Journal of Ubiquitous Systems & Pervasive Networks
Volume 18, No. 1 (2023) pp. 39-48

Hybrid-MiGrror: An Extension to the Hybrid Live Migration to Support Mobility in Edge Computing **

Arshin Rezazadeh [a] *, Davood Abednezhad [b], Hanan Lutfiyya [a]

[a] *Computer Science Department, Western University, London ON N6A 3K7 Canada*
[b] *Information and Communications Technology, Khouzestan Oxin Steel Company, Ahvaz, Iran*

** (Invited Paper) Data from this extended paper were partially presented and published in the proceedings of the 17th international conference on future networks and communications (FNC) [1].

Abstract

User-Equipments (UEs) capable of working with cloud computing have grown exponentially in recent years, leading to a significant increase in the amount of data production. Moreover, upcoming Internet-of-Things (IoT) applications such as virtual and augmented reality, video streaming, intelligent transportation, and healthcare will require low latency, communications, and processing. *Edge computing* is a revolutionary criterion in which dispersed edge nodes supply resources near end devices because of the limited resources available on UEs. Rather than transmitting massive amounts of data to the cloud, edge nodes could filter, analyze, and process the data they receive using local resources. Mobile Edge Computing (MEC), in particular, when user mobility is considered, has the potential to significantly reduce processing delays and network traffic between UEs and servers. This research demonstrated a novel technique for migration that minimizes delay and downtime by utilizing edge computing. Our proposed method syncs more frequently than the pre-copy method which is the most used migration method that synchronizes (sync) the source and destination only based on multiple rounds. When compared to established migration methodologies, our results indicate that our mechanism has less latency, downtime, migration time, and packet loss. These results allow delay-sensitive applications that require ultra-low latency to function smoothly during migration.

Keywords: *delay (latency), mobile edge computing (MEC), downtime, hand-off (handover), live migration, fog computing.*

1. Introduction

Consider Internet-of-Things (IoT) apps that monitor the sensors of a physical object. Data must be processed in real-time or near real-time if the application has strict Quality of Service (QoS) and Quality of Experience (QoE) requirements. Accessing cloud resources may require multiple hops. The latency in communicating data from data sources to the cloud may not be timely enough for a delay-sensitive application that requires real-time responses [2-4].

Fog computing was proposed with the goal to distribute computing, storage, control and networking services at the network edge, where data is generated [5]. This reduces the latency since fewer hops are required to transfer data. However, complications occur if a mobile User Equipment (UE) moves, and hence it may move away from the node that is hosting a service with which it is communicating with. Real-time video streaming and conferencing, face recognition, online gaming, augmented reality (AR), and virtual reality (VR) are examples of applications [6] impacted by mobility. Mobile Edge Computing (MEC) envisions that fog computing resources are provided at the edge of the network [7]. Services can be deployed as a virtual machine (VM) or as a container that is placed on a fog node. Migrating VM/containers that encapsulate a service between edge nodes can be used to deal with UE mobility.

Migration in MEC typically refers to the process of moving a running virtual machine (VM) or a container from the current edge node to either an edge node or cloud without disconnecting applications [8]. Hand-off is a component of migration [6] and is triggered when a device disconnects from an edge node's access point (AP) and connects to the AP of another edge node. VM/container migration typically results in downtime ranging from a few seconds to a few minutes [6, 9-

* Corresponding author. Tel.: +15196613566
Fax: +15196613515; E-mail: arezaza6@uwo.ca

DOI: 10.5383/JUSPN.18.01.006

17]. This downtime accounts for a significant portion of the delay [18, 19]. Furthermore, the UE is unable to access services and data during hand-off because it must relocate from the previous connection point to the next. Their downtimes are typically greater than one second, which negatively affects latency, QoS, and QoE requirements, while low latency and real-time apps, such as some AR applications, are in demand. An AR application in a head-mounted device, for example, requires less than 17 millisecond *(ms)* of end-to-end delay to function smoothly [20], whereas most research delays and downtimes exceed this number and cannot support real-time responses. These findings highlight the need for a new approach to further reduce delay and downtime for modern applications, which could also be integrated into most existing techniques.

Efforts have been made to reduce VM/container migration downtime when a UE hands off from one edge node to another. Data transfers are required when migrating VMs or containers. Recent research has focused on reducing the amount of data transferred during hand-off using different metrics such as runtime and offline characteristics. Their research focuses on predicting the best time to trigger a hand-off and improving the selection of edge nodes to allow for shorter processing times [9-11, 21]. These approaches mostly use live migration approaches borrowed from cloud computing, which uses the pre-copy migration technique, to reduce migration downtime [6, 9-17, 21].

During migration, the pre-copy method transfers data from the source to the destination in predefined rounds [22], whereas our proposed approach synchronizes data between source and destination more frequently as data at the source changes. The main advantage of our proposed technique is that more frequent synchronization results in less data transfer during hand-off.

Most researchers use the pre-copy method, while the work described in this paper focuses on an alternative to pre-copy that improves the performance of the migration method by reducing downtime when handover is triggered.

The rest of this paper is organized as follows: Section 2 provides background information on traditional migration techniques and related work. Section 3 discusses the design of our approach to edge infrastructure. Section 4 evaluates the proposed method, and Section 5 concludes this paper.

2. Background and related work

2.1. Background: Fundamental Migration Techniques in Mobile Edge Computing

Stateless migration occurs when the state (which may include CPU, register, signal, and memory states) of the services of the users is not saved; otherwise, it is stateful migration [10, 22]. The primary focus of this paper is on stateful migration techniques. In this subsection, we provide a sketch of the most fundamental approaches to stateful migration. We will use these methods for comparison and evaluation in section IV.

Cold migration: The VM/container execution is halted at the source. The VM/container is then transferred to the destination and resumes as soon as the VM/container becomes available at the destination. Apps hosted on the UE cannot access their service during this time until the VM/container restarts execution at the new location. Cold migration produces a lengthy downtime compared to live migration methods [22]) since live migration methods enable VMs/containers to continue running during most of the migration process [22, 23]. These techniques are described below.

