chen, and F. C. M. Lau, "Moving big data to the cloud," in *Proc. of IEEE INFOCOM*, 2013, pp. 405–409.

[63] Z. Usmani and S. Singh, "A survey of virtual machine placement techniques in a cloud data center," *Procedia Computer Science*, vol. 78, pp. 491 – 498, 2016.

[64] X. Meng, V. Pappas, and L. Zhang, "Improving the scalability of data center networks with traffic-aware virtual machine placement," in *Proc. of IEEE INFOCOM*, 2010, pp. 1–9.
[65] T. Sigwele, A. S. Alam, P. Pillai, and Y. F. Hu, "Energy-efficient cloud radio access networks by cloud based workload consolidation for 5G," *J. Netw. Comput. Appl.*, vol. 78, p. 1–8, Jan. 2017.
[66] J. W. Jiang, T. Lan, S. Ha, M. Chen, and M. Chiang, "Joint VM placement and routing for data center traffic engineering," in *Proc. of IEEE INFOCOM*, 2012, pp. 2876–2880.
[67] N. Tziritas, M. Koziri, A. Bachtsevani, T. Loukopoulos, G. Stamoulis, S. U. Khan, and C. Xu, "Data replication and virtual machine migrations to mitigate network overhead in edge computing systems," *IEEE Transactions on Sustainable Computing*, vol. 2, no. 4, pp. 320–332, 2017.
[68] J. Burgués, J. M. Jiménez-Soto, and S. Marco, "Estimation of the limit of detection in semiconductor gas sensors through linearized calibration models," *Analytica chimica acta*, vol. 1013, pp. 13–25, 2018.

BIOGRAPHIES

Domenico Scotece is a junior assistant professor at the University of Bologna, Italy, right after having obtained the Ph.D degree at the same University in April 2020. He received the Master Degree in Computer Science Engineering from the University of Bologna, in 2014. His research interests include pervasive computing, middleware for fog and edge computing, the Software-Defined Networking, the Internet of Things, and management of cloud computing systems.

Claudio Fiandrino is a senior researcher at IMDEA Networks Institute. He obtained his Ph.D. degree at the University of Luxembourg in 2016. Claudio has received numerous awards for his research, including a Fulbright scholarship in 2022, a 5-year long Spanish Juan de la Cierva grants and several Best Paper Awards. He is member of IEEE and ACM, serves in the Technical Program Committee (TPC) of several international IEEE and ACM conferences and regularly participates in the organization of events. Claudio is member of the Editorial Board of IEEE NETWORKING LETTERS and is Chair of the IEEE ComSoc EMEA Awards Committee.

Luca Foschini received the Ph.D. degree in computer science engineering from the University of Bologna, Italy, in 2007, where he is currently an Associate Professor of computer engineering. His interests span from integrated management of distributed systems and services to edge computing, from management of cloud computing systems to Industry 4.0. He is Director of the IEEE ComSoc EMEA Board.