Received September 2, 2019, accepted September 9, 2019, date of publication September 26, 2019, date of current version October 7, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2943848

Differentiated Service/Data Migration for Edge Services Leveraging Container Characteristics

PAOLO BELLAVISTA, (Senior Member, IEEE), ANTONIO CORRADI, (Senior Member, IEEE), LUCA FOSCHINI, (Senior Member, IEEE), AND DOMENICO SCOTECE, (Student Member, IEEE)
Dipartimento di Informatica: Scienza e Ingegneria (DISI), University of Bologna, 40136 Bologna, Italy

Corresponding author: Domenico Scotece (domenico.scotece@unibo.it)

ABSTRACT The Multi-access Edge Computing (MEC) and Fog Computing paradigms are enabling the opportunity to have middleboxes either statically or dynamically deployed at network edges acting as local proxies with virtualized resources for supporting and enhancing service provisioning in edge localities. However, migration of edge-enabled services poses significant challenges in the edge computing environment. In this paper, we propose an edge computing platform architecture that supports service migration with different options of granularity (either entire service/data migration, or proactive application-aware data migration) across heterogeneous edge devices (either MEC-based servers or resource-poor Fog devices) that host virtualized resources (Docker Containers). The most innovative elements of the technical contribution of our work include i) the possibility to select either an application-agnostic or an application-aware approach, ii) the possibility to choose the appropriate application-aware approach (e.g., based on data access frequencies), iii) an automatic edge services placement support with the aim of finding a more effective placement with low energy consumption, and iv) the in-lab experimentation of the performance achieved over rapidly deployable environments with resource-limited edges such as Raspberry Pi devices.

INDEX TERMS Container migration, Docker containers, edge computing, service migration.

I. INTRODUCTION

The rapid increase in demand for mobile devices within the realms of real-time mobile applications, augmented reality, and mobile gaming, and Industry 4.0 (just to cite a few) motivate the need for real-time mobile cloud applications. Necessarily, these real-time applications require low latencies to provide seamless end-user interaction imposing very strict Quality of Service (QoS) requirements. For instance, cloud-based multimedia real-time applications require end-to-end latencies below 60ms and much lower values within specific contexts such as the industrial one [1], [2]. The only way to comply with those requirements is moving cloud computing to the edge of the network, so to lessen these otherwise unacceptable delays by locally providing needed resources.

Accordingly, in the last years various models and solutions have been proposed by academia and industry, such as Cloudlet [3], Follow Me Cloud [4], and Micro Data Center (MDC). All these solutions share the idea of interposing an intermediate layer of middleboxes, either statically or dynamically deployed at the network edge, between the leaf device layer and the global cloud. The primary idea is to have services in proximity of mobile users so that network congestion is reduced, battery life is enhanced, and service experience is improved in terms of Quality of Experience (QoE) and QoS. In particular, this intermediate layer provides storage, computation, and network resources enabling the possibility of moving and hosting services at the edge of the network (e.g., offloading), to decrease latency and to increase scalability through local interactions, whenever possible.

More recently, along the same direction, Multi-access Edge Computing (MEC) [5] and Fog Computing [6] have become prominent concepts in many recent studies and technologies [7]. The differences between these models are mainly in terms of deployment and administrative management. In MEC, the infrastructure resides at the edge of a telco operator infrastructure and MEC nodes are typically MDCs. Instead, the Fog infrastructure and all its derivatives reside on premises and at the edge of end-system infrastructure,

The associate editor coordinating the review of this manuscript and approving it for publication was Christos Verikoukis.

typically at home gateway level. Moreover, Fog nodes can be resource-constrained devices such as general-purpose routers and/or single-board computers including Raspberry Pi. In the reminder of this work, we use Edge Computing as a general term to refer to all above emerging models and, especially to MEC and Fog Computing.

Edge Computing has helped to significantly reduce the delays between mobile nodes and cloud, and it is considered one of the enabler technologies for the 5G vision. At the same time, it has introduced new problems such as the management of users' mobility [8]. Generally, edge nodes have small network coverage and it is very common that a mobile node changes its connection during its path. Due to limited coverage of a single edge node, user mobility could affect the degradation of network performance reducing the QoS or in some cases causing the interruption of the edge service. Thus, in order to guarantee service continuity, it is necessary to support efficient service/data migration between edge nodes. However, at the current stage there are several heterogeneous virtualization technologies and migration strategies and some of them might not be practical when used with resource-poor edge devices (e.g. Raspberry Pi). Moreover, future 5G networks will be composed of heterogeneous devices, such as home gateways and MEC micro-servers that do not host the same resources. That makes service migration management a very complex task; for instance, heavy-computation services should be migrated to high powerful micro-servers rather than to poor Fog gateway nodes. Energy consumption should also be considered, especially for battery-power edge nodes.

Focusing on existing solutions, most seminal efforts have been focused on the concept of Live Migration of Virtual Machines (VMs) [9] to guarantee the lowest possible downtime of the service. Live migration considers service migration as the stateful migration of services where the service contains internal state data of the user. After the completion of the migration, the service resumes exactly where it had stopped before. To ensure this, complex mechanisms have been proposed including the main two variants called pre-copy and post-copy. Pre-copy pushes most of the data to destination host before stopping and migrating the VM [9]. Post-copy pulls most of the data from source host after resuming VM at the destination host [10].

Successively, with the diffusion of container technologies such as Docker [11], most of the research efforts have been focused on service container migration. This is justified by several experiments conducted to compare the performance of VMs and Containers [12]. In [13], performance evaluations are carried out to compare standard VM hypervisors and container-based virtualization technologies for edge-based IoT applications. Since containers are a more lightweight virtualization option, several companies started using them to develop applications. Today, containers are largely used for edge-based services due to their adaptive characteristics including lightness and portability. Nevertheless, only very few proposals have started to target the technological advances associated with container migration in place of VM. Some proposal tried to use container virtualization technology as the object of the live migration (container state migration). Other efforts, instead, have focused on service migration by leveraging Docker technology.

At the current stage, Edge Computing does not have a standardized fast service/data migration support. To tackle this problem, we focus on reducing the total time of service migration by leveraging the service characteristics. We show how to design a fast handoff schema that exploits the knowledge of the characteristics of the service. At the heart of our schema, there is Docker which allows the separation between data and service containers. Moreover, our schema profiles data characteristics and leverages that awareness in order to reduce the migration time. Finally, most of the existing researches about Edge Computing focus on offloading tasks to edge servers in order to save energy for mobile devices. However, only saving energy for mobile devices is not enough because the energy consumption of edge servers is non-negligible. To address this issue, we propose an edge services placement solution able to reduce energy consumption. In particular, the proposed work has the following primary innovation elements and features, which we claim provide a non-negligible contribution to the advancement of the literature in the field.

First, it enables either application-agnostic or application-aware approaches for container migration. In the application-agnostic mode, our framework can perform migration without having any visibility of the application behavior (just container migration). Dually, in the application-aware case, the framework can fully exploit application specific knowledge to determine which data should be migrated proactively in order to minimize latency and to optimize the usage of possibly limited resources, e.g., inter-edge bandwidth and edge storage. Second, an efficient way to take advantage of data characteristics has been investigated to reduce application-aware migration times. We have developed a Decision Module that contains a set of mechanisms for carrying out application-aware migration based on data change probability. It associates to data with lower access frequencies that can be proactively moved, while it postpones the migration of data with higher access frequencies. Third, our framework also guarantees data consistency between edge nodes after the handoff: if some data moved proactively to the new edge node have been changed during the handoff period, our framework automatically reconciles those data. Last, our framework addresses the problems of heterogeneity and energy management at edge nodes by defining an affinity relationship that depends on service characteristics. This solution allows our framework to decide the best target node toward which to perform the handoff in terms of needed resources and energy consumption. We consider as edge node both resource-poor devices, including Raspberry Pi acting as Fog nodes and powerful computers used as MEC node.