Post-copy migration: This live migration technique first freezes the VM/container to stop run-time state modification and then transfers the latest state to the destination. The VM/container continues operating at the target while the remainder of the latest state is being transferred. The VM/container is connected to the destination while still reading data from the source until the state transfer is completed [22].

Pre-copy migration: With this live migration technique, the entire VM/container state from the source to the destination is transferred. It then transmits modified memory pages, called dirty pages, over several iterations. It later stops the VM/container execution at the source node to copy the last dirty page to the destination. Finally, the VM/container continues execution at the destination [22]. Pre-copy requires more data transfer than post-copy during migration since dirty pages must be periodically sent before hand-off. However, post-copy causes longer delays because it still has some data in the source that must be read from the destination until the state transfer is completed [22].

Hybrid-copy migration: Both pre-copy and post-copy methods, as previously discussed, have drawbacks: (i) Non-deterministic downtime occurs during the pre-copy phase; (ii) service performance during the post-copy stage is affected by faulted pages [22]. The first stage of hybrid-copy migration is identical to pre-copy migration, sending the entire state and then dirty pages to the destination while the virtual machine or container remains operational on the origin [24, 25]. The VM/container is then paused **after the hand-off is triggered**, and its state is transmitted. The VM/container can be restarted at the target when the state and memory have been delivered. The most recent VM/container's execution state and memory pages are now present in the target. However, pages may have been dirtied throughout the pre-copy process. Accordingly, the final phase of the hybrid method is to transfer dirty pages to the target using the post-copy method [26]. Since the hybrid technique sends only the dirty pages after the hand-off, it typically transfers fewer memory pages than post-copy. According to the abovementioned research findings, the hybrid technique outperforms both pre- and post-copy migrations.

2.2. Migration Strategies in Mobile Edge Computing

Researchers employ various strategies to reduce migration time. In recent years the focus has been on various prediction-based, e.g., [9-11, 21] and, e.g., data transfer [6, 12-17] approaches that are used to reduce migration time. The data transfer approach papers focus on reducing the amount of data transferred during hand-off to fundamentally reduce downtime, resulting in significantly reduced latency and migration time and, as a result, user QoE. While prediction-based methods reduce downtime, latency, and migration time when compared to other ML techniques, they do not fundamentally reduce them. In the following, we will provide a brief description of each paper. Current research focuses on prediction-based approaches or data transfer approaches.

Data Transfer: To reduce transfer size during hand-off, Ma et al. [6, 15] proposed an improved migration technique based on the hierarchical structure of the container file system. Transferring the basic image only at the beginning of the migration, followed by iterative memory difference, can help reduce transfer size during hand-off. In the best-case scenario, they had 2.7 seconds of downtime. Machen et al. [14] proposed a layered migration framework that supports container and virtual machine technologies. The framework breaks the

application down into various layers and transfers only the missing layers to the destination. This layered method can reduce downtime by transmitting less data from the source to the destination during hand-off. They assessed their performance using various applications, such as video streaming and gaming servers. They significantly lowered overall migration times but with a 2-second average downtime for a blank container. This time is still unacceptable for delay-sensitive applications, such as the stated head-mounted AR application. Farris et al. [16] used the pre-copy technique in stateless migration to achieve low latency by transferring as little data as possible during hand-off. They send data ahead of time before handing off to achieve lower latency. In the best-case scenario, their experiments had more than a second latency. To reduce downtime and migration time, Addad et al. [12] use memory, partial, and full migration strategies with predefined and non-predefined paths, as well as different numbers of pre-copy iterations. They accomplish this by storing the container files in a shared storage pool accessible to all edge nodes and proactively sending data, resulting in less data to transfer during hand-off. They evaluate their performance using video streaming and blank containers, but they have more than one second of downtime in their best-case scenario. Most papers on reducing downtime and delays aim to reduce transfer time during hand-off by reducing data transfer during that period. On the other hand, Zhou et al. [13] propose a hardware accelerator concept to expedite data transmission reduction computations and, as a result, service migration. They achieve about 300 *ms* of downtime in their best-case scenario. Puliafito et al. [17] stated that an AR application that uses a smart helmet should have a maximum end-to-end latency of 20 *ms*. This study used pre-copy and proactively sent data using compression before hand-off, resulting in less data transfer during hand-off. Despite their proposal, they still have 3.67 seconds of downtime in the best-case scenario.

Prediction-Based Approaches: Handover can be triggered when the value of a metric such as the Received Signal Strength Indicator (RSSI) falls below a preset threshold value. The ideal threshold value would provide sufficiently for the migration process to occur with minimal disruption of service. However, there are other factors to be considered, e.g., the load on the wireless links and the speed of the UE. This makes it difficult to determine a fixed threshold value. A poor threshold value results in the handover being done too late or too early. Prediction-based approaches consider multiple metrics as well as a prediction of the UE's movement to determine the migration destination. For example, Ngo et al. [11] used the pre-copy method and proactively sent data prior to hand-off by memory checkpointing before and during the migration phase in order to determine the time to trigger hand-off and the destination edge node. The downtime period and migration start simultaneously in their work, unlike other work where the migration starts earlier than the downtime. The downtown period in this work covers the handover time, which represents the time that the service is unavailable, but it also includes a period of time when the service is up. Although this re-definition influences results, in the best-case scenario, their experiments show about 7 seconds of total downtime and about 300 ms of end-to-end delay, even with this re-definition. This also makes it difficult to compare with most work where downtime corresponds to handover. Yang et al. [9] developed a multi-tier MEC server deployment framework based on the pre-copy method in order to predict the next node based on the UE's position, direction, speed, and delay requirements. Their experiments demonstrate several seconds of downtime when using various prediction-based techniques with varying UE speeds. Majeed et al. [10] use four regression models to predict offloading time in MEC using various runtime and offline metrics such as CPU and disk utilization, network bandwidth, and container image size to reduce end-to-end latency compared to their other evaluated ML approaches. In the best-case scenario, they achieved 1.4 seconds of delay by proactively transferring data prior to hand-off and employing the pre-copy technique. Pomalo et al. [21] used K-Nearest Neighbor, Logistic Regression, Random Forest, and XGBoost machine learning (ML) algorithms to predict how much time the migration service should start in advance to the new edge node in order to continue service without interruptions. Their results were evaluated by comparing these ML approaches to determine which method best-handled service continuity. This study requires the path type, e.g. main road, highway, and train, to predict the proper migration start time; otherwise, it produces long downtimes. Furthermore, they stated that their approach still has downtime but did not specify its exact amount. The research mentioned above reduces downtime when compared to other ML algorithms. The downtime is reduced because handover is triggered earlier by taking into consideration factors such as the load on the wireless links and the speed of the UE. Our work is able to reduce downtime regardless of when the handover is triggered. As a result, in most cases, the two method groups mentioned above, prediction-based and data-transfer, can be combined with some modifications to achieve better performance results.

