Container placement and migration strategies for Cloud, Fog and Edge data centers: A survey

Kiranpreet Kaur, Fabrice Guillemin, Francoise Sailhan

Container placement and migration strategies for Cloud, Fog and Edge data centers: A survey

Kiranpreet Kaur[1,2], Fabrice Guillemin[1], and Francoise Sailhan[2]

[1]Orange Innovation, 2 Avenue Pierre Marzin, Lannion, France, email: {firstName.lastName}@orange.com
[2]Cedric Laboratory, CNAM Paris, 292 rue St Martin, Paris, France, email: {firstName.lastName}@cnam.fr

Abstract

The last decade has witnessed important development of network softwarization that has revolutionized the practice of networks. Virtualized networks bring novel and specific requirements for the control and orchestration of containerized network functions that are scattered across the network. In this regard, the migration of virtualized network functions plays a pivotal role to best meet the requirements of optimal resource utilization, load balancing and fault tolerance. The purpose of this survey is to offer a detailed overview of the progress on container migration so as to provide a better understanding of the trade-off between the benefits associated with the migration and the practical challenges. The paper includes a classification of the placement algorithms that map the containerized network functions on the virtualized infrastructure. Following, a taxonomy of the migration techniques that perform the transfer of the containerized microservices is proposed.

Keywords: Container migration; migration techniques; placement strategies.

1 Introduction

A notable trend in current networks is the network softwarization that promotes the adoption of virtualization technologies to support the rapid development of new services that readily adapt to the evolving customer needs. Network softwarization leads to the gradual replacement of hardware network function operating on purpose-built & proprietary network equipment by Virtualized Network Functions (VNFs) that are consolidated on commodity hardware. In practice, a network function (NF) CNFs may offer a wide range of networking capabilities that operate on the Universal Customer Premise Equipement (uCPE), up to the core network supporting e.g. tunneling, firewalling or application-level functions. Microservices have become instrumental in the design of complex NFVs that necessitate a decomposition into many of services - e.g. several hundreds services for core network functions. In such case, microservices are small services implementing a limited amount of functionalities that can be executed independently (even if they are logically dispersed at the edge, fog or in the cloud) - each microservice executes its own processes/functionalities and communicates via lightweight protocols.

Overall, cloud-native design offers a different approach to the development of softwarized networks, an approach that is suited to the agility and that supports an efficient scaling up and orchestration of the distributed network functions. Container-oriented approach is also increasingly privileged as a containerized microservice can be rapidly instantiated as required and also can be scaled-out independently, to support the increasing demand for more processing or storage, without unnecessarily scaling the overall network function.

While expectations are high for the wide applicability of network softwarization, putting the deployment of VNFs into practice is far from being a trivial task especially. With NFV, a network service is composed of series of network functions (a.k.a microservices) characterised by a predefined order, which is known as *service chaining*. By design, VNF supports a dedicated specific functionality and often remains state-dependant, i.e., in practice, states are stored and updated locally with the associated VNFs. Following the chain, traffic goes through the series of ordered network functions such that traffic may flow back and forth among distant VNFs as VNFs reside on distinct physical servers or data centers. During the operation of the VNF, traffic, network bandwidth, available storage and computational resources typically fluctuate over time, which results in imbalanced links/ resource usage. Thus, the efficient allocation and the continuous management of NFVs become more complex, considering the heterogeneity and the dynamics of the physical resources as well as the ephemeral nature of the services. To overcome this issue, a growing number of research effort has been devoted to support the migration of network services possibly to other physical server(s)/data center(s), which is key to preserve the Quality of Service (QoS) and meet the expectation of the user in terms of performance. While related topic including VM/container migration in cloud data centers has matured, the decoupling and elastic re-allocation of small networking functions across data centers spanning the edge to the core remains challenging.

1.1 Related surveys

With the virtualization gaining attention, many surveys on VM (e.g. [1–7]) and container (e.g., [8–10]) attempt summarizing the different Virtual Machine (VM) and container migration techniques and their challenges, from a general perspective, without special emphasis on their usage to support network softwarization. In addition, some surveys [11–13] provide a comprehensive state of the art on NFV solutions, introducing the basic concept and principles associated with the network slicing and virtualization considering in majority VM-enabled virtualization.

Among general purpose surveys, most surveys specifically deal with VMs for edge or fog [14–16] computing. Only few recent surveys [8–10] specifically concern the container-enabled NFV migration and each focuses either on edge [8], fog [9] or cloud migration [10], With mobile edge computing [8], the user mobility is the key aspect that is considered. In [9], a detailed analysis outlines the trade-off that is made between the benefit associated with migrating a service (e.g., QoS improvement) and the related cost, which also brings out architectural design and implementation. In [10], authors classify various techniques for cloud and container-based migration on the basis of the following taxonomy : (i) Architecture, (ii) Tools, (iii) Purpose, (iv) Scope, (v) Migration technique, and (vi) Evaluation.

Overall, the aforementioned surveys depict the migration strategies to adopt within a specific area of the network infrastructure (cloud versus edge, versus fog) with regards to some specific use cases (such as Connected & Autonomous Vehicles, Cloud video, Online gaming and Augmented reality). While it is essential to accurately characterize the migration strategy by taking into account the network infrastructure as a whole, that is, to identify the design rationale that needs to be made in order to migrate a containerized service on top of a shared infrastructure.

1.2 Contribution

The survey presented in this paper focuses on container migration spanning the Cloud, Fog and Edge computing levels. The main contributions are as follows: We provide for an extensive survey and classification of the placement strategies, whose goal is to identify the appropriate target server(s) to allocate the migrated service. A classification of the existing container migration techniques is proposed and reflects the way the service components hosted in a container are moved from one (or several) physical server(s) to another one(s). Subsequently, we perform a holistic review of the strategies for container migration over a geographically spanned networks (edge, fog, core and cloud levels) and we describe the frameworks and algorithms that have been used to migrate container-based services.

The organization of this paper is as follows: In Section 2, we present the motivation for container migration and the associated benefits. Placement strategies are reviewed in Section 3. A classification of container techniques is proposed in Section 4. In the subsequent Section 5, we review the container migration strategies for networks geographically spanned. Concluding remarks and research perspective are presented in Section 7.

2 Motivation for migration

The rise of microservice architecture amplifies the usage of containers that offer an ideal host for the small and self-contained microservices. Nowadays, network operators, cloud providers (e.g., AWS, Google) and content providers (e.g., Netflix, BBC) are adopting the microservice architectural style [17,18] and deal with applications that may comprise even hundred or thousands of containers. Even though containers come up with the benefit of packing all the dependencies of a Network Function (NF) into a single unit, managing, deploying and migrating these containers in a large and multi-cloud infrastructure using self-made tools or scripts becomes increasingly complex and difficult to manage. In this regards, various container orchestration frameworks such as, Docker Swarm, Kubernetes and Apache Mesos provide additional support for deploying and managing a multi-tiered application as a set of containers on a cluster of nodes. In this regards, Kubernetes is characterised[1] by a strong industry adoption - interested reader may refer to [19,20] for comparative studies.

With Kubernetes, the simplest and most straight-forward strategy to adopt for migrating stateless containers running in a pod (i.e. in a group of containers utilizing shared resources) is the following (Figure 1): a pod is recreated at the destination node; then, the controller/scheduler shifts all the requests from the old pod to the new one when this latter is in full-active stage; finally, the source pod is deleted. The container migration is facilitated because containers come

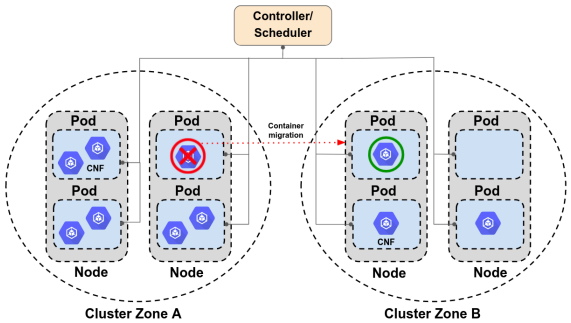

Figure 1: General layout of migration in Kubernetes: container is migrated from cluster A to cluster B

up with the benefit of packing all the dependencies of a NF into a single unit which is conveniently moved. Nonetheless, the orchestration of the container migration is a key challenge with large and multi-cloud infrastructure:

1. **Inter-node balancing:** Depending on the service dynamics, some nodes/clouds may become overloaded, provided that nodes/clouds capacity is heterogeneous. As illustration, edge and fog data centers are small comparing to cloud data centers. In order to balance the load between nodes and data centers, it is necessary to support migration.

2. **Node failure and system maintenance:** When a node encounters unexpected/planned failures/shutdown, the service continuity needs to be guaranteed through service replication or migration.

[1] as pointed in the openstack annual user surveys: https://www.openstack.org/analytics

3. **Attain the optimality of placement:** Services join and leave the system of data centers while 5G users are expected to move. Containers share one set of resources (such as, CPU, RAM, disk on a physical server) that may vary over time. Depending on the placement of microservices and users, it may be relevant to move some containerized (micro)services e.g., close to the new location of the end user or far away to free resource for the incoming services.