The remainder of the paper is structured as follows. The next two sections provide first an overview of related works, then report background material and propose design guidelines needed to fully understand our original proposal.

Section IV describes the architecture of our solution and details our novel protocols for fast service handoff management at the edge. Section V shows a wide range of performance results including both in-the-field experimentation and CloudSim simulations. Conclusive remarks, and directions of ongoing related work end the paper.

II. RELATED WORK

Service migration at the edge of the network and container live migration have been heavily investigated in recent years. In this section, we discuss two main research directions that are related to our proposal: i) service migration based on VMs at the edge, and ii) container service migration.

A. SERVICE MIGRATION AT THE EDGE

In the past few years, notable efforts of researches have been focused on the benefits and challenges of Edge Computing. One of the uncleared challenges is service migration, which guarantees service continuity as users move across different edge nodes. Ha *et al.* [14] proposed Cloudlet, as one of the seminal examples of computing at the network, and a mechanism for edge-enabled handoff management based on VM service synthesis and migration to the newly visited edge nodes. This has led to a few important middleware solutions to address service migration in the presence of user mobility. Mobile Micro-Cloud (MMC) [15] started exploring the idea to place micro-clouds closer to end-users. In that work, authors faced out the service migration problem by taking into account the costs associated with running service at the same MMC server and the costs associated with migrating the service to another MMC server. To do this, the authors define an algorithm to predict the future costs for finding the optimal placement of services. Another important work in the same context is Follow-Me Cloud [4] that enables mobile cloud services to follow mobile users alongside datacenters. The framework allows service migration by migrating all or portions of services to the optimal data center. Service migration decision is based on user constraints and network conditions.

Some other recent proposals are based specifically on VM migration. Preliminary research efforts focused on the impact on network performance [16]. To overcome this limitation, various VM migration systems exploit the concept of live migration optimized for the edge computing. Live migration is mostly identifying as a technique for VM migration in datacenters at the cloud layer. Indeed, datacenters are assumed to be stable environments with high-bandwidth data paths always available. Most important solutions are based on pre-copy approaches, where VM control is not transferred to the destination until all VM state has been copied. On the other hand, post-copy approaches resume VM at the destination first and then the state is retrieved [10].

Ha *et al.* [14] highlight the limitations of traditional live VM migration on edge devices and propose live migration in response to client handoff in cloudlets, with less involvement of the hypervisor and by promoting migration to optimal offload sites, adapting to changing network conditions and processing capacity. The same authors also proposed a mechanism called VM handoff that supports agility for cloudlet-based applications [17]. The mechanism preserves the core properties of VM live migration for data center while optimizing for the agile environment of Edge Computing. This approach leverages on pipelined stages that aim at reducing the differences between the VM state at the source and VM state at the destination.

B. CONTAINER SERVICE MIGRATION

Containers differ from VMs technology since they directly share the hardware and the kernel with their host machines. As a result, containers occupy fewer resources and have lower virtualization overhead than VMs. For this reason, container migration has started to be a very active area that has not been systematically studied in the literature yet. Machen *et al.* [18] investigate live migration of LXC containers [19] by proposing a three-layer framework with synchronized filesystem methodology for memory state sync. Substantially, that work shows a quantitative view on the difference between LXC containers migration and KVM [20] migration.

Live migration of containers become possible since CRIU [21] supports checkpoint/restore functionalities for the most container solutions such as OpenVZ [22], LXC/LXD, and Docker [11]. Several solutions have been explored in the literature that leverage CRIU for migrating stateful containers. OpenVZ supports live migration of containers [23]; however, it exploits Virtuozzo Storage System [24], that is a distributed storage system where all files are shared across the network. In most cases, the network bandwidth of edge servers is limited, and the deployment of a distributed storage could be not possible [25]. Moreover, the implementation is not optimized due to the transfer of root filesystem of the container across edge nodes. IBM proposes Voyager [26], a live container migration service designed in accordance with the Open Container Initiative (OCI) principles: the IBM solution implements a novel filesystem-agnostic and vendor-agnostic migration service with consistency guarantees.

Summarizing the related work, although a few solutions have been proposed to contribute to the field of container migration in Edge Computing, there is no ready solution that exploits the characteristics of the application. As a consequence, we present our solution to make container migration faster and easier than existing proposals. In addition, we claim that the paper provides a significant contribution to the community because, to the best of our knowledge, this is the first system-oriented work on MEC/Fog handoff that leverages the service characteristics.

III. BACKGROUND AND DESIGN GUIDELINES

To better understand our original proposal, this section provides all needed definitions for the involved technologies and methodologies and an overview of our proposed design guidelines.

A. MULTI-LAYER CONTAINER MIGRATION AND RELATED BACKGROUND

The main objective of our proposal is to develop a novel framework that enables fast service migration for edge-enabled service by leveraging its characteristics. For this purpose, we distinguish between so-called layered services and monolithic services. Layered services consist of diverse layers, such as service and data parts, that could be managed as different separated blocks. Differently, monolithic services have to be considered by our framework as a single block, which internally includes service components, data components, and all associated resources. To better understand layered services, let us describe a simple example: a simple Python web application represents the service part, while a Redis database in which data are stored represents the data part. In this way, the service part and the data part could be managed as distinct blocks. On the contrary, monolithic services can be put into execution within dedicated and self-contained VMs as proposed in some recent literature: for instance, [3] have proposed a mechanism for edge-enabled handoff management based on VMs synthesis and migration to the closest edge nodes. However, the usage of VMs introduces non-negligible latency and overhead due to VM size and complexity. The exploitation of container-based virtualization techniques could reduce the above weaknesses, by enabling the opportunity of considering more layered services that can be decomposed in various microservices.

We present a novel approach for multi-layer container-based service migration by leveraging service characteristics. To achieve this, edge-enabled services in our proposal are built as Docker Containers composed by a *service layer* (acting as the "business logic" part of the service) and a *data layer* (representing the state stored and managed through the service layer) that should be managed as separate containers. The Data Container is used to persist data and could be managed by either a DBMS or a NoSQL manager. We followed a general approach able to work also with no-database-based storage including general filesystems. In addition, our proposal is able to support two kinds of service migrations: *application-agnostic* and *application-aware*. Application-agnostic handoff enables the migration of the entire Data Container, as the data backup, without requiring any previous knowledge of the specific data software layer technology. Application-aware, instead, leverages service characteristics to extract and proactively transfer part of data to the target edge node in order to reduce service interruption. However, the latter mode requires partial visibility of some characteristics of the implementation of the Data Container, and for this reason, it is not usable for any kind of application.

As mentioned in the previous section, Docker technology doesn't provide any official migration tool, but, recently, few developers have constructed tools for specific versions of Docker. For instance, the work [25] supports Docker Containers migration of docker version 1.9.0, and Boucher [27] extends the previous work to support docker-1.10 migration. However, both methods simply transfer all the files located under the mount point of the container root file system. More recently, also Docker provided some mechanisms not directly related to the migration but useful for this purpose. For instance, *docker export* command enables users to create a compressed file from the container filesystem as a "tar" file. This compressed file can be copied over the network to the target edge node via file transfer and then imported into a new container via *docker import* command. The new container created in the target edge node can be accessed using *docker run* command. One drawback of docker export tool is that it doesn't copy environment variables and underlying data volume which contains the container data. Another method based on Docker commands to move the container to another host is container image migration. For the container that has to be moved, its corresponding Docker image is saved into a compressed file by using *docker commit* command. Then compressed file is moved to the target edge node and a new container is created with *docker run* command. Using this method, the data volume will not be migrated, but it preserves the data of the application created inside the container. Last, Docker provides a mechanism to save an image to a tarball which preserves the history, layers, and entrypoints via *docker save* command; at the same time, it provides the equivalent command to load the image in the new host: *docker load*.