2.3. Contribution

The pre-copy migration method is used in most of the studies mentioned. Most of the included work described in this section still results in significant downtime, delay, and migration time. The issue with the previously mentioned studies is that they continue to use pre-copy as the primary transfer method [6, 9-17, 21] while reducing transfer size during hand-off [6, 12-17]. Consider an AR application in a head-mounted device. This app requires a latency of less than 17 *ms* to operate appropriately [20], whereas the above-mentioned research latencies and even solely downtimes outputs exceed 17 *ms*. As a result of such a long delay, the user's QoE may suffer, and there is a demand for a new approach to smoother migration that replaces the pre-copy migration technique.

MiGrror [1] and Hybrid-MiGrror are two solutions to the problem stated above; it is intended for applications that require low latency or real-time response. Using MiGrror and Hybrid-MiGrror rather than pre-copy and hybrid-copy can result in less downtime, delay, and migration time. The pre-copy technique is based on rounds, but memory contents may change more than once during a round, and the source must wait for a certain amount of time before sending dirty pages from the source to the destination. As a result, the source may send dirty pages at a low rate until the hand-off triggers. To improve the efficiency of future IoT and 5G applications, a new design is required that synchronizes more frequent memory differences from the source to destination during migration, reducing the transfer size during hand-off and, thus, downtime and delay. In the remainder of this paper, we will present our approach as a complement to current research, i.e. prediction-based methods and other techniques, since it can be used instead of pre-copy, resulting in less downtime and delays than pre-copy.

3. MiGrror and Hybrid-MiGrror: mirroring service on fog/edge migration

This section describes the MiGrror and Hybrid-MiGrror migration methods, which are based on VM/container mirroring (a combination of the terms migration and mirror). Mirroring is used in our technique to synchronize the source and destination VM/containers more rapidly during hand-off.

3.1. Description of Pre-Copy

This subsection presents the pre-copy method. With pre-copy, a node is selected for migration. A VM/container is initialized on the target node. This is followed by transferring the memory state to the target node even as the VM/container executes on the source node. This transfer is round 1. For round i, where $i > 1$, a page is transmitted if it has been written to (dirtied). The rounds typically end based on a predefined number. At this point, the VM/container on the source node is terminated, and devices using it are now expected to use the VM/container on the target node. It is critically important to minimize the downtime when the VM/container stops running. During this time, the VM/container stops running, and the service/data are inaccessible. Since downtime contributes to performance loss, this should be kept as low as possible, if not zero. We define downtime as follows:

$$t_{downtime} = t_{transfer} + t_{resume} \quad (1)$$

where $t_{transfer}$ is the time required to transfer the last dirty memory of the VM/container, which is calculated as follows:

$$t_{transfer} = Amount\ of\ the\ last\ dirty\ memory\ (bits)\ /\ second \quad (2)$$

This is defined as the time necessary to disconnect from the current AP (the AP of the source edge node) and connect to the next AP (AP of the destination edge node). We define t_{resume} as the time needed to resume a VM/container at the destination based on the received data after hand-off. Since $t_{transfer}$ has the largest impact on downtime and migration time when compared to other stated parameters [22], we focused on $t_{transfer}$ to reduce downtime.

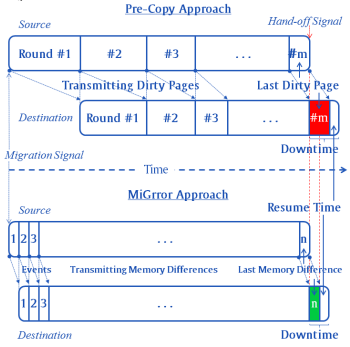

Fig. 1. The distinction between MiGrror and Pre-Copy methods.

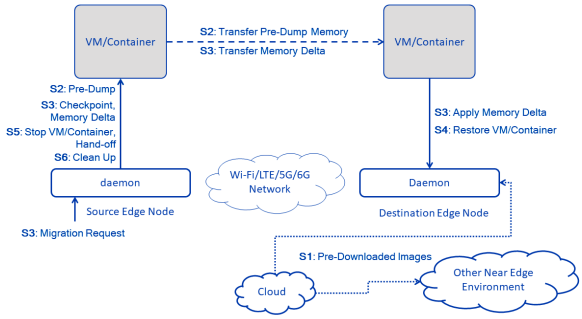

Fig. 2. Workflow of MiGrror.

3.2. Description of MiGrror

Figure 1 depicts the distinction between the MiGrror and pre-copy approaches. Assume a UE is moving from one node to another. As illustrated in Figure 1, pre-copy transmits dirty memory at the end of a round representing a predefined amount of time. With MiGrror, the goal is to reduce the amount of data that must be transferred during downtime in order to achieve higher performance; therefore, we focused on reducing $t_{transfer}$. The distinction between the two methods is that MiGrror, as shown at the bottom of Figure 1, uses events to synchronize (sync) the source and destination as events occur, rather than waiting for the end of a round as pre-copy does. Each memory change at the source causes an event to be generated, indicating that the source and destination must be synced. MiGrror does not need to wait for a period of time to elapse. Instead, MiGrror allows the possibility of multiple synchronizations of the source and destination during the period of time that corresponds to the pre-copy's round in order to mirror the current VM/container available at the destination. These n MiGrror sync events and m rounds of pre-copy are depicted in Figure 1. In most cases, n is expected to be larger than m since MiGrror syncs as soon as a memory change occurs and sends memory differences as soon as they become available. A small amount of dirty memory remains when hand-off is triggered. After the hand-off trigger, this data is the last memory difference that the source sends to the destination for synchronization. This is less memory than the last round in pre-copy since other memory differences have already been transmitted. As a result, $t_{transfer}$ is reduced in MiGrror when compared to pre-copy. Consequently, as shown in Figure 1, downtime is reduced compared to pre-copy. The diagram's right-most section represents the resumption time required to restart the VM/container at the destination. The remainder of this section will discuss MiGrror.