As illustrated in Figure 2, orchestration technology is needed to continuously monitor the virtualized network (e.g., resource usage, failures and the arrival or departure of services); if necessary find the best place for the containerized micro-services (Section 3); perform the migration of the containerized (micro)services accordingly (Section 4).

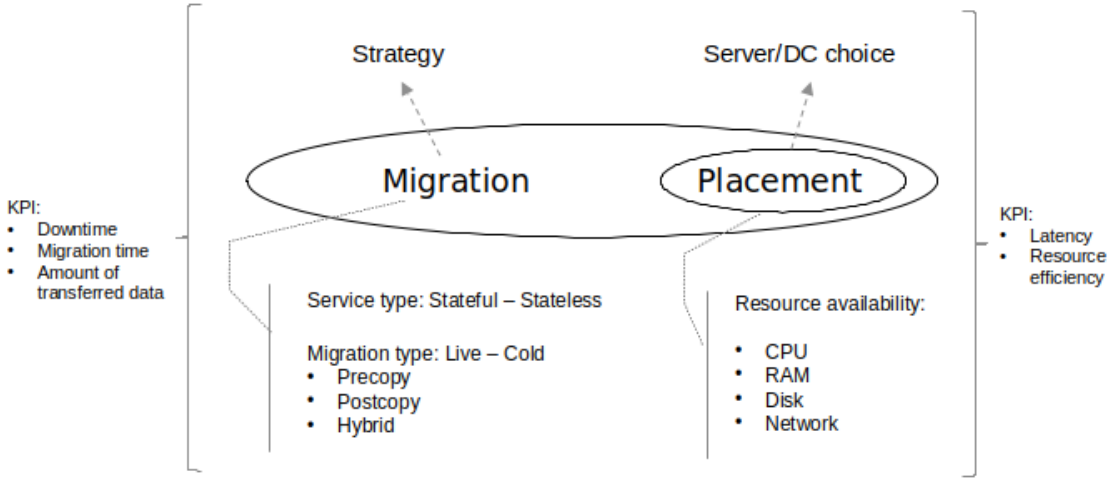

Figure 2: Representation of Migration strategy with Placement

For these reasons, the study and encapsulation of numerous container-based approaches for placement and most importantly for migration that could fulfill these principle requirements is a major demand by business units offering services to customers.

3 Container placement strategies

Migration of a set of Cloud-Native Network Function (CNF)s is known to effectively bring more elasticity and scalability to (mission/latency-critical) applications. On the other hand, migration may entail service disruption and may come at the cost of intensive use of computing and communication resources, even though there is a strong practical need for migration. The service migration entail taking a decision concerning the service placement, which consists in determining whether, when and where to migrate. The service placement problem is an optimization problem that involves a balanced trade-off between the cost associated with the migration and the expected benefits.

The service placement problem (Table 1) is usually framed as a mathematical optimization (Integer Linear Programming and Mixed Integer Linear Programming), which is further solved by an optimization solver, heuristic methods or Machine Learning (ML) approaches. ILP and MILP solvers typically find nearly optimal solutions but are quite time-consuming and hence are not practically viable for large and complex problem instances.

Instead, heuristics (e.g., greedy algorithms) and meta-heuristics produce comparatively faster but sub-optimal results that usually achieve less objectives (e.g., low response time or reduced communication delay or load balancing or limited energy consumption). On the other hand, ML-based approaches (e.g., genetic algorithm, ant colony) are known to be more accurate solutions [33] thanks to their interactive learning and decision making abilities.

Table 1: Classification of placement methods

Methods	**Reference**
Interger Linear Programming (ILP)	[21–26]
Mixed Interger Linear Programming (MILP)	[27–32]
Heuristic Method	[22, 23, 25, 26, 28, 30–32]
Machine Learning (ML)	[33–37]

As detailed in Table 2, the above container placement strategies can be further categorized based on target architecture (cloud, fog, edge), type of placement (static versus dynamic), key objectives, algorithm to solve and evaluation method. In the case of static placement, an initial placement is typically proposed only once (at start). Instead, dynamic placement involves multiple reallocation decisions that are made over time. A new placement is proposed e.g., in case of overuse/under-use of computing or network resources, inflow/outflow of service instances [38, 39]. Contrary to the static placement which is based on initial constraints (e.g., expected delay/latency, initial bandwidth usage and initial resource availability), the dynamic placement involves a continuous monitoring of the physical resources and network to support the selection of the appropriate hosting server(s) and/or data center(s) despite changing resources/requirements. Static and dynamic placement strategies attempts to enhance the service quality and/or reduce the operational cost by means of various strategies, particularly: *1) Resource-aware placement:* to avoid unwanted overuse/under-use of resources and decrease operational cost by balancing the load among distributed data-centers and hosts; *2) Latency-aware placement:* to facilitate the fast inter-communication considering processing and migration delay, transmission and queuing delay; *3) Security-aware placement:* to avoid the allocation on container owned by an adversary user, identifying the unexpected threat or failure and cross-container attacks. Once the placement decision has been made, migration must be carried out.

4 Classification of container migration

Container migration refers to the process of transferring or moving the components of a network function hosted within a container from one physical server (source node) to another one (destination node), possibly interrupting the network function operation. Network functions that are migrated are either *stateless* (i.e., no past data nor state needs to be persistent or stored) or *stateful* (i.e., the application state lasts and is stored, e.g., on disk). With stateless network function, migration is quite straightforward [52] because the container operates in an isolated manner and is hence portable: the stateless container is simply re-allocated and restarted from scratch without conserving the existing state.

As depicted in Figure 3, there exists several techniques for moving the container from the source to the destination. They subdivide into cold and live migration depending on whether the containerized service should remain active and network-accessible during the whole migration.

4.1 Cold and Live Migration

There exists two ways of migrating a container: during the migration the containerized application is inactive (cold migration) or remains active (live migration).

4.1.1 Cold migration

This is the trivial form of migration in which the container is simply suspended and migrated between hosts. As illustrated in Figure 4, cold migration involves the freeze-transfer-resume steps: First, the container is freezed to ensure its associated state is not modifiable. Second, the dump

Table 2: Comparison of existing work related to placement of containers/instances

Ref.	Archi.	Placement Type	Objectives	Algorithm	Evaluation
[40]	Fog	Static	Response time, Inter-container network communication	Greedy & Genetic Algorithm	Comparison with 3 approaches
[41]	Fog	Static	Response time of task	Ant colony optimization	Simulation
[42]	Edge	Dynamic	Scheduling	Reviewed heuristic-based algorithms	Case study
[43]	Cloud	Dynamic	Rebalancing, Load balancing	Scheduling & Rebalancing process	Simulation (with real-time load)
[44]	Cloud	Dynamic	Resource utilization, Number of instances	Best Fit (BF), Max Fit (MF) & Ant Colony Optimization based on Best Fit (ACO-BF)	Simulation with real-time workload (compare three algorithms)
[45]	Three-tier (Container-VM-PM)	Dynamic	Resource utilization	Best-fit	Use-case
[46]	Cloud	Dynamic	Traffic flow, Placement cost, Resources	One-shot, Rounding and heuristic algorithm	Theoretical Analysis and trace-driven simulations
[47]	Cloud	Dynamic	Communication cost, Load balancing	Communication Aware Worst Fit Decreasing (CA-WFD), Sweep&Search	Extensive evaluation on Baidu's data centers (Comparison with exiting SOA strategies)
[48]	Edge	Static	Container images' retrieval time	KCBP (k-Center-Based Placement), KCBP-WC (KCBP-Without-Conflict)	Trace-driven simulations (Compared with Best-Fit and Random)
[49]	Edge-Fog-Cloud)	Dynamic	Service delay, Resource management	Particle-swarm-optimization (PSO)-based meta-heuristic, Greedy heuristic	Use-case benchmarking (comparison of 4 approaches)
[50]	Containers as a service (CaaS)	Static	Energy consumption	Improved genetic algorithm	Compared with other 6 algorithms
[51]	Edge-Fog-Cloud	Dynamic	Automate database container placement decision	Markov Decision Processes (MDP)	Testbed
[21]	Edge-Fog-Cloud	Static	End-to-end service Latency	Greedy & Genetic Algorithm	Evaluation of proposed strategy solved using 2 algorithms

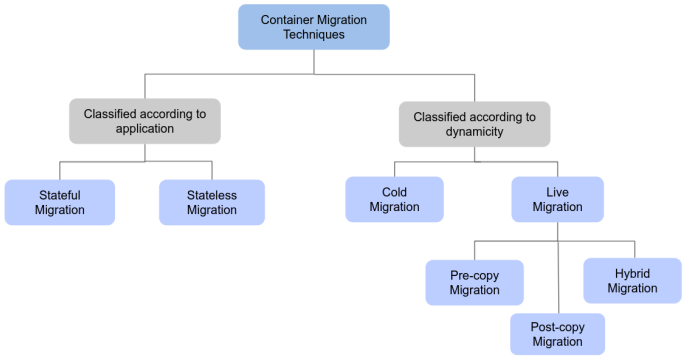

Figure 3: Container migration Techniques

state is transferred while the container is stopped. After the reception of the state at the destination node, the container is finally re-started and its state is resumed. Overall, cold migration involves a service downtime and thereby should be used in specific cases only, for instance when users are not using the service for a given time period or when the downtime is planned and users are informed.