TABLE 1. Docker containers migration time (bandwidth 40mb/s, latency 0ms, and delay 0ms).

Application	Down Time	Total Time	Total Size
Busybox	1,85 s	2,16 s	1,4 MB
Face Recognition	205,61 s	222,89 s	~1 GB

Unfortunately, aforementioned methods completely ignore the composition of the service which we claim could pave the way for smarter migration management. To verify our claim, we have conducted preliminary experiments to migrate containers over different network connections. The experiments use one simple container such as Busybox and one application based on OpenCV for face recognition, to conduct edge task offloading. Busybox is a software suite that provides several Unix utilities in a single executable file. It has a tiny file system inside the container. Instead, the face recognition application is an application that dispatches video streaming from mobile devices to the edge server, which executes the face recognition tasks, and sends back a specific frame with the name of the person. This container hosts a large filesystem to store all the images (i.e., more than 1 GB).

Table 1 reports obtained preliminary results that show that migration can be done within 2 seconds for Busybox, and within 223 seconds for face recognition application. The network between these two hosts is a Wi-Fi connection with

TABLE 2. Docker containers migration time (bandwidth 10mb/s, latency 1.5ms, and delay 40ms).

Application	Down Time	Total Time	Total Size
Busybox	4,78 s	12,11 s	1,4 MB
Face Recognition	>1200 s	>1500 s	~1 GB

40 MB/s bandwidth, and further tested container migration over a 10 MB/s Wi-Fi network.

As previously stated, poor performance is caused by transferring large files comprising the complete file system, for instance the total migration time of face recognition application (Table 2). This performs worse than the state-of-the-art VM migration solution. Migration of VMs cloud avoid transferring a portion of the filesystem by sharing the base VM images [3], which will finish migration within few minutes. Therefore, we require a new tool to efficiently migrate Docker Containers, avoiding transmission of the entire container. This new tool should leverage the characteristics and composition of edge-enable services to transfer proactively part of the service data.

B. DESIGN GUIDELINES FOR APPLICATION-AWARE HANDOFF AND FOR HETEROGENEITY MANAGEMENT

In the following, we formally define a way to select chunks of data to be moved proactively. Mainly, we select data that have not changed lately, and then we propose a framework that is able to proactively move that selected data. Finally, we describe how our framework faces the problem of device heterogeneity and energy consumption.

1) APPLICATION-AWARE STRATEGY

Our goal is to enable proactive, transparent migration of edge-enabled services (typically represented by both the data part and the service part). The idea of this method is based on the observation that not all data or records are used all the time. In fact, from a recent study [28], it was found that most data or records are stored, but rarely or never accessed after a certain time frame. Therefore, data or records can be categorized according to their access frequencies: least accessed data (cold data) and most accessed data (hot data). In our solution, we define a probability of data migration according to the data access frequencies, means cold data are more inclined to be part of the proactive migration process. To do this, we first need to introduce an operation meter that, for each data or records chunk, calculates the total number of operations did until certain time. The operation meter is defined as:

$$O_k = I_k(t) + U_k(t) \tag{1}$$

where O_k denotes the total number of operations did on particular data or record chunk (k) which is defined as the sum of the number of *insert* operations (I_k) and the number of *update* operations (U_k). Hence, by repeating the ahead formula (1) for all data it is possible to obtain the value of access frequencies defined as:

$$f_k = \frac{O_k}{\sum_i^n O_i} \tag{2}$$

where f_k represents the access frequencies of data or record chunk defined as the relationship between operations did on chunk k and the number of total operations did in all data. Finally, we define the migration probability assigned to each data chunk k as:

$$P(x) = \frac{1}{f} \tag{3}$$

As expressed in (3), the migration probability is defined as the inverse of the access frequencies. This means that data accessed often have a low value of migration probability, while data rarely accessed have a high value of migration probability. Thus, the goal of our work is to get the maximum benefit from the data characteristics in order to reduce the overhead and the service interruption during the handoff procedure. This requires us to calculate the access frequencies, as well as the migration probability before the handoff happens. Let us note that the decision on when, where and whether to perform the migration depends on many aspects, such as user mobility, user historical paths, resource availability at the edge nodes, and so on. Our Prediction Module guarantees the right execution of our service migration. The module is composed by two components: monitoring and trigger. The monitoring component monitors users' location in order to predict their movement. Several monitoring strategies have been proposed in the literature, and we design the Prediction Module to work with any such strategy. The trigger component is in charge of determining the appropriate time to initiate the handoff (both long- and short-term). We have identified two distinct handoff triggering strategies: a coarse-grained model (long-term), and a fine-grained model (short-term).

Coarse-Grained Model: Typically, services running at the edge server have a limited period of validity, ranging from few minutes to few hours. The goal of this model is to predict well in advance the user movement in order to calculate data to be moved from one edge node to another. Most long-term handoff triggering algorithms proposed in the literature have taken into account both QoS of application and users' mobility traces. However, if users' historical path is available, the system can proactively predict the handoff timing and can early move service and data from one edge to another.

Fine-Grained Model: The goal of this model is to establish with high accuracy when the handoff happens. In general, short-term handoff triggering algorithms are based on monitoring wireless indicators, such as Received Signal Strength Indications (RSSI) [29]. Otherwise, there are other works that take into account the QoS of application such as TCP throughput [30].

Our framework works with both long- and short-term handoff prediction. When a long-term strategy predicts the handoff, the Prediction Module notifies our Decision Module that starts to calculate the migration probability for each data chunk. In order to choose which data move proactively, we defined a probability threshold, statically or dynamically determined, in which our framework migrates all data chunks that have a probability value greater than the threshold. The appropriate value of the threshold is chosen based on the variability characteristics of the data. For instance, if the data have a high value of variability it would be better to use a high value of probability threshold (e.g., around 0.9-0.95). That's because, using a low value of probability threshold may cause problems in the reconciliation phase, in the sense that early migrated data chunks may result changed after the handoff procedure is completed. Instead, if the data have a low value of variability could be convenient to use a low value of probability threshold. Let us note that the correct value of the probability threshold may be decided dynamically -on the fly- in relation to the data characteristics. Once defined a proper value of the probability threshold, when the Prediction Module predicts the handoff the Decision Module starts to migrate all data chunks that satisfy the threshold condition. Finally, when the handoff occurs our framework migrates all remaining data chunks (data reconciliation phase – step 9" Fig. 4) and then checks the data integrity.

Instead, when a short-term strategy predicts the handoff, as specified before, fine-grained models detect the handoff more precisely than coarse-grained models. Thus, the system has better accuracy but less time to operate. For this reason, could be not possible to send all selected data chunks from one edge to another before the handoff happens; to avoid this, we have adopted a strategy named "sequential execution" that starts to sequentially migrate data chunks with the highest value of the migration probability until the handoff happens. Finally, when the handoff happens our framework migrates all remaining data chunks.

Regardless of the model, once the handoff terminates the framework has to guarantee data consistency. To ensure this, our framework checks if data chunks sent proactively have changed during the handoff. If some data chunks differ, the framework reconciles them. To correctly check if all data chunks sent proactively are consistent after the handoff, our framework sends hash values of each data chunk, before and after the handoff, and checks if these hash values correspond. If the hash value of some data chunks does not correspond, we must resend those chunks.