Figure 2 shows the design details of our entire workflow. The source edge node is the node that is currently providing services to end-user applications. After these steps are completed, the destination will provide the migrated service.

3.3. The Hybrid-MiGrror Migration Method

We intend to reduce the amount of data sent after hand-off since the destination will need to remotely read the data from the source, increasing the delay. The hybrid-copy method has a significant disadvantage due to the remote read, which increases the end-to-end latency. Therefore, if we synchronize the source and destination faster in the pre-hand-off phase of hybrid migration by replacing the pre-copy with the MiGrror, we can reduce data to transfer in the post-hand-off phase. As a result, we use MiGrror for fast synchronization of the source and target during the pre-hand-off phase. Thus, a mirror of the source's VM/container is ready at the destination throughout the migration process. While the device is still connected to the source during the pre-hand-off phase, the destination is ready for the UE to hand off and connect to the target when it enters the destination's range during the post-hand-off phase.

3.3.1. General Overview of the Migration Procedure

The *Hybrid-MiGrror* approach is divided into three phases, as shown in Figure 3: pre-hand-off, hand-off, and post-hand-off. We will first briefly explain these phases, followed by a more in-depth discussion in the rest of this section.

3.3.1.1 Pre-hand-off Phase

In the first phase, the source sends the current VM/container's memory and runtime states to the destination.

Afterwards, each memory modification causes an event to be generated in the source edge node. The source re-transmits

modified memory pages (dirty pages) to the destination. These events may occur several times prior to the hand-off being triggered. As a result, this process is repeated while the VM/container is running at the source.

3.3.1.2 Hand-off Phase

During the hand-off phase, the VM/container is stopped at the source node and copies the minimum required memory and state to the destination. The VM/container is then reloaded at the destination.

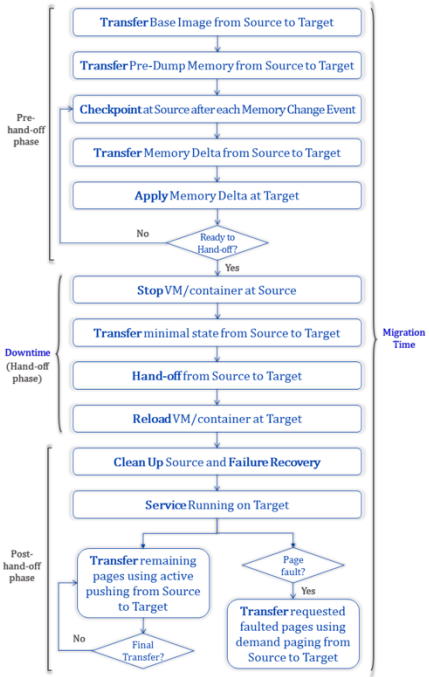

Fig. 3. Hybrid-MiGrror Major Migration Procedures.

3.3.1.3 Post-hand-off Phase

The VM/container is now running at the destination. However, some data at the source was not copied since it was dirtied during the latest copy event and hand-off. This data should be transferred to the destination node proactively by the source node. Furthermore, if the VM/container requires data that has not yet been transferred from the source, a page fault is generated in order to access the required page.

3.3.2. The Hybrid MiGrror, Step-by-Step

Hybrid MiGrror begins with the **pre-hand-off phase**, which is comprised of five steps:

S1: Transfer the Base Image

When a UE approaches the source, the source edge node or cloud sends the base image to the next prospective edge node, the target (Figure 3, first step).

S2: Transfer the Pre-Dump Memory

The source edge node synchronizes (syncs) a memory snapshot to the destination before receiving the migration request (Figure 3, step 2). To decrease the size of the transmitted memory image during hand-off, this technique first checkpoints the source VM/container and then dumps (stores memory content) a snapshot of VM/container memory in step S2 without disrupting the VM/container service at the source edge node.

S3: Migration Request and Checkpoint

After receiving the migration request from the cloud/edge node, the source edge node captures a memory snapshot to synchronize in the following steps to the target (Figure 3, step 3). The source node monitors changes in memory events and the hand-off signal event. When a memory change event occurs, the VM/container on the source edge node performs a checkpoint to capture a memory snapshot. The dirty memory is then identified by comparing the latest snapshot with the previous one for memory deltas.

S4: Transfer Memory Delta

The memory delta is then transferred from the source to the target edge node and reassembled (Figure 3, step 4). This technique transmits memory deltas from the source edge node to the destination in order to provide synchronous mirroring without interfering with the VM/container service at the source as the source sends a copy to the destination. It may sync to one or more nodes, depending on the policy.

S5: Apply Memory Delta

When a memory delta arrives at its destination, an event is generated to apply the received memory delta into the VM/container's memory (Figure 3, step 5). The VM/container is now restored at the target with the most current source changes.

Since the source and destination are **synchronized**, a **mirror** of the source's VM/container continues to run at the target as a result of steps S1 through S5 in the pre-hand-off phase. Furthermore, depending on the needs of the application, mirroring may begin prior to the migration request.

Hand-off phase: The second phase starts after the edge-cloud control mechanism initiates the hand-off. This phase consists of four steps, which are as follows:

S6: Stop VM/container

The source edge node stops the VM/container to prevent further memory and state modification.

S7: Transfer Minimal Memory and States

At this point, only the most recent minimum memory and runtime states required at the destination to reload the VM/container in the next steps should be transferred from the source to the destination. This action reduces the amount of data transferred during hand-off and, as a result, downtime to a bare minimum.