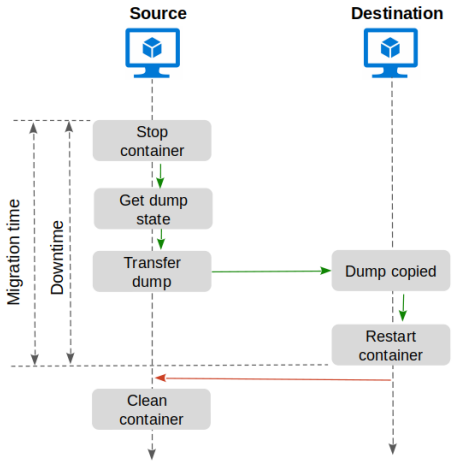

Figure 4: Cold migration

4.1.2 Live migration

consists of migrating a running container without service interruption, i.e., container migrates from one node to another while it is running. The main portion of the state is transferred while the container is running; the container is stopped only during the transmission of the execution state. Therefore, service downtime is quite negligible for the end-user.

Both cold and live migration entail the transfer of the original container. In practice, migrating an inactive service (cold migration) involves shutting down the running instance and thereby eliminating the need to handle the memory state. Instead, moving an active service (live migration) necessitates maintaining state consistency during the migration. In particular, in-memory state (including both kernel-internal and application-level state) should be moved in a consistent and efficient fashion. With live migration, the main concern lies in maintaining state consistency (as will be shown) while keeping to a minimum downtime (i.e. time between the container stops and resume) and total migration time (duration between when migration is initiated and when the container may be finally discarded at the source.

4.2 Handling State Consistency with Live Migration

Live migration can be approached in several ways: memory state can be sent ahead of time before the container is transferred (pre-copy) or later, i.e., after the container is transferred (post-copy) or combining the pre-copy and post-copy migration techniques (hybrid).

4.2.1 Pre-copy Live Migration

As shown in Figure 5, the container at source continues to run while pre-dump states are transmitted from source node to destination node. Therefore the service stays responsive during the transmission phase. At that time of copying and transferring of the pre-dump state, memory is kept modifiable at the source node. Then, the container is stopped and restarted at the destination node. The dump state and the memory content (memory pages) that have been modified are transferred. The service downtime (i.e., time between when the container is halted and resumed) is minimized because the container is stopped after the transmission of its state while the memory is also changeable.

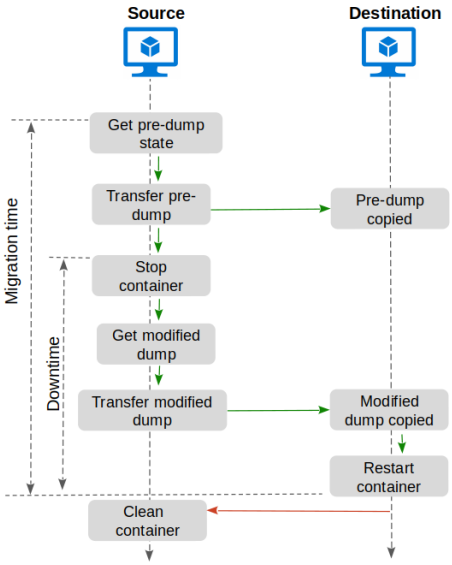

Figure 5: Pre-copy live migration

4.2.2 Post-copy Live Migration

As Figure 6 depicts, the process is initiated by first halting the container at source node, the (minimal subset of) execution state is transmitted to the destination node and the container is resumed as soon as possible based on its latest execution state. Later on, the remaining state (including memory pages) is transferred to the destination node before deleting the container at source node. At the destination, if the restarted container attempts to access a memory page that is not yet available, fault page is demanded to the source node, hence causing an additional delay.

4.2.3 Hybrid Live Migration

As shown in Figure 7, hybrid approach advents by combining the pre-copy and post-copy migration techniques. Following the pre-copy approach, the pre-dump state is transmitted while the container is still alive at the source node. After halting the container, the full dump state (modified and execution state together) is transmitted. Then, the container is restarted using the full dump state. Final step proceeds to transfer the memory contents (faulted pages) that were caused

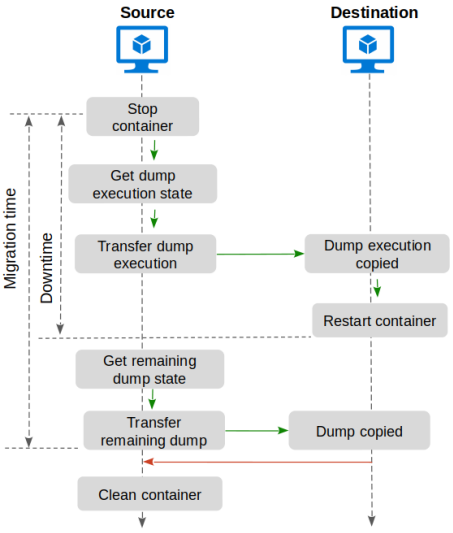

Figure 6: Post-copy live migration

during the pre-copy phase. Hybrid migration addresses the issues related to non-deterministic downtime with pre-copy migration and performance degradation by dint of faulted pages in the post-copy migration approach.

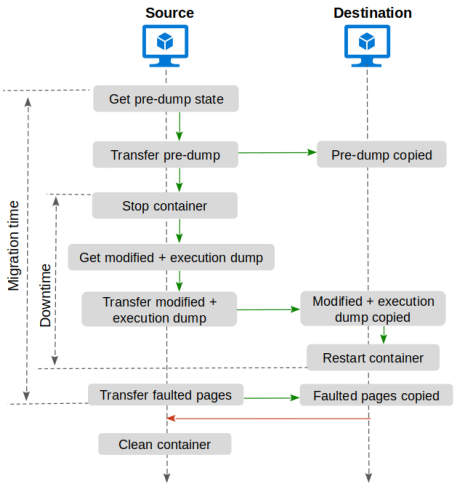

Figure 7: Hybrid live migration

In practice, pre-copy, post-copy or hybrid migration is performed using a snapshot/restore tool such as CRIU[2], which has become a de facto standard to handle migration of linux container with OpenVZ, LXC, and Docker. CRIU is an open source tool that dumps the state of processes/containers into a collection of image files on disk and makes it possible to further resume an app (i.e., to restore an app) from exactly where it was suspended. Nonetheless, CRIU has some limitations. CRIU focuses on the internal state of the containerized application, which includes the states of the CPU, registers, signals and memory that are associated with the container. CRIU does not transfer any file/state across physical nodes. To this aim, complementary techniques shall be used to dispose of the files/information necessary for recovery at the destination node. In practice, files are transferred using the rsync primitive or a shared and possibly distributed file-system

[2]https://www.criu.org/

such as NFS, GlusterFS, or Virtuozzo[3] that are used to store files and avoid transferring them.

4.3 Storage Migration

Typically, the state of a network function is local (i.e., accessed by the container by a virtual local disk) if the state is frequently accessed. For example, per-flow state (such as state for individual TCP connections) is local, as long as the traffic is distributed on a flow basis. In the container, the internal state is stored with the network function instance and thereby achieves good performance (e.g. fast read, write). Early work, e.g., [53], on NFV management assume that the state is internal; this assumption permits easy migration and elastic scaling of network functions. In practice, the state is then migrated as part of the container image.

Nevertheless, the transfer of the whole container file system results in a high network overload. In order to optimize and reduce the size of the container file system that is transferred from the source to the destination node, a number of works [54, 55] take advantage of the layered structure of Docker. Docker storage is formed of several layers: base image layers are read-only while upper layer is read-write. Read-write layer encapsulates all the file system updates issued by the container since its creation, which encompasses (i) the files created by the containerized application as well as (ii) the files corresponding to the updated versions of the read-only layers. Thus, read-only layers can be fetched before the migration from a Docker repository (e.g., public cloud repository such as Docker Hub[4] or self-hosted image hubs) while the thin top writable layer is transferred from the source to the destination node. Following, [55] goes one step further and also checkpoint the current state of the read-write container layer, which further reduces the container's migration downtime.

Another line of research breaks the tight coupling between the NF state from the processing that network functions need to perform by externalizing the storage leveraging a resilient data store that is either central [56–58] or distributed [59] and that can be accessed by any NF. Nonetheless, any access (read, write, delete) to the externalized datastore involves a significant communication overhead. To reduce the communication overhead, in-memory data store is privileged in [59, 60]: the state is stored in DRAM leveraging RAMCloud [61] which corresponds to a key-value in-memory datastore with low latency access or Redis [5]).