2) HETEROGENEITY AND ENERGY

As stated in Section I, our framework addresses the problem of edge node heterogeneity by defining an affinity relationship between the service and the edge node. In general, edge nodes are heterogeneous devices spanning from powerful micro data servers (typical for MEC infrastructure) to general-purpose resource-poor gateways (typical for Fog Computing infrastructure). Therefore, if more than one edge node is available in a certain zone, would it be convenient to select the best target edge node according to both resource needs and energy consumption considerations. Our affinity relationship guides the system to take this decision.

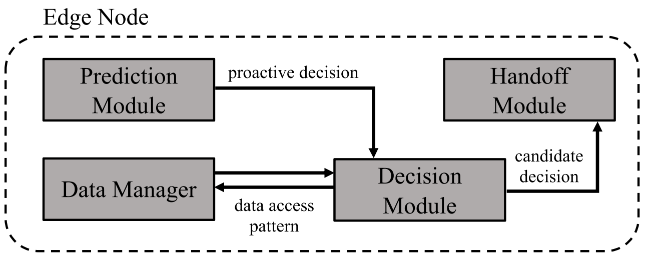

FIGURE 1. System architecture.

In the edge computing environment, we can have many types of affinity, among them: Communication Affinity (CA), Resources Affinity (RA), and Energy Affinity (EA).

CA depends on communication technologies between the mobile node and the target edge node. RA is derived from resources needed for executing the service at the edge node. EA is induced from the minimum resources needed to run the service and resource availability at the edge node.

Regardless of various affinity types, our work denotes affinity of edge services as a key factor for allocation of edge services at the target edge node and takes Energy Affinity as an example. Let us consider a basic scenario that comprises a large number of edge nodes available in an edge environment. The edges are distributed in several zones and are different from each other (some MEC-based other Fog-based). Each edge node has a resource capacity to run a specific service. We aim at minimizing the number of resources needed to run the service in order to save energy in an edge infrastructure.

In this scenario, we describe the Energy Affinity parameter as follows. Given the resources consumption of the running service at the old edge node (in terms of CPU and RAM consumption), we find the optimal allocation of the service by comparing with resources available at the edge nodes. Therefore, the best association is when EA is about 1.

IV. ARCHITECTURE AND HANDOFF MANAGEMENT

This section presents our system architecture and describes our handoff protocols including reactive handoff, proactive handoff, and application-aware handoff.

A. ARCHITECTURE

Fig. 1 shows the architecture of our edge node that consists of a set of components that are deployed at the service layer and enable our handoffs process.

- **Prediction Module (PM)** is in charge of determining the appropriate time to initiate the handoff by monitoring users' location in order to predict their movement. Several monitoring strategies have been proposed in the literature, and we design the PM to work with any such strategy. Particularly, this module collects information about users and calculate several metrics to trigger the Decision Module and to start the handoff process. We claim the importance of distinguishing two kinds of mobility prediction: a coarse-grained historical-based mobility prediction and a fine-grained RSSI-based mobility prediction. The first one is based on the history of user movements: edge nodes, possibly coordinating also with the mobile node to gather mobility traces (such as GPS positions) and the global cloud layer (to process those traces), track user movement to enable long-term predictions of user mobility habits, such as during working days, during weekend, and so forth. Moreover, when it is enabled allows the system to execute proactively long-running operations such as migration of (static) service parts towards the target edge. The second one, namely, fine-grained mobility prediction, evaluates handoff decision by using the value of edge-to-mobile RSSI by employing the monitored RSSI values obtained through heterogeneous short-range wireless technologies, such as Wi-Fi and Bluetooth. As widely recognized in the literature, this kind of prediction is expensive, typically works on shorter time intervals, but gives more accurate information about when to trigger handoff and consequently the migration of more dynamic data parts. Finally, the availability of both prediction modes enables higher flexibility for handoff management, as better explained in the next subsection.

- **Data Container Manager (DcM)** enables the application-aware handoff by embedding the application-specific knowledge to manage finer grained data migration. In other words, this module observes the underlying data container by providing several data connector and returns data to be migrated based on the migration strategy.
- **Decision Module (DM)** contains a set of strategies which are used to determine data mobility. Several strategies can be used for this purpose; in this work, we propose an approach based on data access frequencies where data accessed less have a higher migration probability. Moreover, this module is in charge of choosing the best place (MEC node or Fog node if there are multiple edge nodes in the same region) to forward the handoff procedure by evaluating the service affinity relationship.
- **Handoff Module (HM)** executes the handoff process. This component offers a set of common APIs that enable the interactions of our handoff protocol between all involved distributed entities. This module relates to the same HM of the target edge node and contains all handoff steps to perform such as handoff request, start, stop, and so on.

B. HANDOFF MANAGEMENT

1) REACTIVE HANDOFF

Fig. 2 depicts the primary steps of the baseline (reactive handoff based on Docker tools) of default Docker Containers migration. Generally, the reactive handoff procedure starts when the mobile node loses connection with the old edge node and sends a *handoff request* message to the target edge node (step 1). Upon the old edge node receives the handoff

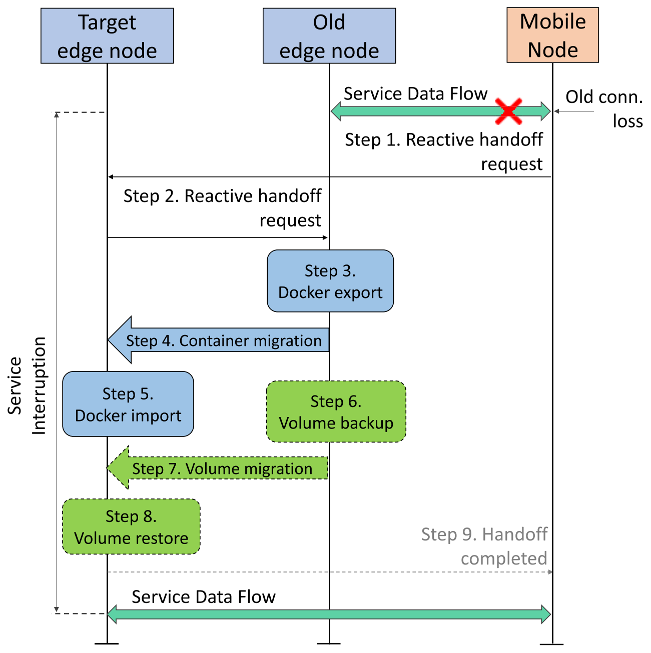

FIGURE 2. Docker basic reactive handoff.

request (step 2), it starts the migration process by exporting the container to be migrated by using Docker export command (step 3). Then, the old node sends the compressed container to the target edge node via network file transfer (step 4). Once the target edge node has received the compressed container it restarts it via docker import command (step 5). In parallel, the old edge node starts to prepare the backup of Data Container, if necessary and sends it to the target edge node which restores it (steps 6-7-8). Finally, the handoff procedure ends (step 9).

Let us clarify that Fig. 2 describes the basic Docker migration protocol (reactive handoff), which impose the service interruption from step 1 to step 9 typically suffered by monolithic services and VMs as well. Note that, to better understand our proposal, we implemented the basic Docker migration protocol also as a baseline to use for comparison in our experimental evaluation (see experimental evaluation section). Of course, our framework would allow to optimize also this reactive handoff management, e.g., leveraging service/data software layering to avoid migrations (if needed layers are already available at the target edge) and, similarly, applying application-aware data management if possible.