S8: Hand-off

The destination now contains all the required data. This data was obtained as a result of stages S1 through S7 execution. The control is now passed to the next edge node (target).

S9: Reload the VM/container

The target edge node reloads the VM/container with the most recent updates obtained in step S7. Steps S8 and S9 can be completed simultaneously.

Steps S6 through S9 are completed during the downtime period, as indicated in Figure 3. During this period, the UE is unable to use its services from the source or destination. Therefore, this time must be kept as short as possible.

The last phase, the **post-hand-off**, consists of three steps. The VM/container executing at the destination can start running at this point, but it may still have data at another edge node (source) that needs to be transferred since some data is not transferred during downtime.

S10: Clean-Up and Failure Recovery

The current VM/container is removed, leaving the source edge node memory clean. The aim is to wait a certain amount of time before removing the VM/container from the source. The waiting period has the advantage of serving as a backup if the connection to the next node fails or if the UE goes back.

S11: Service Running on the Target

After reloading the VM/container at step S9, the VM/container can run at the destination. The target edge node will now service the UE. Steps S10 and S11 could be completed simultaneously to save time because they will be performed in separate nodes.

Step S12, as depicted in Figure 3, is divided into two subordinate steps, S12.1 and S12.2, which complete their tasks **concurrently**.

S12.1: Transfer the Remaining Pages

Despite employing the MiGrror concept in the first phase, which causes less data to remain at the source than live migration approaches that have not yet been transmitted, the source proactively transmits the remaining migrated VM/container data to the destination. This step avoids multiple fault pages at the destination, which causes significant delays since the target must read the faulty pages remotely from the source.

S12.2: Transfer Faulted Pages

The remaining data is delivered proactively by the source; however, the destination VM/container may require data that has not yet been transferred. As a result, the target issues a page fault to the source in order to access the required part of the VM/container's memory.

Our motivation for employing this strategy is to provide up-to-date data and services to the destination node for the incoming UE and its applications as soon as possible. As a result of combining the two principles, the delay of the *Hybrid-MiGrror* approach would be kept to a minimum level as the result of these actions: (i) Using the MiGrror concept prior to hand-off and (ii) transferring the least amount of data feasible during the hand-off phase. Since less data remains at the source during the post-hand-off phase to transfer, the *Hybrid-MiGrror* causes fewer delays than live migration techniques, resulting in page faults and lengthy delays. This contributes to keeping the delay to a minimum for the real-time and near-real-time applications mentioned in the introduction. Consequently, the migration process could be faster and smoother than if only live migration methods were used.

In the final phase, the target edge node will proceed to step S1 to prepare for a possible future migration.

Fig. 4-A, 4-B, and 4-C. A smart city scenario. A: top, B: middle, C: bottom. (Image location: Nathan Phillips Square, Toronto, ON Canada, from maps.google.ca. Red car from ferrari.com, both accessed Dec. 2020).

3.4. A Smart City Scenario

Assume a road on which a mobile UE is traversing. The mobile UE could run applications such as video conferencing, VR, or an autonomous vehicle. As shown in Figure 4-A,B, and C, a UE will go out of the node *i* range and enter the node *j* range. In this figure, node *i* is the source node, and node *j* is the destination.

We can see a model of mobility, edge nodes, migration, and hand-off in Figure 4-A,B, and C. As the figure shows, the end-user moves from point A to C while keeping the connection alive for apps and their services. Figure 4-A shows the user located at point A; the migration starts here. Mirroring starts in the first phase of *Hybrid-MiGrror* when the end-user is in the range of node II. Therefore, a live copy of the source node's data resides at the destination and is ready to be used.

As the end-user device moves forward, it reaches point B presented in Figure 4-B. The hand-off is triggered when the UE is at point B. It passes the connection from the current edge node (node *i*) to the new one (node *j*). At this point, application data should have finished transferring from edge node *i* to *j*, allowing apps to continue to provide services when the end-user hands-off and reaches point C, as shown in Figure 4-C. As soon as the hand-off is completed in the third phase of the *Hybrid-MiGrror*, the destination actively fetches the remaining VM/container's data from the source.

When the user reaches point C, the previous node has already finished the VM/container migration (source node *i*). Thus, apps receive their service from the new edge node (destination node *j*). Each time a user goes from one point to another, this process happens. Since data is synchronized swiftly from the source to the destination, the applications' response time at points B and C will take minimum delay regarding migration interruptions. Our proposed technique also helps apps handle their services while the end-user crosses from one edge node range to another. This performance has a cost, which might be losing some of the network bandwidth during synchronization.

4. Results and Discussions

This section covers the simulation setup and presents insights obtained from the proposed method's simulation results.

4.1. Simulation Setup

The simulations in this section are run using MobFogSim [27], extended to deliver results. We also make use of UEs' real-time movement patterns. These patterns are embedded in MobFogSim by using the Simulation for Urban Mobility (SUMO) [28] and Luxembourg SUMO Traffic (LuST) [29]. The LuST is a dataset that analyzed vehicles in Luxembourg for 24 hours in 2015, covering an area of 156 km2 and 932 km of road. Our simulation used this dataset to determine the vehicle's location, speed, timing, and movement direction.

This study evaluated the performance of our proposed techniques using a virtual machine running on an Ubuntu 18.04.5 LTS with a 2.2 GHz quad-core CPU, 64 GBs of storage, and 32 GBs of RAM. We assumed a mobile edge computing infrastructure with 2 Mega Bits Per Second (Mbps) bandwidth for each UE and 100 Mbps bandwidth for each AP. Other features, such as a processing limit of 300 MIPS, can also be used to describe UE configuration. A million

Instructions Per Second (MIPS) is a unit of measurement for computing power.