Another approach introduced a variant of CRIU named VAS-CRIU that avoids costly file system operations that dominate the runtime costs and impact the potential benefits of manipulating in-memory process state. Contrary to CRIU that suffers from expensive filesystem write/read operations on image files containing memory pages, VAS-CRIU saves the checkpointed state in memory (as a separate snapshot address space in DRAM) rather than disk. This accelerates the snapshot/restore of address spaces by two orders of magnitude, and restore time by up to 9 times.

4.4 Applicability and Performance Evaluation

Few empirical studies evaluate the performance of container migration such as [52, 62, 63]. They compare the performance of various container migration techniques (e.g. cold, live migration) to that of VM migration and consider multiple virtualization platforms. First, the referred work [52] analyzes the performance of cold and live - pre-copy, post-copy and hybrid - migration to identify the best techniques while transmitting stateful containers from one node to another. The comparison between cold and live migration indicates that, as expected, cold migration has the lowest total migration time and highest downtime in comparison to various live migration techniques because cold migration transmits the whole state at once after the suspension of the container at source node.

The delay associated to post-copy migration is high migration compared to that of cold migration as it passes on the faulted pages served on request from source node after resuming the container at destination node. Likewise, pre-copy migration depicts better results than post-copy

[3] https://wiki.openvz.org/Virtuozzo_Storage
[4] https://hub.docker.com
[5] https://redis.io/

migration when the network has sufficient throughput to convey changed pages quickly, which is the case if network throughput is greater than or close to the page change rate. Otherwise, pre-copy migration is less efficient compared to post-copy. On the other hand, the hybrid migration always involves higher migration time as it results from the combination of pre-copy and post-copy techniques.

Significantly, live migration keeps the container active during the migration process to reduce downtime and maintain responsiveness of the containerized service throughout the communication exchange. The evaluation of the downtime shows that the downtime is lower for the post-copy technique compared to the pre-copy technique and remains comparable to the hybrid technique. The evaluations concerning the amount of transferred data is also showing better results for post-copy, wherein the quantity of transferred data is always lower than for pre-copy and hybrid, but remains competitive to cold migration.

In [62], authors provide a detailed comparison of the performances associated with a VM-enabled and container-enabled live migration supporting the functions of core network functions, including the Home Subscriber Server (HSS), Mobility Management Entity (MME), and Serving and Packet Gateway (SPGW).

First, the analysis of the migration time associated with the HSS VM is comparatively twice that of the HSS container. It takes a modest amount of additional time to complete the VM and container migration process while using a longer path. On the other hand, containers incur a higher downtime than VMs because the containerized HSS is stopped on the source host when checkpointing is initiated and is resumed only once after the complete restoration at the destination host.

Second, the MME VM has a migration time six/seven times higher than the MME container, as the network load and metadata size of the container is comparatively smaller than the VM. Therefore, the large image size and longer path clearly have an impact on the migration time of the VM. Conversely, the analysis of the container downtime shows double that of the VM because the migration process has to be stopped at the checkpoint stage and restarted only after restoration.

Finally, the SPGW VM also implies much higher migration time than the container due to the large size of the metadata for the VM. However, an interesting result can be observed: the downtime improves for the SPGW VM compared to the container migration, which was not the case with HSS and MME. During the SPGW migration, the UE recovery time is affected by the new UE connection that has to be successfully re-established by updating the sockets after the temporary failure occurred.

The work [63] analyse the real-time behaviour of containers in the cloud environment, under two distinct workloads (100% and 66%) to . With regard to total migration time, downtime and disk utilization, Linux Containers (LXC) exhibits better outcomes compared to Kernel-based Virtual Machine (KVM) except for the CPU utilization which is better with KVM. In particular, the downtime of KVM is increased by 1.6 and resp. 1.75 times with the workload of 66% and resp. 100% in comparison to LXC. Similarly, the migration time of KVM is 1.35 and 1.45 times higher compared to LXC at the workload of 66% and 100% respectively. Similarly, the live migration with KVM and LXC which has an impact on on their disk utilization. The highest disk utilization of KVM is 455,555 writes/sec and 482,672 writes/sec at the workload of 66% and 100% respectively. Whereas, LXC has a maximum disk utilisation of 301,192 writes/sec and 330,528 writes/sec for a workload of 66% and 100% respectively. Moreover, the evaluations related to CPU utilization shows that LXC has a maximum CPU usage of 78.12% and 86.24% for a workload of 66% and 100% consecutively. However, KVM on the other hand performs better outcomes by lowering upto 73.09% and 74.07% for 66% and 100% workload respectively.

5 Strategies for container migration techniques

As shown in Figure 8, container migration schemes can be classified into three computing layers, which form the underlying virtualization infrastructure.

The topmost cloud layer constitutes the largest centralized storage and computing resource

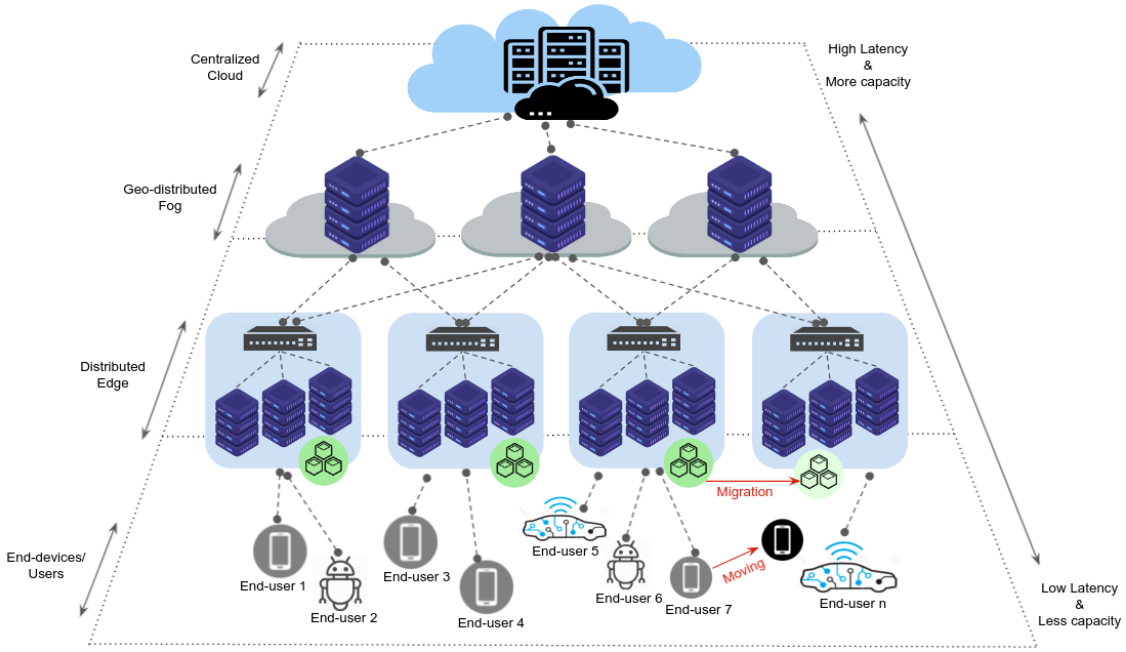

Figure 8: Three-layered Cloud-Fog-Edge Infrastructure

along with high scalability that is persuasively acquired by end-users in an on-demand manner. The utilization of container-based infrastructures for large-sized environments evidently constitutes a popular choice by dint of its key characteristics - lightweight, scalability, and high portability. Moreover, the cloud-native principle enables network services to be implemented as a bundle of microservices interconnected to each other and deployed on distributed and container-based infrastructures (e.g., Kubernetes) [64] in the cloud. Nonetheless, there exists an inherent limitation associated with cloud computing: the long communication distance results in excessively long delay and the security factors in public cloud models risk the users privacy and unauthorized access to databases [65].

Fog computing provides a promising solution by decreasing the distance between end user's devices and cloud data centers. Cloud functions can be moved towards the end user device in the event of low-latency interactivity. In practice, containerized microservices migrate from centralized cloud to geo-distributed fog nodes [52,66], which share the workload and lessen the network traffic. Therefore, strategies under fog perform the migration among geo-distributed and heterogeneous data centers. In such case, careful migration of data volumes plays a significant role especially for live and stateful containers. Nonetheless, microservice requesting more computing/storage resources can be offloaded from fog nodes to cloud data centers.

Further, the edge nodes located near the end users provide comparatively lower latency at a cost of limited resource capacity in comparison to cloud and fog servers. Edge clouds enable the deployment of servers near to the user to fulfill the demand of latency-critical applications.In particular, migration techniques map/migrate the containers from one location to another depending on the user moves. That, later on optimize the quality-of-experience (QoE) and network-related requirements by dynamically mapping the containerized services on container-based virtualized environment [67].

While a cloud-fog-edge architecture has the potential to unlock tangible opportunities for industry, it remains pivotal to rely on a mature container migration strategy. In the following, we consider the migration techniques that can be followed to support the migration at any layer of the virtualization infrastructure. Table 3 compares the proposed approaches based on their migration type, architecture, scope and considered factors to be handled during migration and the

detailed explanation is also provided in the proceeding section. Compared to VM Migration that has attracted considerable interest, there are not so much works that address container migration within the cloud (§ 5.1), the fog (§ 5.2) or the edge (§ 5.3).