The next section, focuses on our proposed protocols that implement and enhance a proactive approach in order to reduce the service interruption interval due to user handoff between different edges, including pre-loading of all selected services/data software layers at edge nodes. In this case, we also distinguish different types of handoff management improvements depending on how data state is transferred in application-agnostic and application-aware cases.

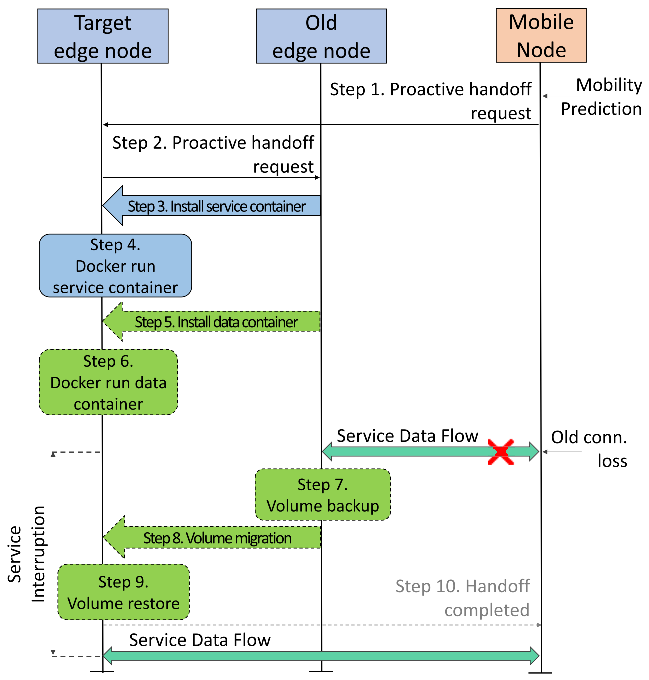
Target edge node
Old edge node
Mobile Node
Step 1. Proactive handoff request
Mobility Prediction
Step 2. Proactive handoff request
Step 3. Install service container
Step 4. Docker run service container
Step 5. Install data container
Step 6. Docker run data container
Service Data Flow
Old conn. loss
Step 7. Volume backup
Step 8. Volume migration
Step 9. Volume restore
Step 10. Handoff completed
Service Data Flow
Service Interruption

FIGURE 3. Docker proactive handoff.

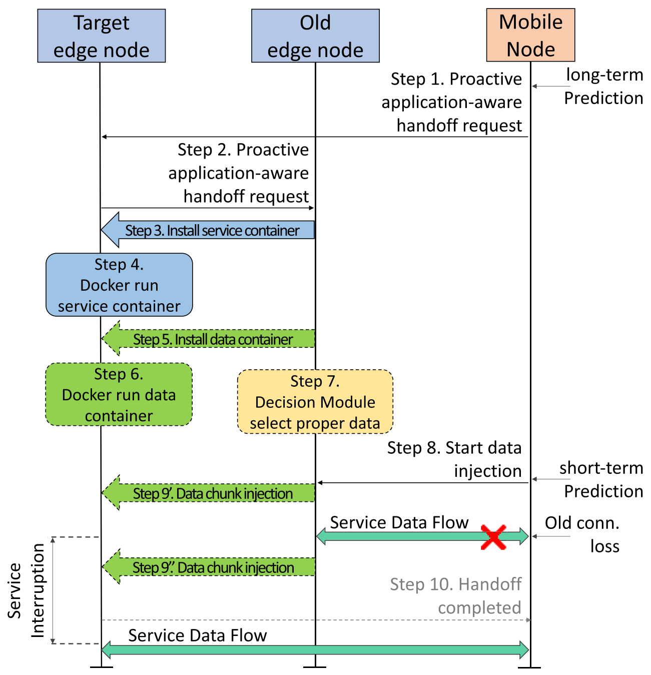
Target edge node
Old edge node
Mobile Node
Step 1. Proactive application-aware handoff request
long-term Prediction
Step 2. Proactive application-aware handoff request
Step 3. Install service container
Step 4. Docker run service container
Step 5. Install data container
Step 6. Docker run data container
Step 7. Decision Module select proper data
Step 8. Start data injection
short-term Prediction
Step 9'. Data chunk injection
Service Data Flow
Old conn. loss
Step 9''. Data chunk injection
Step 10. Handoff completed
Service Data Flow
Service Interruption

FIGURE 4. Docker proactive application-aware handoff.

2) PROACTIVE HANDOFF

This subsection describes the design principles behind our service/data migration protocols, detailing also the application-aware optimization. We start by presenting our optimized proactive application-agnostic handoff procedure and then we detail our application-aware handoff strategies.

Fig. 3 shows our optimizations of basic reactive handoff based on Docker tools. The optimizations leverage both long- and short-term predictions to enable proactive provisioning.

Our handoff procedure begins when the Prediction Module predicts the migration and triggers the proactive execution of the handoff procedure (steps 1-2). Then, the protocol starts the migration of the service part (steps 3-4) and the installation of the data part (steps 5-6). In this approach, we consider the data part as a black box with no information about its inner characteristics; in the next optimizations, we describe the additional modalities of operation of Decision Module which can operate also the application-aware handoff. Therefore, steps 5-6 install only the data container while we postpone the request for data backup migration until the mobile node loses connection from the old edge node, so to make sure to receive a more consistent data state, with all changes made at the old edge node (step 7). Once completed the data container backup, the old edge node starts to send the backup to the target edge node (step 8) and then the target edge node restores the data backup with all latest changes made at the old edge node by the user (step 9). Finally, the target edge node sends the *handoff complete* signal to the mobile node.

As one can see from Fig. 3, this approach of service/data migration decreases the service interruption time compared to the previous one (Docker basic reactive handoff). Let us note that this does not imply any application-specific knowledge and requirements to perform it. Furthermore, we can further reduce the service interruption time leveraging the Decision Module that contains decision mechanisms to move proper data. This is the core of our proposal and we will explain it in detail in the next subsection.

3) APPLICATION-AWARE HANDOFF

The idea behind application-aware optimizations is the exploitation of our Decision Module; thus, beyond the strategies, the Decision Module selects proper data to be moved proactively to the target edge node.

In our solution, the application-aware service/data migration process is composed by multiple steps, as depicted in Fig. 4. The user's mobility is observed by Prediction Module that can activate a trigger when the user mobile node is likely to go towards the new edge node. The Prediction Module can take advantage of both short- and long-term user's mobility prediction; for the purpose of application-aware optimizations, we need to use a long-term user's mobility prediction in order to select and move data proactively (steps 1-2). Let us note that compared to the Docker proactive application-agnostic handoff, application-aware handoff allows us to proactively move certain data to the target edge node. Thus, the old edge node migrates the service (steps 3-4) and the data container part (steps 5-6) to the target edge node. Then, in order to select proper data to be moved, based on the migration strategy, we need to invoke the Decision

Module (step 7). In this time interval, the mobile node continues to be provisioned through the old edge node. Consequently, our procedure introduces a periodic data reconciliation phase, triggered by a short-term mobility prediction (steps 8-9'), to reduce the service interruption interval, by limiting the whole data state migration to those data chunks. This periodic data injection phase (managed by our Decision Module by using its strategies) terminates when the mobile node loses the connection with the old edge node: our protocol guarantees data consistency by sending another data chunk update from the old edge node to the target edge node, and that completes the whole handoff procedure (steps 9''-10).