We made different assumptions regarding the input values as part of the simulation environment. We chose a square 10 × 10 km region with 144 edge nodes evenly distributed around the area as the environment. Each edge node is linked to a single access point with a signal range of 1000 meters. When the user approaches the migration point, which is configured as 40 meters from the coverage border of the connected access point, the migration procedure begins (source node). Among the potential edge nodes present in the user's route, our migration strategy selects the candidate edge node with the shortest distance (Euclidean distance) between the user and the access point. The stated dataset determines the user speed, and the network bandwidth between edge nodes is set to 1000 Mbps with a latency of 1 *ms*. These values are summarized in Table 1 and Table 2. We ran 50 simulations with identical setups and randomly selected users from the LuST dataset in each simulation run. Therefore, we have the same bandwidth and latency between network edges, but there may be different users with different speeds and directions in each run. The average of the 50 simulation runs for each method is then used in the results. The simulation ends once the user has finished the migration steps.

Table 1. Configuration of various devices

Device Type	Downlink Bandwidth	Uplink Bandwidth
Mobile-Device (UE)	2 Mbps	2 Mbps
Access Point	100 Mbps	100 Mbps

Table 2. Input parameters and values assumed in experiments

Parameter	Value
Access point range (radius)	1000 m
Number of edge nodes	12×12 (144)
Density of edge nodes per access points	1:1
Migration strategy	Shortest Distance
Migration point	Static (40 m)
User's speed	Variable, based on the dataset
Network Bandwidth between edge nodes	1000 Mbps
Network Latency between edge nodes	1 ms

4.2. Performance Characteristics

The techniques are evaluated using five metrics which are described below.

4.2.1. Average Delay

The average latency is defined as the average delay between mobile user equipment running an application and the edge node running the corresponding VM/container. The definition of delay does not include packet loss in this paper as they never arrived at the destination, and we cannot measure their delay. A lower value means that users must wait less time to acquire data from the migrated service hosted on the destination edge node, and as a result, the performance of their running applications improves.

4.2.2. Downtime

Downtime is the time it takes to relocate a paused virtual machine or container from one edge node to another and restart it. The VM/container stops functioning during this time, and the services and data associated with it are no longer accessible to the user.

4.2.3. Average Migration Time

The average migration time is the average time it takes to prepare, complete multiple transfers, and restart the VM/container at the destination edge node. Migration is completed when the VM/container resumes at the target edge node for the final time with all the required data.

4.2.4. Network Usage

Network usage refers to the amount of data exchanged between the source and destination nodes for the duration of the migration, from when migration initiates until the migration is completed.

4.2.5. Packet Loss

Packet loss during downtime refers to the number of packets that user equipment needs to send to or receive from an edge node but is unable to do so due to a lack of connectivity between the UE and the edge node. This measure represents a pause in service delivery.

4.3. Simulation Results

The pre-copy, MiGrror, Hybrid-Copy and Hybrid-MiGrror simulation results are compared in terms of metrics described in the previous subsection. Notably, in our experiments with hybrid migrations, we employed a post-copy via active pushing since it produces less delay and migration time than other types of post-copy [30].

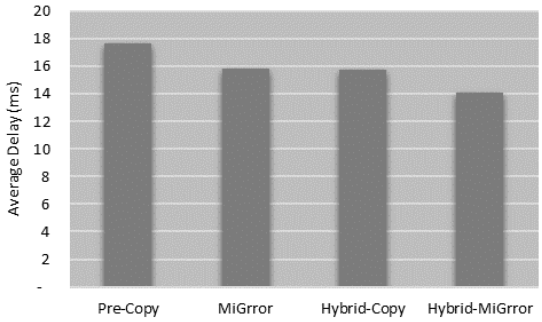

Fig. 5. Average Delay.

Figure 5 depicts the average delay. The average delay does not account for lost packets and only evaluates packets that reach their destination. As shown in the figure, MiGrror has lower average latency when compared to the pre-copy method. Since we use MiGrror in the first phase rather than pre-copy, we expected a lower average delay in our proposed hybrid method. Figure 5 illustrates that Hybrid-MiGrror has more than a 10% lower average delay than hybrid-copy. As a result, delay-sensitive apps can run more smoothly, resulting in a better user experience. Based on these findings, it is reasonable to conclude that applications using Hybrid-MiGrror will have less waiting time before accessing the migrated service. This performance contributes to the smooth running of latency-sensitive applications while maintaining high performance in the edge environment. This latency should be acceptable for most applications, especially for 5G applications requiring a minimal delay, generally less than 15ms end-to-end latency [31].

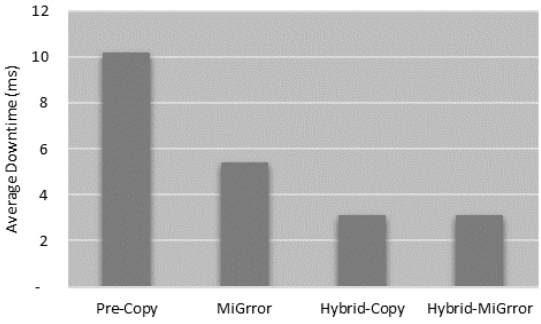

Fig. 6. Average Downtime.

Another factor that we take into consideration is downtime. The downtime associated with the pre-copy migration is the longest, lasting more than 10ms, as shown in Figure 6. The downtime for the MiGrror migration is 50% less than that for the pre-copy migration. Both hybrid migration techniques have roughly the same amount of downtime, which is less than the downtime for pre-copy and MiGrror procedures, around 3ms. Hybrid versions have less downtime since they do not need to wait for all the remaining memory and state to be transferred from the source to the destination in order to resume the VM/container at the target, as explained in section 3.3.2.

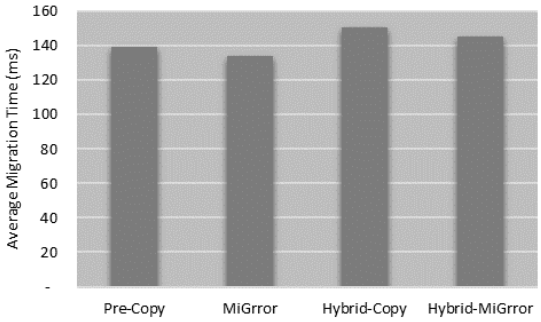

Fig. 7. Average Migration Time.