5.1 Container Migration on Cloud

CloudHopper [69] supports live migration of multiple interdependent containerized applications across multiple clouds over a wide network. The automated solution (relying on Ansible [81]) offers multi-cloud support for three commercial clouds providers (namely, Amazon Web Services, Google Cloud Platform, and Microsoft Azure). The migration of multiple interdependent containers necessitates a network migration to (i) easily locate the other containers and (ii) hold the incoming traffic during the effective migration and eventually redirect when the service gets restored and ready. For this purpose, an IPsec VPN is set up between the source and target and a TCP/HTTP load balancer (HAProxy [82]) is used and tuned to redirect the http traffic and to return unavailability message (HTTP 503 Service Unavailable Response) if timeout occurs during the migration. To support memory pre-copy, the CRIU's iterative migration capability is leveraged. Rather than supporting a parallel transfer of the multiple containers, migration is scheduled: containers are ordered by size and large-size containers are migrated first. The next container starts its migration when the previous container has a remaining transfer size that is equal to its transfer size. This scheduling approach uses more efficiently the network bandwidth and enables to start all containers almost immediately upon arrival at the target.

Further, the work [68] adopts the pre-copy algorithm for docker migration accross data centers of a cloud network. Different from VM, Docker has a layered image and Docker containers share the same OS kernel, which make live migration of a Docker container more complex as image, runtime state and context should be migrated. The migration starts by transferring the base layers of the docker image that are read-only by disconnecting the storage volume at the source and re-attaching it at the target node. Then, CRIU performs incremental memory checkpoint and supports the iterative migration of the upper layer which is read-write and thereby possibly updated during the whole migration process. The experimental results show 57% lessened total migration time, 55% lower image migration time, and 70% of downtime on average in comparison to mentioned state-of-the-art.

The work presented in [77] proposed a solution for live container migration named Voyager, which follows the design principle specified by Open Container Initiative (OCI) [83] which is a consortium initiated by industry leaders (e.g. Docker, CoreOS) and encourages the common and open specifications of container technology. Voyager provides stateful container migration by using the CRIU-based memory migration and union mounts so as to retrieve source container data on the target node without copying container data in advance in order to minimize migration downtime. Thus, voyager support the so-called just-in-time zero-copy migration where container can restart before transmission of whole states at destination node. This allows Voyager containers to instantly restart at destination host during disk state transmission by means of on-demand copy-on-write and lazy replication.

The live migration model ESCAPE [71] focuses on defense mechanisms for cloud containers by modeling the interaction between the attackers and respective victim hosts as a prey game. The container acts as a prey whose aim is to evade attacks/predator. For the checkpointing of a running containerized application while migration, the model employs an experimental version of Docker that includes the CRIU checkpoint tool. ESCAPE detects and circumvents attacks by either preventing any migration during an attack or migrating the container(s) far away from the potential attacker(s).

In [72], authors propose the frequent relocation of docker containers to reduce the impact of data leakage. Inspired by Moving target defense (MTD) technology, the approach promotes the container migration to shorten the container's life cycle and thereby guarantee the security of large-sized multi-tenant service deployment. Similarly, the defense framework introduced in [73] offers fast and high frequency migration of VMs/containers so as to obscure the migration process for the attackers. In particular, the destination hosts are chosen randomly which may degrade the

performance by means of load and latency.

MigrOS [79] enables the transparent live migration of RDMA-enabled containers which require specialised circuitry of the network interface controllers (NICs) and thereby are not transparently supported so far. The OS-level migration strategy requires a modification to the RDMA protocol but still supports full backwards-compatibility and interoperability with the existing RDMA protocol, without compromising RDMA network performance. In order to evaluate the solution, the modified RDMA communication protocol has been integrated with SoftRoCE, a Linux kernel-level open-source implementation of the RoCEv2 protocol. In addition, the solution has implemented in NIC hardware.

In [65], the first migration framework of Intel Software Guard Extensions (SGX)-enabled containers is presented. SGX provides a trusted execution environment named enclave for containers. An *enclave* [84] corresponds to a secure separate encrypted area used by a process to store code or data. The key challenge behind migrating SGX-enabled containers relates to the SGX security model that prevents the states of the enclaves, which is encrypted, to be accessed during the migration process. In order to support the migration of the enclave, the solution encrypts the persistent data stored of the enclave using a symmetric key that is shared by the source and destination node. An empirical evaluation show that the migration of SGX-enabled containers introduce about 15% overhead. In [70], author secure the live migration of container for both stateful and stateless applications. Application server acts as a control manager that orchestrates the migration process. Also, a secure migration path is established using SSH/SFTP that both support authentication, communication confidentiality and integrity.

5.2 Container Migration on Fog

The container migration strategy [55] within Kubernetes for stateful services in geo-distributed fog computing environments attempts to minimize the downtime. In case of stateful migration, it is required to migrate the disk state along with the container, which is a time consuming process in large-sized and distributed migration. To address this issue, the layered structure provided by the OverlayFS file system [85] is used to transparently snapshot the pod volumes and transfer the snapshot content prior to the actual container migration. At the source server, the snapshot content becomes read-only and a new empty read/write layer is added on top. Overall, the approach supports the check-pointing of the current state of the container layer. If needed, several snapshot transfers may be performed, which led to minimizing the container's migration downtime: experiments on a real fog computing test-bed show up to factor 4 downtime reduction during migration in comparison to a baseline with no volume checkpoint.

In [74], the migration framework supports both horizontal migration where containerized IoT functions are migrated from one gateway to another gateway and vertical migration in which IoT function containers are migrated from the gateway located at the edge to the Cloud. The strategy is quite straightforward: the stateless container is re-created at the target node and then deleted from the source node.

The formerly known Heptio Ark project, currently stands out as Velero [80] to leverage the migration of Kubernetes applications and their persistent volumes. Compared to existing tools, it utilizes the Kubernetes API instead of Kubernetes etcd to extract and restore the states. Which can be advantageous when users do not have access to etcd databases. The resources exposed by API servers are simple to backup and restore even for several etcd databases. Further, additional functionality of backing up and restoring of any type of Kubernetes volume is provided by activating the restic [86]. The release of Velero is available on GitHub [87].

5.3 Container Migration on Edge

The work presented in [75] designs a third party tool to perform a live migration of services on edge infrastructure. The goal is to reduce the migration time by minimizing the transferred file size by leveraging the layer structure of the docker container storage system. As docker image is composed of layers usually emerged from Dockerfile represents a set of instructions in the image. During

the container's whole life cycle, only the top storage layer is changeable. The layers underlying the top layers remain unchanged. Therefore, the proposed strategy transmits only the top layer during the migration process, rest underlying layers have been transmitted before commencing the process.

Moreover, authors consider the migration of the service to the end server located near the actively moving end-user: when a user shifts at a new location, then the offloading computation service also passes on to the edge server which is closer to the end user's new location. In order to attain the fast migration and lessen network traffic, the proposed framework already starts preparing the target edge node before the commencement of the migration process and parallelize & pipeline the following steps:

1. Parallelize the downloading of the images from a centralized registry at the target nearest edge node and pre-dump/send base memory images from source to target node while starting the container.

2. Reload the docker daemon on the target host (after halting the container at source node). The reload can also be parallelized with the dirty memory transmission from source to target host or could be trigger just after the transmission of latest container layer. Note that container layer can be compressed before to transmit. Also, container layer compression and transmission can be pipelined. Similarly, the process of acquiring the memory difference at the target server could be pipelined.

The work [67] supports live migration based on Linux Container Hypervisor (LXD) and CRIU and introduces a novel heuristic scheme. The proposed heuristic follows these steps: First, a source node shortlists the containers that are characterised by high latency. For each high-latency container, the source node finds the neighbor node that is geographically closed and that is characterised by good resources availability (e.g. load, CPU, RAM, bandwidth) to migrate the container.

In order to perform the live migration of containers for latency critical industrial applications, the work [88] demonstrates the redundancy migration approach for edge computing. The approach skipped the stop-and-copy phase of traditional live migration that followed the snapshot and checkpointing, transmission and restoration of state image at the target node. Therefore, the key four composed phases are - 1) Buffer and routing initialization phase, 2) Copy and restore phase, 3) Replay phase and 4) switch phase. This significantly minimizes the downtime by a factor of 1.8 in comparison to LXD (Linux containers Daemon) stock live migration as per the evaluation.

In [76], authors present the migration framework that follows the three-layered architecture - Base layer, Application layer and Instance layer to relocate containers or VMs across MECs. Aiming to enhance the performance by placing the service near to the user, the paper considers the stateful migration of applications and induces to minimize the overall migration time and service downtime. The following procedure is: First, the primary system configuration (guest OS, kernel, etc.) except application included base layer is transmitted on each MEC in order to avoid the transmission on each migration request. Second, the idle application and its data-included application layer is passed on when migration is triggered while keeping the service running. Then finally, the running states included instance layer is transmitted after suspending the service. Therefore, only the transmission time of the instance layer accumulated as service downtime. However, the detailed experimental explanation is not provided in the paper due to lack of space.