V. EXPERIMENTAL EVALUATION AND SIMULATION WORK

As already stated, one of the key contributions of this paper is that our service/data migration solution has been implemented and completely integrated into the a real MEC/Fog architecture. As a valuable side-effect, differently from seminal efforts available in the existing literature, we are able to report results obtained in our lab deployment scenario, with heterogeneous edge devices, and a simulation work for additional quantitative evaluations and comparisons. To thoroughly test and evaluate the performance of our framework, we carried out three different sets of experiments, respectively, for Docker basic reactive handoff, for our application-agnostic proactive handoff, and for our application-aware proactive handoff. The results reported in this section are average values; all presented measurements have exhibited a limited variance (under 5% for 30 runs).

A. REAL IN LAB TESTBED EXPERIMENTAL MEASUREMENTS

To better understand improvements in terms of system complexity and migration time, we quickly introduce our in-lab deployment scenario. Our evaluation testbed consists of three Linux boxes (Ubuntu 18.04 distribution): two 3.06GHz Intel(R) CORE i5 and 8GB 1300 MHz DDR3 memory as MEC-based edge nodes, and one Raspberry Pi3 equipped with 64-bit quadcore ARM Cortex-A53 processor, 1 GB of RAM and 16 GB of storage as a Fog-enabled node. Due to space constraints, we present a case with heterogeneous edge devices where the old edge node is a micro datacenter and the target edge node is a fog node. This because we want to highlight one of the critical cases of our work. Those nodes host Docker 18.09-ce and Java 12, and all our framework components. During our experiments, we have considered the service migration performance of our framework for a specific cloud- edge-enabled application based on Docker Compose [31] defined by the Docker Compose yml file as follow:

```
version: '3'
    services:
        java:
            build: .
            ports:
            - "8080:8080"
        mongo:
            image: mongo
            volumes_from:
                - mongo: dbdata
```

This is the general schema used for implementing multi-layering Docker-based applications. Our test service consists of a Java web application defined as the service layer and an instance of MongoDB as the data layer. The test service consists of a web-based application where users report some information, they find along the road such as obstacles, restaurants, and groceries. In particular, the Java part provides a simple human interface which users compile and insert information through a form that are stored in the MongoDB database.

The MongoDB container is linked to another container (dbdata) that acts as Docker Volume [32] via the *volumes_from* Docker primitive. Let us recall that in Docker, a Volume is a mechanism for persisting data in the local filesystem used by a Docker container. To create a Docker container for persisting data, it is possible to use the following dockercli command:

```
docker create -v /data/db --name dbdata mongo
                /bin/true
```

The proposed characterization of our test service helps us to better understand how application-aware handoff works. Indeed, the data layer (composed by a MongoDB instance) is physically separated from the rest of the service, which means that our framework optimizations can exclusively focus on this part. MongoDB has been chosen for its simplicity and because it provides mechanisms that allow us to implement each step of our application-aware handoff. In particular, MongoDB assembles the data in the form of collections which represent our data chunks. Finally, MongoDB provides mechanisms for sending and restoring only portions of data (i.e., data chunk) by using mongodump and mongorestore integrated tools.

Unless otherwise specified, edge nodes connect each other via IEEE 802.11n connections and their maximum nominal available bandwidth is 40 Mbit/s. During the first set of experiments (Docker container migration), the persistent layer to migrate is around 300 MB until 330 MB depending on the different number of records (from 10K to 100K). Let us clarify that we considered the service already installed at the target edge node, hence we need to migrate only the data container. The handoff process starts when the mobile node loses connection with the old edge node. Hence, the total time of migration is obtained by the sum of three different steps: export container, send container, and restore container.

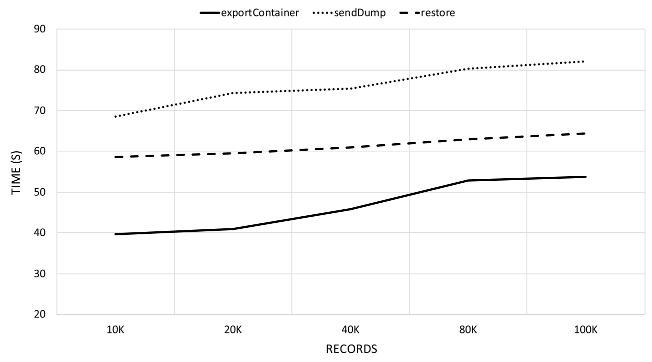

FIGURE 5. **Docker basic handoff total migration time.**

The first step (step 6 Fig. 2), export container, is done at the old edge node and consists in collect all container files into one tar archive file. The second step allows the system to transfer the tar through the network from the old edge node to the target edge node. We have run the test several times by changing the amounts of data stored on MongoDB, from 10K to 100K records. Fig. 5 shows the total migration time for different amounts of data by highlighting the time needed to complete each step. As depicted in Fig. 5, the number of records affects the total migration time by a factor of around 10 s per each stage. In the worst case, the overall service interruption is around 170 s.

The remaining sets of experiments are related to the more interesting proactive scenario. We have evaluated the proposed application-aware approach in terms of total migration time. In this scenario, when the Prediction Module foresees the handoff (long-term), the Decision Module at the old edge node starts to calculate data to migrated (cold data) and migrates the data toward the target edge node. Then, when the handoff happens, the system sends only the remaining data (hot data) towards the destination edge node. Finally, the destination edge node has to check the consistency of all cold data (some data may have changed value during the handoff). For this set of experiments, we set the number of cold blocks at 35% of the total blocks. Each block in our implementation correspond to a MongoDB collection; in order to calculate the migration probability of each block, we used the collection stats command (*db.collection.stats()*), provided by mongo API, that returns statistics about the collection including the total number of insert and update operations. We simulated different percentages of a correct guess that means the correctness of the forecast made on the cold data. The different simulated percentages of correct guess are: 25%, 50%, 75%, and 100%. If the forecast on the cold data is incorrect, we need to resend all cold blocks that do not match. Fig. 6 shows the performance of the container migration for different amounts of data at different percentages of a correct guess. On the one hand, when the percentage of correct guess increases the total migration time decreases. On the other hand, when more records need to be processed the total migration time increases.

B. SIMULATION RESULTS ABOUT TOTAL MIGRATION TIME AND DATA LOSS COMPARED WITH DATA VARIABILITY

For additional quantitative evaluations and comparisons, we employed CloudSim [33], an extensible and widely adopted simulation toolkit that enables the modeling and simulation of cloud computing environments. In particular, CloudSim simulation framework supports the modeling and creation of infrastructures and application environments for distributed multiple clouds. A recent extension of CloudSim, named EdgeCloudSim [34], builds the concept of Edge Computing upon CloudSim by adding necessary functionalities in terms of computation and network capabilities. In particular, we map the framework into the simulator by creating:

- Two micro datacenters, used to migrate our service to;
- one host per datacenter, with 2GB RAM and 250GB storage each;
- two VMs for each host, with 512MB RAM, 100GB storage, and 1 CPU each;
- one process per VM representing MongoDB instance.

In this simulated environment, we extensively compared our application-aware solution with two baseline approaches, such as reactive migration and proactive migration. The reactive migration adopts the approach of migrating all data at once when the handoff happens. Thus, it is characterized by high migration time (because it sends all data), and also may cause significant data loss in case of high amount of data received during the migration process. The proactive approach, instead, moves the data in advance before the handoff happens according to the migration probability. We simulated different values of data variability and migration probability in order to show how migration time and data loss vary. Fig. 7 and 8 show, respectively, the results about the total migration time and data loss in relation to the data variability for the reactive and the proactive migration. The total migration time for the reactive migration always remains the same regardless of data variability value, that is so because the reactive migration ever sends all data. The same does not apply to data loss. If we have a high value of data variability, a long interruption of the service (caused by the migration), may generate a high value of data loss, because a mobile device can still use the service at the old edge node during the handoff. The situation is different when we analyzed the proactive migration. Let us note that the results reported in Fig. 8 have been obtained by simulating a migration of data container with size 200MB, and migration probability at 0.7. Then, the figure shows how the total migration time depends on the choice of the migration probability. Therefore, if the system has more than 1kB/s of data variability rate, the choice to have 0.7 as migration probability does not lead to any benefits. In other words, the results in Fig. 7 and 8 highlight the relevance of being able to dynamically adapt the migration behavior to expected data variability, as in our proposed framework where we use the migration probability as a threshold to decide whether or not to move data.