Figure 7 represents the average migration time for each technique. MiGrror and pre-copy migration techniques require less migration time than hybrid methods since all data is delivered before and during hand-off. However, both hybrid approaches require slightly more migration time than the other two methods, albeit only by about 5% since both methods use the post-copy strategy in their third phase. Furthermore, Hybrid-MiGrror takes less time to migrate than hybrid-copy since it uses MiGrror in the first phase and has less data to transfer at the source when the UE hands off to the destination. As a result, less data is required to be sent after hand-off than with the hybrid-copy method.

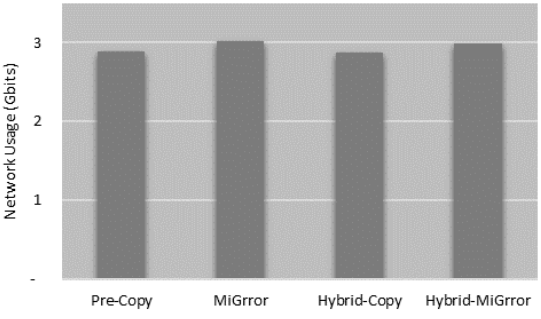

Fig. 8. Network Usage.

The amount of data sent for migration is shown in Figure 8. Since all techniques transfer memory pages multiple times before handing them off, their network bandwidth consumption is considerable. MiGrror uses more bandwidth than pre-copy as it syncs the source and destination more frequently than pre-copy. The network consumption of the Hybrid-MiGrror is greater than that of the hybrid-copy, as shown in the figure. This is due to the deployment of MiGrror in the first step of the proposed method.

Furthermore, the amount of data transferred by pre-copy and hybrid-copy methods is roughly the same. Similarly, as illustrated in Figure 8, MiGrror and Hybrid-MiGrror have almost the same data transmission. They sent the most data because they sent more memory deltas than the other two migration methods; however, they used only 5% more bandwidth. This increased bandwidth usage is because MiGrror and Hybrid-MiGrror sync more frequently during migration than pre-copy and hybrid-copy. Sync events data transfer is low. Even though there are a high number of sync events, the small volume produced by a sync event results in the amount of data transferred during migration rising by 5%. Moreover, Hybrid-MiGrror synchronizes using local network bandwidth rather than the remote traffic that applications currently use to communicate with cloud servers.

The data transmission volume may be an issue when using the proposed methodology since a less bandwidth-intensive technique is preferable. 5G networks, on the other hand, have more than ten times the bandwidth of 4G networks, ranging from 1 to 10 Gbps [32]. Edge nodes using our migration methodology may sync with each other more frequently than previous approaches, consuming more bandwidth in order to provide reasonable downtime for future highly delay-sensitive applications requiring a low-latency response. It is still dependent on the application's requirements and use-cases. An app may require ultra-low latency or be bandwidth constrained. Different criteria influence the use of the stated migration techniques, which should be considered when selecting a migration approach.

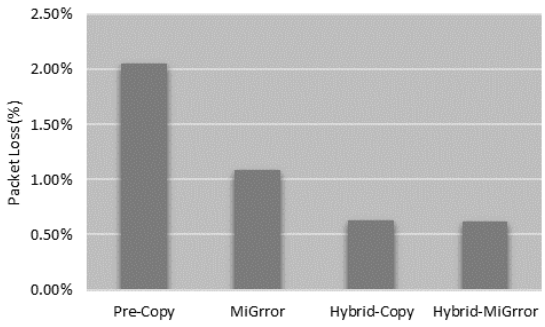

Fig. 9. Packet Loss.

Figure 9 shows that the pre-copy method has the highest packet loss during downtime since it causes more downtime than other methods; it affects packet loss. At the same time, the MiGrror has a packet loss of slightly more than 1%. Both hybrid techniques have the lowest packet loss during downtime, around 0.6 percent. This result was expected since they had the shortest downtime of the evaluated techniques, indicating another advantage of hybrid migration methods.

4.4. Discussions

Although MiGrror significantly reduces the delay compared to pre-copy, Hybrid-MiGrror will reduce the delay even further. The major distinction between the four migration methods is that the Hybrid-MiGrror reduces delay with negligible impact on other parameters. Therefore, Hybrid-MiGrror can offer services with low delay and downtime for real-time applications despite a slight increase in migration time and network usage. The reason that the Hybrid-MiGrror

network usage is slightly higher than the hybrid-copy is that despite more frequent syncs, the data transfer amount in each sync event is insignificant. This is because doing more frequent syncs reduces the volume of data transferred in each sync event.

5. Conclusion

This study investigated the significance of VM/container migration in mobile edge computing. Migration may consider the edge node's geographical location with respect to the user, the user's direction and speed, as well as the network characteristics of both the user and the edge node. Four distinct migration mechanisms were depicted and simulated. We account for the user's movement and wireless connection in the simulation. Our research discovered that Hybrid-MiGrror performs well when supporting MEC applications with mobile users, especially when the real-time response is critical. Even though MiGrror can serve the majority of latency-critical applications, when it comes to applications requiring a real-time response, Hybrid-MiGrror performs better by reducing delay and downtime. Delay-insensitive applications may use the pre-copy or hybrid-copy method when the delay is not critical since, despite reducing delay and downtime, MiGrror and Hybrid-MiGrror use higher network bandwidth.

Future research should define and enhance various migration mechanisms in the fog/edge computing environment for various scenarios/use-cases. It would be worthwhile to consider trade-offs in various scenarios. In order to provide a higher quality of service to applications, different methods may be required in different situations; for example, Hybrid-MiGrror has a lower delay and a longer migration time than MiGrror.

A Hybrid-MiGrror extension can be used in stateless migration. Because this container does not need to store states, it can replicate by cloning or mirroring from the source node or cloud to the prospective destination node/nodes without requiring migration. However, some data may still be linked to containers, as containers require this data in order to retain the user's data.

Furthermore, combining Hybrid-MiGrror with existing machine learning (ML) and non-ML approaches would also contribute to a promising method for minimizing delay and downtime. Synchronizing from a source to multiple nodes should be beneficial when the source is uncertain of the following potential node to migrate to or when the application wants cooperation with more than one node to improve performance.

Acknowledgments

This work is partially funded by the Natural Sciences and Engineering Research Council of Canada.