Another, an open-source multi-cloud and edge orchestration platform - Cloudify [89] affirms to support the pod migration without interrupting the containerized service from one node to another within the Kubernetes cluster.

The service-oriented architecture based KubeVirt [90] project introduced by Red Hat enables the additional functionality to Kubernetes. It allows live migration of VM instances (acted as a pod) from one host to another host. Therefore, it could be profitable to relocate the containerized applications (running inside the VM) from one one node to another within a cluster. It's release is hosted on GitHub [91].

Another prototypical implementation is also available on GitHub [92, 93] to include the additional commands of <kubectl migrate> and <kubectl checkpoint> with the help of modified

kubelet and customized container/cri. In this way, running pods can be checkpointed and migrated within a single or multi-clusters. Despite the fact that the work is considered to be a rough prototype, it is quite appreciable that contributed to the pod migration feature of stateful containers in Kubernetes.

6 Key Research Perspectives

Even though extensive research work has been actively proposed and improved during last decades on VMs live migration techniques. The same interest shifts towards applying these techniques on containers due to their unavoidable advantages. The aforementioned studies tried to solve some of the issues faced by container migration (OS-based virtualization) that are not concerned with VM migration (hardware-aware virtualization). Different approaches were also proposed to handle stateful & stateless container migration while reducing the migration time, downtime and size of transferred data. Still, there remain the unresolved challenges to address specific to dis-aggregated data centers, deploying and managing the chain of containerized microservices while migration along with avoiding the service disruption, drop in the QOS & disturbance of the ongoing exchange.

Current approaches consist in offloading the entire - or a portion of Docker image, preparing in advance the target node or using compression technique to tackle various migration-related KPIs. Stateful container migration requires transmitted memory states to resume the container from its suspension point at target nodes and storage data also required to transmit in case servers are located at different geographical locations as remote disk access could lead to increase the latency and transmission delay.

Existing orchestrators such as Kubernetes (K8s) [94] are mostly cloud-oriented, it contains the features of horizontal scaling - means creating a set of instances of microservice based on workload or other criteria; failure recovery and service continuity. However, it is lacking the deployment of a chain of microservice across a multi-cluster network which is initiated by Edge Multi Cloud Orchestrator (EMCO) [95, 96]. Currently, it is a key demand of an orchestrator that can handle multiple clusters along with different types of clouds (such as edge/fog/core) to deploy applications for 5G and MEC services. Notably, as per the study done by Gartner [97], 75% of generated data is expected to be processed outside centralized cloud by 2025. Thus, the execution of applications on edge/fog/cloud data centers carefully placed within the network infrastructure, implies completely different settings.

EMCO is a central orchestrator that facilitates the management and deployment of geo-distributed services across multiple distributed K8s clusters. It automates the life cycle (such as instantiation and termination) of composed service which is quite complex to handle for a large set of services supposed to deploy on distinct data centers across a multi-cluster network. It also supports multiple placement constraints based on affinity, anti-affinity or cost.

Nevertheless, there are still some challenges that require focus on. The design tool must be able to stick the microservices together. Also, it is hard to manage the connection alive during termination and re-establishment as a chain of microservices not only communicating with end-users but also with respective microservice that could possibly be placed on the server in different clusters.

Migration of containers is not only concerned with memory and storage migration but also needs to tackle the selection of an appropriate target host as complexity gets enlarged for multiple migrations. Containerization underlying on shared OS & some libraries and at the target-end there could be other containers running. Therefore, preparing the set of required libraries and docker image of the migrated container at the target host is a significant issue to disclose. Along with migrating the workload near to end users to meet various requirements (e.g., latency and service continuity) for the dis-aggregated fog and edge data centers that distribute and scale the workload.

Further, variant size of containerized NFs also required to examine service type while mapping - where first, the service composed of a chain of microservices must objectify the characteristics

that could affect the model based on time-sensitivity, latency or load efficiency. Second, related to multiple migrations to transmit multiple container's states and memory simultaneously.

In this regard, a decentralized and multi-cloud orchestration of the migration across multiple technological domains constitutes a missing building block. The design of the model must be able to distinguish the services in order to place the particular set of microservices on distributed edge centers and others on centralized clouds. Aiming to save the resources at edge as these are the critical one. Moreover, approximating towards online strategy - where services are continuously arriving or departing the system and raising the issue of resource imbalance with uncertainty of arrival/departure time, the research must consider the problem of when to trigger the migration and selection of container to be migrated in a way to attain the lower migration rate. As migration rate directly influences the system's energy consumption. Also, it is difficult to predict an optimal placement for service in exchange with moving users. At the edge layer, the movement of a user from one geographical area to another needs to be handled by the controller that in turn takes a decision that may result in triggering a migration. Therefore, specific strategies should allow for an efficient decision which is more complex with the rise of inter-dependencies in large sets of microservices deployed on different data centers.

7 Conclusion

Majority of companies and open-source communities are adapting the cloud-native approaches as of their performance efficiency which further lays together on technologies - container, orchestration and microservices that are capable of providing the highly scalable, light-weighted, portable and flexible solutions. Through this work, we aimed to evaluate study analysis on the techniques focused on container migration starting from centralized followed to geo-distributed infrastructure.

The proposed taxonomy states the importance of re-allocating the containerized services in larger-cloud data centers in case of more resource requirements or placing them on latency-efficient fog/edge data centers in the event of latency-critical highly communicable application. Therefore, the development of a real-time migration model considering the telco infrastructure as a whole induces some challenges to address concerning the application downtime and migration time.

References

[1] A. Choudhary, M. C. Govil, G. Singh, L. K. Awasthi, E. S. Pilli, and D. Kapil, "A critical survey of live virtual machine migration techniques," *Journal of Cloud Computing*, vol. 6, no. 1, pp. 1–41, 2017.

[2] S. Venkatesha, S. Sadhu, and S. Kintali, "Survey of virtual machine migration techniques," *Memory*, 2009.

[3] F. Zhang, G. Liu, X. Fu, and R. Yahyapour, "A survey on virtual machine migration: Challenges, techniques, and open issues," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 2, pp. 1206–1243, 2018.

[4] P. G. J. Leelipushpam and J. Sharmila, "Live vm migration techniques in cloud environment—a survey," in *2013 IEEE Conference on Information & Communication Technologies*. IEEE, 2013, pp. 408–413.

[5] P. D. Patel, M. Karamta, M. Bhavsar, and M. Potdar, "Live virtual machine migration techniques in cloud computing: A survey," *International Journal of Computer Applications*, vol. 86, no. 16, 2014.

[6] D. Kapil, E. S. Pilli, and R. C. Joshi, "Live virtual machine migration techniques: Survey and research challenges," in *2013 3rd IEEE international advance computing conference (IACC)*. IEEE, 2013, pp. 963–969.

[7] A. Strunk, "Costs of virtual machine live migration: A survey," in *2012 IEEE Eighth World Congress on Services*. IEEE, 2012, pp. 323–329.

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

[9] Z. Rejiba, X. Masip-Bruin, and E. Marín-Tordera, "A survey on mobility-induced service migration in the fog, edge, and related computing paradigms," *ACM Computing Surveys (CSUR)*, vol. 52, no. 5, pp. 1–33, 2019.

[10] G. Singh and P. Singh, "A taxonomy and survey on container migration techniques in cloud computing," in *Sustainable Development Through Engineering Innovations: Select Proceedings of SDEI 2020*. Springer Singapore, 2021, pp. 419–429.

[11] J. G. Herrera and J. F. Botero, "Resource allocation in nfv:a comprehensive survey," *IEEE Transactions on network and Service Management*, vol. 13, no. 3, 2016.

[12] X. Fei, F. Liu, Q. Zhang, and al., "Paving the way for nfv acceleration: A taxonomy, survey and future directions," *ACM Computing Survey*, 2019.

[13] A. A. Barakabitzea, A. Ahmadb, R. Mijumbic, and A. Hine, "5g network slicing using sdn and nfv: A survey of taxonomy, architectures and future challenges," *Computer Networks*, vol. 167, 2020.

[14] Z. Rejiba, X. Masip-Bruin, and E. Marin-Tordera, "A survey on mobility-induced service migration in the fog, edge and related computing paradigms," *ACM Comput. Surv.*, vol. 1, 2019.

[15] T. Taleb, K. Samdanis, B. Mada, and al., "On multi-access edge computing: A survey of the emerging 5g network edge cloud architecture and orchestration," *IEEE Communications Surveys and Tutorials*, vol. 19, 2017.

[16] Y. Mao, C. You, J. Zhang, and al., "A survey on mobile edge computing: The communication perspective," *IEEE Communications Surveys and Tutorials*, vol. 19, 2017.