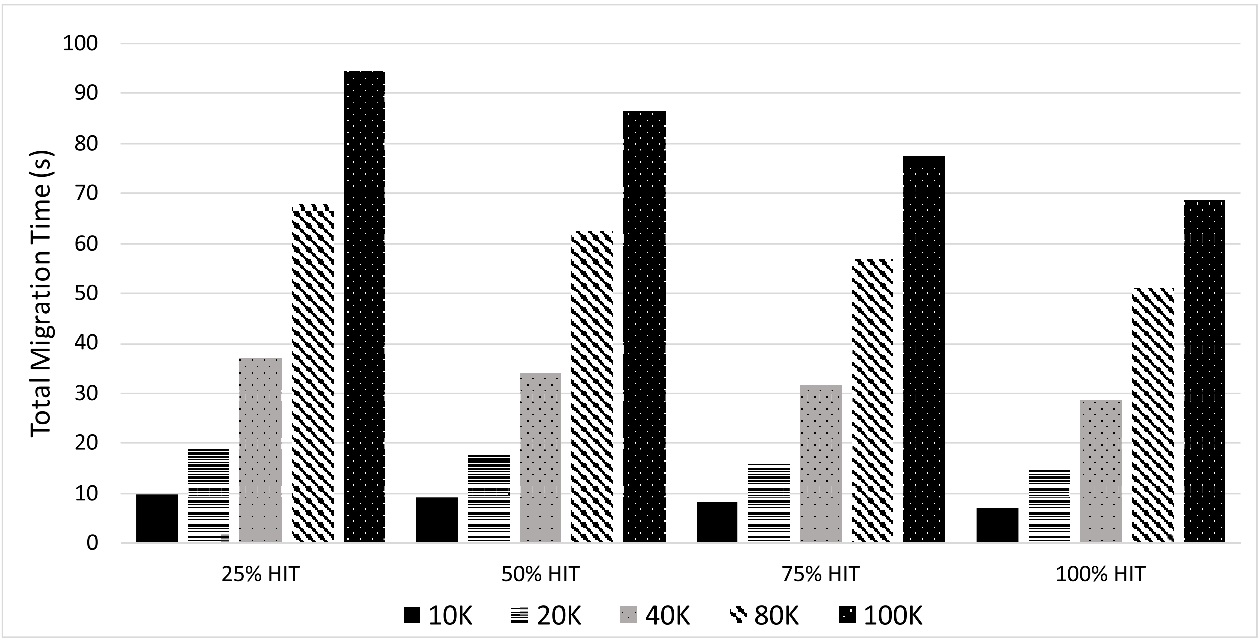

FIGURE 6. Docker basic handoff total migration time.

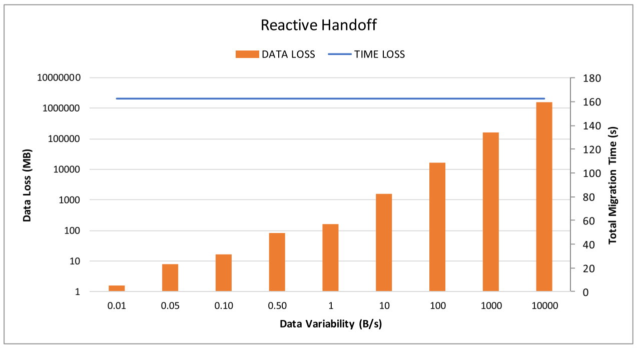

FIGURE 7. Total migration time for reactive handoff.

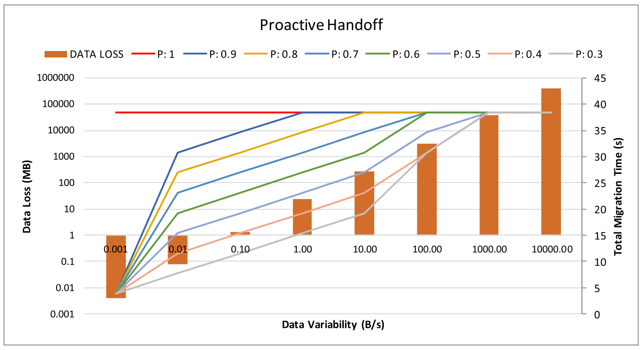

FIGURE 9. Total migration time relates to migration probability.

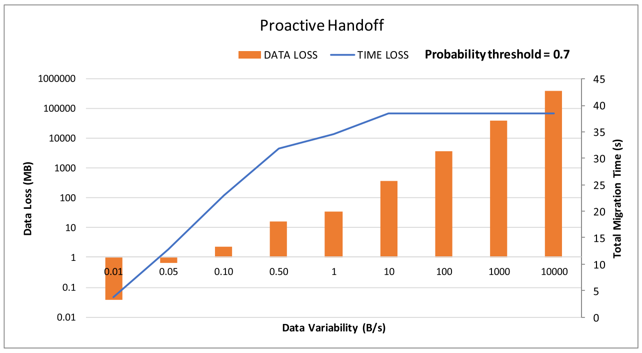

FIGURE 8. Total migration time for proactive handoff.

Finally, Fig. 9 reports about how we have modeled the migration probability in our simulations, by showing how the total migration time changes in relation to the migration probability and the data variability rate. Indeed, the figure represents a general model to choose the more suitable migration probability value in relation to how quickly data records change. With the simulation results, we want to give a baseline guide to choose the best value of migration probability related to the data variability if available.

VI. CONCLUSION AND ONGOING WORK

The proposed framework supports the mobility of edge-enabled services in a three-layer edge computing environment. In particular, our support works either in application-agnostic mode and application-aware mode (if possible), and it manages the heterogeneity of the edge environment. We have already validated our approach both via real experiments and using simulations with synthetic values of data variability. The reported results confirm that proactive migration adopted can significantly minimize the service downtime in the case of layered services (total migration time reductions of 30% $\sim$ 50%), by imposing a very limited overhead on the overall support infrastructure.

A future implementation of the proposed framework may involve different software components, not only related to virtualization technologies, such as filesystems. In this case, our framework should be able to realize which portion of data can be moved proactively towards the target edge node.

Finally, fueled by these significant results, we are working on two main ongoing research directions. On the one hand, we are deploying the realized solution, already widely tested in the geographically distributed Edge Computing testbed, in a federated cloud environment with heterogeneous devices. On the other hand, we are running extensive experiments to thoroughly assess the impact of our framework, to mitigate the potentially disruptive effect on other concurrent ongoing migration sessions.

REFERENCES

[1] B. Taylor, Y. Abe, and A. Dey, *Virtual Machines for Remote Computing: Measuring the User Experience*. Pittsburgh, PA, USA: Carnegie Mellon Univ., 2015.

[2] P.-V. Mekikis, K. Ramantas, L. Sanabria-Russo, J. Serra, A. Antonopoulos, D. Pubill, E. Kartsakli, and C. Verikoukis, "NFV-enabled experimental platform for 5G tactile Internet support in industrial environments," *IEEE Trans. Ind. Informat.*, to be published.