References

1. Rezazadeh, A., D. Abednezhad, and H. Lutfiyya, *MiGrror: Mitigating Downtime in Mobile Edge Computing, An Extension to Live Migration.* Procedia Computer Science, 2022. **203**: p. 41-50.
2. Habibi, P., et al., *Fog Computing: A Comprehensive Architectural Survey.* IEEE access, 2020. **8**: p. 69105-69133.
3. Debauche, O., et al., *Towards Landslides Early Warning System With Fog - Edge Computing And Artificial Intelligence.* Journal of Ubiquitous Systems and Pervasive Networks, 2021. **15**: p. 11-17.
4. Abdelaziza, J., M. Adda, and H. McHeick, *An Architectural Model for Fog Computing.* Journal of Ubiquitous Systems and Pervasive Networks, 2018. **10**: p. 21-25.
5. Bonomi, F., et al., *Fog computing and its role in the internet of things*, in *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*. 2012, Association for Computing Machinery: Helsinki, Finland. p. 13–16.
6. Ma, L., et al., *Efficient Live Migration of Edge Services Leveraging Container Layered Storage.* IEEE Transactions on Mobile Computing, 2019. **18**(9): p. 2020-2033.
7. Chiang, M. and T. Zhang, *Fog and IoT: An Overview of Research Opportunities.* IEEE Internet of Things Journal, 2016. **3**(6): p. 854-864.
8. Martinez, I., A.S. Hafid, and A. Jarray, *Design, Resource Management and Evaluation of Fog Computing Systems: A Survey.* IEEE Internet of Things Journal, 2020: p. 2494-2516.
9. Yang, R., H. He, and W. Zhang, *Multitier Service Migration Framework Based on Mobility Prediction in Mobile Edge Computing.* Wireless Communications and Mobile Computing, 2021. **2021**: p. 1-13.
10. Majeed, A.A., et al. *Modelling Fog Offloading Performance*. in *2020 IEEE 4th International Conference on Fog and Edge Computing (ICFEC)*. 2020.
11. Ngo, M.V., et al. *Coordinated Container Migration and Base Station Handover in Mobile Edge Computing*. in *GLOBECOM 2020 - 2020 IEEE Global Communications Conference*. 2020.
12. Addad, R.A., et al., *Fast Service Migration in 5G Trends and Scenarios.* IEEE Network, 2020. **34**(2): p. 92-98.
13. Zhou, Z., et al. *Hardware-assisted Service Live Migration in Resource-limited Edge Computing Systems*. in *2020 57th ACM/IEEE Design Automation Conference (DAC)*. 2020.
14. Machen, A., et al., *Live Service Migration in Mobile Edge Clouds.* IEEE Wireless Communications, 2018. **25**(1): p. 140-147.
15. Ma, L., 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, Association for Computing Machinery: San Jose, California. p. Article 11.
16. Farris, I., et al., *Providing ultra-short latency to user-centric 5G applications at the mobile network edge.* Transactions on Emerging Telecommunications Technologies, 2018. **29**(4): p. 1-14.
17. Puliafito, C., et al., *Design and evaluation of a fog platform supporting device mobility through container migration.* Pervasive and Mobile Computing, 2021. **74**: p. 101415.
18. Souza Junior, P., D. Miorandi, and G. Pierre. *Stateful Container Migration in Geo-Distributed Environments*. in *CloudCom 2020 - 12th IEEE International Conference on Cloud Computing Technology and Science*. 2020. Bangkok, Thailand: IEEE.
19. Bonanni, M., F. Chiti, and R. Fantacci, *Mobile Mist Computing for the Internet of Vehicles.* Internet Technology Letters, 2020. **3**(6): p. 1-6.
20. Salman, S.M., et al. *Fog Computing for Augmented Reality: Trends, Challenges and Opportunities*. in *2020 IEEE International Conference on Fog Computing (ICFC)*. 2020.
21. Pomalo, M., et al. *Service Migration in Multi-domain Cellular Networks based on Machine Learning Approaches*. in *2020 7th International Conference on Internet of Things: Systems, Management and Security (IOTSMS)*. 2020.
22. Puliafito, C., et al., *Container Migration in the Fog: A Performance Evaluation.* Sensors, 2019. **19**(7): p. 1-22.

23. Martins, R.d.J., et al., *Virtual Network Functions Migration Cost: from Identification to Prediction.* Computer Networks, 2020. **181**: p. 1-16.
24. Conforti, L., et al. *Extending the QUIC Protocol to Support Live Container Migration at the Edge*. in *2021 IEEE 22nd International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM)*. 2021.
25. Deshpande, U., et al. *Agile Live Migration of Virtual Machines*. in *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. 2016.
26. Zhang, F., et al., *A Survey on Virtual Machine Migration: Challenges, Techniques, and Open Issues.* IEEE Communications Surveys & Tutorials, 2018. **20**(2): p. 1206-1243.
27. Puliafito, C., et al., *MobFogSim: Simulation of mobility and migration for fog computing.* Simulation Modelling Practice and Theory, 2020. **101**: p. 1-25.
28. Behrisch, M., et al., *SUMO – Simulation of Urban MObility: An Overview*, in *SIMUL 2011*, S. Aida Omerovic, et al., Editors. 2011, ThinkMind: Barcelona.
29. Codeca, L., R. Frank, and T. Engel. *Luxembourg SUMO Traffic (LuST) Scenario: 24 hours of mobility for vehicular networking research*. in *2015 IEEE Vehicular Networking Conference (VNC)*. 2015.
30. Hu, L., et al., *HMDC: Live Virtual Machine Migration Based on Hybrid Memory Copy and Delta Compression.* Applied Mathematics & Information Sciences, 2013. **7**(2L): p. 639-646.
31. Panwar, S., *Breaking the millisecond barrier: Robots and self-driving cars will need completely reengineered networks.* IEEE Spectrum, 2020. **57**(11): p. 44-49.
32. Agiwal, M., A. Roy, and N. Saxena, *Next Generation 5G Wireless Networks: A Comprehensive Survey.* IEEE Communications Surveys & Tutorials, 2016. **18**(3): p. 1617-1655.