[17] A. Balalaie, A. Heydarnoori, and P. Jamshidi, "Microservices architecture enables devops: Migration to a cloud-native architecture," *Ieee Software*, vol. 33, no. 3, pp. 42–52, 2016.

[18] A. R. Sampaio, H. Kadiyala, B. Hu, J. Steinbacher, T. Erwin, N. Rosa, I. Beschastnikh, and J. Rubin, "Supporting microservice evolution," in *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2017, pp. 539–543.

[19] I. M. Al Jawarneh, P. Bellavista, F. Bosi, L. Foschini, G. Martuscelli, R. Montanari, and A. Palopoli, "Container orchestration engines: A thorough functional and performance comparison," in *ICC 2019-2019 IEEE International Conference on Communications (ICC)*. IEEE, 2019, pp. 1–6.

[20] E. Truyen, D. V. Landuyt, D. Preuveneers, B. Lagaisse, and W. Joosen, "A comprehensive feature comparison study of open-source container orchestration frameworks," Arxiv Research report, 2020.

[21] K. Kaur, F. Guillemin, V. Q. Rodriguez, and F. Sailhan, "Latency and network aware placement for cloud-native 5g/6g services," in *Consumer Communications & Networking Conference (CCNC)*, 2022.

[22] Q. Sun, P. Lu, W. Lu, and Z. Zhu, "Forecast-assisted nfv service chain deployment based on affiliation-aware vnf placement," in *2016 IEEE Global Communications Conference (GLOBECOM)*. IEEE, 2016, pp. 1–6.

[23] D. Bhamare, M. Samaka, A. Erbad, R. Jain, L. Gupta, and H. A. Chan, "Optimal virtual network function placement in multi-cloud service function chaining architecture," *Computer Communications*, vol. 102, pp. 1–16, 2017.

[24] A. Hmaity, M. Savi, F. Musumeci, M. Tornatore, and A. Pattavina, "Virtual network function placement for resilient service chain provisioning," in *2016 8th International Workshop on Resilient Networks Design and Modeling (RNDM)*. IEEE, 2016, pp. 245–252.

[25] M. C. Luizelli, L. R. Bays, L. S. Buriol, M. P. Barcellos, and L. P. Gaspary, "Piecing together the nfv provisioning puzzle: Efficient placement and chaining of virtual network functions," in *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*. IEEE, 2015, pp. 98–106.

[26] D. Li, P. Hong, K. Xue *et al.*, "Virtual network function placement considering resource optimization and sfc requests in cloud datacenter," *IEEE Transactions on Parallel and Distributed Systems*, vol. 29, no. 7, pp. 1664–1677, 2018.

[27] H. Hawilo, M. Jammal, and A. Shami, "Orchestrating network function virtualization platform: Migration or re-instantiation?" in *2017 IEEE 6th International Conference on Cloud Networking (CloudNet)*. IEEE, 2017, pp. 1–6.

[28] J. Li, W. Shi, H. Wu, S. Zhang, and X. Shen, "Cost-aware dynamic sfc mapping and scheduling in sdn/nfv-enabled space-air-ground integrated networks for internet of vehicles," *IEEE Internet of Things Journal*, 2021.

[29] H. Tang, D. Zhou, and D. Chen, "Dynamic network function instance scaling based on traffic forecasting and vnf placement in operator data centers," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 3, pp. 530–543, 2018.

[30] M. A. Khoshkholghi, M. G. Khan, K. A. Noghani, and al., "Service function chain placement for joint cost and latency optimization," *Mobile Networks and Applications*, pp. 1–15, 2020.

[31] A. Leivadeas, G. Kesidis, M. Ibnkahla, and al., "Vnf placement optimization at the edge and cloud," *Future Internet*, vol. 11, no. 3, 2019.

[32] H. Hawilo, M. Jammal, and A. Shami, "Network function virtualization-aware orchestrator for service function chaining placement in the cloud," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 3, pp. 643–655, 2019.

[33] J. J. Alves Esteves, A. Boubendir, F. Guillemin, and P. Sens, "A heuristically assisted deep reinforcement learning approach for network slice placement," *arXiv preprint arXiv:2105.06741*, 2021.

[34] ——, "DRL-based slice placement under realistic network load conditions," in *2021 17th International Conference on Network and Service Management (CNSM)*. IEEE, 2021, pp. 524–526.

[35] J. Pei, P. Hong, M. Pan, J. Liu, and J. Zhou, "Optimal vnf placement via deep reinforcement learning in sdn/nfv-enabled networks," *IEEE Journal on Selected Areas in Communications*, vol. 38, no. 2, pp. 263–278, 2019.

[36] P. T. A. Quang, Y. Hadjadj-Aoul, and A. Outtagarts, "A deep reinforcement learning approach for vnf forwarding graph embedding," *IEEE Transactions on Network and Service Management*, vol. 16, no. 4, pp. 1318–1331, 2019.

[37] D. M. Manias, M. Jammal, H. Hawilo, A. Shami, P. Heidari, A. Larabi, and R. Brunner, "Machine learning for performance-aware virtual network function placement," in *2019 IEEE Global Communications Conference (GLOBECOM)*. IEEE, 2019, pp. 1–6.

[38] B. Zhang, J. Hwang, and T. Wood, "Toward online virtual network function placement in software defined networks," in *2016 IEEE/ACM 24th International Symposium on Quality of Service (IWQoS)*. IEEE, 2016, pp. 1–6.

[39] A. Laghrissi and T. Taleb, "A survey on the placement of virtual resources and virtual network functions," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 2, pp. 1409–1434, 2018.

[40] E. H. Bourhim, H. Elbiaze, and M. Dieye, "Inter-container communication aware container placement in fog computing," in *2019 15th International Conference on Network and Service Management (CNSM)*. IEEE, 2019, pp. 1–6.

[41] M. Gill and D. Singh, "Aco based container placement for caas in fog computing," *Procedia Computer Science*, vol. 167, pp. 760–768, 2020.

[42] O. Oleghe, "Container placement and migration in edge computing: concept and scheduling models," *IEEE Access*, vol. 9, pp. 68 028–68 043, 2021.

[43] U. Pongsakorn, Y. Watashiba, K. Ichikawa, S. Date, H. Iida, and al., "Container rebalancing: Towards proactive linux containers placement optimization in a data center," in *IEEE 41st Annual Computer Software and Applications Conference (COMPSAC)*, vol. 1, 2017, pp. 788–795.

[44] M. K. Hussein, M. H. Mousa, and M. A. Alqarni, "A placement architecture for a container as a service (caas) in a cloud environment," *Journal of Cloud Computing*, vol. 8, no. 1, pp. 1–15, 2019.

[45] R. Zhang, A.-m. Zhong, B. Dong, F. Tian, and R. Li, "Container-vm-pm architecture: A novel architecture for docker container placement," in *International Conference on Cloud Computing*. Springer, 2018, pp. 128–140.

[46] R. Zhou, Z. Li, and C. Wu, "An efficient online placement scheme for cloud container clusters," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 5, pp. 1046–1058, 2019.

[47] L. Lv, Y. Zhang, Y. Li, K. Xu, D. Wang, W. Wang, M. Li, X. Cao, and Q. Liang, "Communication-aware container placement and reassignment in large-scale internet data centers," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 3, pp. 540–555, 2019.

[48] J. Darrous, T. Lambert, and S. Ibrahim, "On the importance of container image placement for service provisioning in the edge," in *2019 28th International Conference on Computer Communication and Networks (ICCCN)*. IEEE, 2019, pp. 1–9.

[49] A. Mseddi, W. Jaafar, H. Elbiaze, and W. Ajib, "Joint container placement and task provisioning in dynamic fog computing," *IEEE Internet of Things Journal*, vol. 6, no. 6, pp. 10 028–10 040, 2019.

[50] R. Zhang, Y. Chen, B. Dong, F. Tian, and Q. Zheng, "A genetic algorithm-based energy-efficient container placement strategy in caas," *IEEE Access*, vol. 7, pp. 121 360–121 373, 2019.

[51] P. Kochovski, R. Sakellariou, M. Bajec, P. Drobintsev, and V. Stankovski, "An architecture and stochastic method for database container placement in the edge-fog-cloud continuum," in *2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 2019, pp. 396–405.

[52] C. Puliafito, C. Vallati, E. Mingozzi, G. Merlino, F. Longo, and A. Puliafito, "Container migration in the fog: A performance evaluation," *Sensors*, vol. 19, no. 7, p. 1488, 2019.

[53] S. Palkar, C. LAN, S. Han, and al., "E2: A framework for nfv applications," 2015.

[54] L. Ma, S. Yi, and Q. Li, in *Second ACM/IEEE Symposium on Edge Computing (SEC)*, 2017.

[55] P. S. Junior, 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.

[56] M. KABLAN, A. ALSUDAIS, E. KELLER, and al., "Stateless network functions: Breaking the tight coupling of state and processing," 2017.

[57] R. Gember-Jacobson, C. Viswanathan, R. Prakash, and al., "Opennf: Enabling innovation in network function control," 2014.