[3] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The case for VM-based cloudlets in mobile computing," *IEEE Pervasive Comput.*, vol. 8, no. 4, pp. 14–23, Oct./Dec. 2009.

[4] T. Taleb and A. Ksentini, "Follow me cloud: Interworking federated clouds and distributed mobile networks," *IEEE Netw.*, vol. 27, no. 5, pp. 12–19, Sep./Oct. 2013.

[5] *ETSI's Multi-Access Edge Computing White Paper*. Accessed: Jul. 30, 2019. [Online]. Available: https://portal.etsi.org/Portals/0/TBpages/MEC/Docs/Mobileedge_Computing_-_Introductory_Technical_White_Paper_V1%2018-09-14.pdf

[6] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the Internet of Things," in *Proc. 1st Ed. MCC Workshop Mobile Cloud Comput.*, Aug. 2012, pp. 13–16.

[7] S. Wang, X. Zhang, Y. Zhang, L. Wang, J. Yang, and W. Wang, "A survey on mobile edge networks: Convergence of computing, caching and communications," *IEEE Access*, vol. 5, pp. 6757–6779, 2017.

[8] I. Sarigiannis, E. Kartsakli, K. Ramantas, A. Antonopoulos, and C. Verikoukis, "Application and network VNF migration in a MEC-enabled 5G architecture," in *Proc. IEEE 23rd Int. Workshop Comput. Aided Modeling Design Commun. Links Netw. (CAMAD)*, Barcelona, Spain, Sep. 2018, pp. 1–6.

[9] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, and A. Warfield, "Live migration of virtual machines," in *Proc. 2nd Conf. Symp. Netw. Syst. Design Implement.*, vol. 2, 2005, pp. 273–286.

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

[11] *Docker*. Accessed: Jul. 30, 2019. [Online]. Available: https://www.docker.com/

[12] W. Felter, A. Ferreira, R. Rajamony, and J. Rubio, "An updated performance comparison of virtual machines and Linux containers," in *Proc. IEEE Int. Symp. Perform. Anal. Syst. Softw. (ISPASS)*, Philadelphia, PA, USA, Mar. 2015, pp. 171–172.

[13] A. Tosatto, P. Ruiu, and A. Attanasio, "Container-based orchestration in cloud: State of the art and challenges," in *Proc. 9th Int. Conf. Complex, Intell., Softw. Intensive Syst.*, Blumenau, Brazil, Jul. 2015, pp. 70–75.

[14] K. Ha, Y. Abe, Z. Chen, W. Hu, B. Amos, P. Pillai, and M. Satyanarayanan, "Adaptive VM handoff across cloudlets," Comput. Sci. Dept., Carnegie Mellon Univ., Ar-Rayyan, Qatar, Tech. Rep. CMU-CS-15-113, Jun. 2015.

[15] S. Wang, R. Urgaonkar, T. He, K. Chan, M. Zafer, and K. K. Leung, "Dynamic service placement for mobile micro-clouds with predicted future costs," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 4, pp. 1002–1016, Apr. 2017.

[16] W. Cerroni and F. Callegati, "Live migration of virtual network functions in cloud-based edge networks," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Sydney, NSW, Australia, Jun. 2014, pp. 2963–2968.

[17] 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. 2nd ACM/IEEE Symp. Edge Comput.*, Oct. 2017, p. 12.

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

[19] D. Lezcano. *LXC—Linux Containers*. Accessed: Jul. 30, 2019. [Online]. Available: https://github.com/lxc/lxc

[20] *Kernel Virtual Machine (KVM)*. Accessed: Jul. 30, 2019. [Online]. Available: https://www.linux-kvm.org/page/Main_Page

[21] *Checkpoint/Restore in Userspace, or CRIU*. Accessed: Jul. 30, 2019. [Online]. Available: https://www.criu.org/Main_Page

[22] *Open Source Container-Based Virtualization for Linux OpenVZ*. Accessed: Jul. 30, 2019. [Online]. Available: https://openvz.org/

[23] P. Haul. *Container Live Migration*. Accessed: Jul. 30, 2019. [Online]. Available: https://criu.org/P.Haul

[24] *Virtuozzo Storage*. Accessed: Jul. 30, 2019. [Online]. Available: https://openvz.org/Virtuozzo_Storage

[25] P. Emelyanov. *Live Migration Using CRIU*. Accessed: Jul. 30, 2019. [Online]. Available: https://github.com/xemul/p.haul

[26] S. Nadgowda, S. Suneja, N. Bila, and C. Isci, "Voyager: Complete Container State Migration," in *Proc. IEEE 37th Int. Conf. Distrib. Comput. Syst. (ICDCS)*, Atlanta, GA, USA, Jun. 2017, pp. 2137–2142.

[27] R. Boucher. *Live Migration Using CRIU*. Accessed: Jul. 30, 2019. [Online]. Available: https://github.com/boucher/p.haul

[28] M. H. Rahman, F. B. Al Abid, M. N. Zaman, and M. N. Akhtar, "Optimizing and enhancing performance of database engine using data clustering technique," in *Proc. Int. Conf. Adv. Electr. Eng. (ICAEE)*, Dhaka, Bangladesh, Dec. 2015, pp. 198–201.

[29] Y. Ge, Z. Zheng, B. Yan, J. Yang, Y. Yang, and H. Meng, "An RSSI-based localization method with outlier suppress for wireless sensor networks," in *Proc. 2nd IEEE Int. Conf. Comput. Commun. (ICCC)*, Oct. 2016, pp. 2235–2239.

[30] J. Kikuchi, C. Wu, Y. Ji, and T. Murase, "Mobile edge computing based VM migration for QoS improvement," in *Proc. IEEE 6th Global Conf. Consum. Electron. (GCCE)*, Nagoya, Japan, Oct. 2017, pp. 1–5.

[31] Docker Inc. *Docker Compose*. Accessed: Jul. 30, 2019. [Online]. Available: https://docs.docker.com/compose/

[32] Docker Inc. *Docker Volume*. Accessed: Jul. 30, 2019. [Online]. Available: https://docs.docker.com/storage/volumes/

[33] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose, and R. Buyya, "CloudSim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Softw., Pract. Exper.*, vol. 41, no. 1, pp. 23–50, Jan. 2011.

[34] C. Sonmez, A. Ozgovde, and C. Ersoy, "EdgeCloudSim: An environment for performance evaluation of edge computing systems," *Trans. Emerg. Telecommun. Technol.*, vol. 29, no. 11, Nov. 2018, Art. no. e3493.

PAOLO BELLAVISTA (SM'06) received the Ph.D. degree in computer science engineering from the University of Bologna, Italy, in 2001, where he is currently a Full Professor. His research interests include mobile agent-based middleware solutions, and pervasive wireless computing to location/context-aware services and management of cloud systems. He serves on the Editorial Board of the IEEE TNSM, IEEE TSC, Elsevier PMC, Springer WINET, and Springer JNSM.

ANTONIO CORRADI (SM'19) graduated from the University of Bologna, Italy, and received the M.S. degree in electrical engineering from Cornell University, USA. He is currently a Full Professor of computer engineering with the University of Bologna. His research interests include distributed systems, middleware for pervasive and heterogeneous computing, and infrastructure for services and network management.

LUCA FOSCHINI (SM'19) 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 include pervasive wireless computing environments, system and service management, edge computing, Industry 4.0, and management of cloud computing systems.

DOMENICO SCOTECE graduated from the University of Bologna, Italy, in 2014, where he is currently pursuing the Ph.D. degree in computer science engineering. His research interests include pervasive computing, middleware for fog and edge computing, the Internet of Things, and management of cloud computing systems.