[58] S. AJAGOPALAN, D. WILLIAMS, H. JAMJOOM, and al., "A. split/merge: System support for elastic execution in virtual middleboxes," 2013.

[59] S. Woo, J. Sherry, S. Han, and al., "Elastic scaling of stateful network functions," 2018.

[60] S. G. Kulkarni, G. Liu, K. K. Ramakrishnan, and al., "Reinforce: Achieving efficient failure resiliency for network function virtualization-based services," 2020.

[61] D. Ongaro, S. M. Rumble, and R. S. al., "Fast crash recovery in ramcloud," in *23rd ACM Symposium on Operating Systems Principles (SOSP)*, 2011.

[62] S. Ramanathan, K. Kondepu, T. Zhang, and al., "A comprehensive study of virtual machine and container based core network components migration in openroadm sdn-enabled network."

[63] S. V. N. Kotikalapudi, "Comparing live migration between linux containers and kernel virtual machine: investigation study in terms of parameters," 2017.

[64] M. J. M. Jay), "Why use containers and cloud-native functions anyway?" in *White Paper Communications Service Providers Cloud-Native Network Functions.* Intel.

[65] H. Liang, Q. Zhang, M. Li, and J. Li, "Toward migration of sgx-enabled containers," in *2019 IEEE Symposium on Computers and Communications (ISCC).* IEEE, 2019, pp. 1–6.

[66] A. Ahmed, H. Arkian, D. Battulga, A. J. Fahs, M. Farhadi, D. Giouroukis, A. Gougeon, F. O. Gutierrez, G. Pierre, P. R. Souza Jr *et al.*, "Fog computing applications: Taxonomy and requirements," *arXiv preprint arXiv:1907.11621*, 2019.

[67] S. Maheshwari, S. Choudhury, I. Seskar, and D. Raychaudhuri, "Traffic-aware dynamic container migration for real-time support in mobile edge clouds," in *2018 IEEE International Conference on Advanced Networks and Telecommunications Systems (ANTS).* IEEE, 2018, pp. 1–6.

[68] B. Xu, S. Wu, J. Xiao, H. Jin, Y. Zhang, G. Shi, T. Lin, J. Rao, L. Yi, and J. Jiang, "Sledge: Towards efficient live migration of docker containers," in *2020 IEEE 13th International Conference on Cloud Computing (CLOUD).* IEEE, 2020, pp. 321–328.

[69] T. Benjaponpitak, M. Karakate, and K. Sripanidkulchai, "Enabling live migration of containerized applications across clouds," in *IEEE INFOCOM 2020-IEEE Conference on Computer Communications.* IEEE, 2020, pp. 2529–2538.

[70] Z. Mavuş, "Secure model for efficient live migration of containers," Master's thesis, 2019.

[71] M. Azab, B. Mokhtar, A. S. Abed, and M. Eltoweissy, "Toward smart moving target defense for linux container resiliency," in *IEEE 41st Conference on Local Computer Networks (LCN).* IEEE, 2016, pp. 619–622.

[72] R. Huang, H. Zhang, Y. Liu, and S. Zhou, "Relocate: a container based moving target defense approach," in *7th International Conference on Computer Engineering and Networks*, 2017, p. 8.

[73] M. Azab and M. Eltoweissy, "Migrate: Towards a lightweight moving-target defense against cloud side-channels," in *2016 IEEE security and privacy workshops (SPW)*. IEEE, 2016, pp. 96–103.

[74] C. Dupont, R. Giaffreda, and L. Capra, "Edge computing in iot context: Horizontal and vertical linux container migration," in *2017 Global Internet of Things Summit (GIoTS)*. IEEE, 2017, pp. 1–4.

[75] L. Ma, S. Yi, N. Carter, and Q. Li, "Efficient live migration of edge services leveraging container layered storage," *IEEE Transactions on Mobile Computing*, vol. 18, no. 9, pp. 2020–2033, 2018.

[76] A. Machen, S. Wang, K. K. Leung, B. J. Ko, and T. Salonidis, "Migrating running applications across mobile edge clouds: poster," in *Proceedings of the 22nd Annual International Conference on Mobile Computing and Networking*, 2016, pp. 435–436.

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

[78] A. Mirkin, A. Kuznetsov, and K. Kolyshkin, "Containers checkpointing and live migration," in *Proceedings of the Linux Symposium*, vol. 2, 2008, pp. 85–90.

[79] M. Planeta, J. Bierbaum, L. S. D. Antony, T. Hoefler, and H. Härtig, "Migros: Transparent operating systems live migration support for containerised rdma-applications," *arXiv preprint arXiv:2009.06988*, 2020.

[80] "Velero. https://velero.io/," accessed: 17-Nov-2021.

[81] "Ansible. https://www.ansible.com/," accessed: 06-Dec-2021.

[82] "HAProxy - The Reliable, High Performance TCP/HTTP Load Balancer. http://www.haproxy.org/."

[83] "Open Container Initiative. https://opencontainers.org/," accessed: 24-Oct-2021.

[84] "Intel SGX: Enclave. https://www.intel.com/content/ dam/ develop/ external/ us/en/documents/ overview-of-intel-sgx-enclave-637284.pdf," accessed: 17-Nov-2021.

[85] N. Mizusawa, K. Nakazima, and S. Yamaguchi, "Performance evaluation of file operations on overlayfs," in *2017 Fifth International Symposium on Computing and Networking (CANDAR)*. IEEE, 2017, pp. 597–599.

[86] "Restic. https://velero.io/docs/v1.7/restic/," accessed: 17-Nov-2021.

[87] "Velero GitHub. https://github.com/vmware-tanzu/velero," accessed: 17-Nov-2021.

[88] K. Govindaraj and A. Artemenko, "Container live migration for latency critical industrial applications on edge computing," in *2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA)*, vol. 1. IEEE, 2018, pp. 83–90.

[89] "Cloudify [Official site]. https://cloudify.co/," accessed: 17-Nov-2021.

[90] "KubeVirt [Official site]. https://kubevirt.io/," accessed: 17-Nov-2021.

[91] "KubeVirt GitHub. https://github.com/kubevirt/kubevirt," accessed: 17-Nov-2021.

[92] "Podmigration-operator https://github.com/schrej/podmigration-operator," accessed: 17-Nov-2021.

[93] "Podmigration-operator [Extended version]. https://github.com/ssu-dcn/podmigration-operator," accessed: 17-Nov-2021.

[94] "Kubernetes. https://kubernetes.io/."

[95] "Edge Multi-Cluster Orchestrator (EMCO). https://smart-edge-open.github.io/ido-specs/doc/building-blocks/emco/smartedge-open-emco/."

[96] "emco-base [Gitlab]. https://gitlab.com/project-emco/core/emco-base."

[97] R. van der Meulen, "What edge computing means for infrastructure and operations leaders," *Gartner, online, available, www. gartner. com*, 2017.

Table 3: Comparison of various container migration techniques

Ref.	Type	Live/ Cold	Archi.	Scope	Factors to handle
[68]	Pre-copy	Live	Cloud	Avoid duplicate Docker image layers transmission, manage container context	Migration downtime
[69]	Pre-copy	Live	Cloud	Automate live migration using Ansible along with traffic redirection	Migration time
[70]	Stateful and stateless	Live	Cloud	Protect from malicious attack	Migration time, Application downtime
[71]	-	Live	Cloud	Protection from malicious attack	-
[72]	-	Live	Cloud	Defensive approach against information leakage attack	Time & space migration
[73]	Pre-copy	Live	Cloud	Migrate VM/containers accross physical hosts and complicate the attacker process of placing VM/containers in the same victim/host	-
[55]	Pre-copy	Live	Fog	Transmit the least modified files before the actual migration from one fog node to another	Downtime
[74]	Stateless	Live	Fog	Support both horizontal and vertical migration	-
[75]	Pre-copy	Live	Edge	Reduce size of the file(s) to transfer, consider user's movement while migration	Migration time
[76]	Pre-copy	Live	Mobile Edge Computing (MEC)	Consider users location and select the nearest node to map container/VM	Service downtime, Migration time
[77]	Post-copy	Live	Cloud	Provide Just-In-Time (JIT) migration to access the data at target host during lazy data copying process running in background	Downtime, Performance overhead - read, write, update, scan workload
[78]	Pre-copy	Live	Cloud	Perform check-pointing and restart procedure for containers at the kernel-level ; facilitate the check-point and restoration of the running container state	Downtime
[65]	Post-copy	Live	Cloud	Allow migration of Intel SGX-enabled container used to protect data from untrusted access)	Migration time
[79]	-	Live	Cloud	Migration for RDMA-enabled containerized application	Analyse required modification in implementation, Migration time
[80]	Pre-copy	Live	Fog	Can integrate with Kubernetes clusters ; allow backing up, restoring of states and migration from one Kubernetes cluster to another	Backup and restoration of resources
[62]	Stateful	Live	Edge	Container migration of the following network functions that are not supported by current CRIU and OpenAirInterface: Home Subscriber Server (HSS), Mobility Management Entity (MME), and Serving and Packet Gateway (SPGW)	Migration time, Downtime