Computer Networks 181 (2020) 107429

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

Virtual Network Functions Migration Cost: from Identification to Prediction

Rafael de Jesus Martins [a,*], Cristiano Bonato Both [b], Juliano Araújo Wickboldt [a], Lisandro Zambenedetti Granville [a]

[a] Federal University of Rio Grande do Sul, Brazil
[b] University of Vale do Rio dos Sinos, Brazil

ARTICLE INFO

Keywords:
Function virtualization
Orchestration costs
Migration prediction

ABSTRACT

The advent of the function virtualization concept, especially that of network functions, leads to important benefits for future networks. Although the orchestration of virtualized functions presents gains for network operators and clients alike, the overhead for moving functions has not been thoroughly explored so far, especially considering functions virtualized by using container technologies. In this work, we investigate orchestration costs associated with the migration of containerized virtual functions. To this end, we first perform a systematic literature review on state-of-the-art virtual function migration costs, electing time and data transferred as so. We then use a well-known container platform (LXD) to perform several orchestration experiments in a controlled environment. By analyzing the container migration process in smaller complementary steps, and designing experiments to evaluate them individually, a pattern for migration costs is observed. Linear regression is then used to derive a prediction model for the necessary time and data transferring for performing a container migration. To assess the predictor's accuracy, we present a cloud computing use case where the predictor is deployed. Results indicate that predictions can be accurate within reasonable range, and therefore orchestration algorithms may be improved by accounting for similar prediction models when determining the migration of one or more virtualized functions.

1. Introduction

Future networks, as exemplified by the forthcoming 5G services for Extreme Mobile Broadband (xMBB) and Massive Machine Type Communications (mMTC) [1], are being built upon technologies such as Software-Defined Networking (SDN) and Network Functions Virtualization (NFV) [2]. In the NFV paradigm, network functions that traditionally rely on specialized equipment (also called middle-boxes) become executable by software running on general purpose hardware. As a consequence, the possibilities for network management are vastly expanded, since the deployment of a given virtual network function (*e.g.*, Firewall) no longer depends on the acquisition of specialized physical hardware. Moreover, not only a Virtual Network Function (VNF) can be easily deployed along the network, but also computational resources for VNFs can be dynamically scaled up and down in response to instantaneous demand.

Resource virtualization offers users a series of benefits, such as hardware independence, readiness of deployment, and elasticity of resources. In contrast to traditional virtualization realized by Virtual Machine (VM), newer container virtualization technologies emerge with the promise of lighter, more efficient virtualization [3]. Several studies have compared the effectiveness, particularly regarding performance and scalability, of virtualization by VMs and container technologies [4,5]. By sharing the Operating System (OS), notably the OS's kernel, with the underlying host, container technologies reduce much of the virtualization overhead in comparison to the heavier footprint of full-fledged VMs, making container virtualization a feasible option for network environments from IoT cloud/fog [6] to High-Performance Computing (HPC) [7]. In particular, running and migrating functions in more resource-constrained environments, *e.g.*, devices on the network edge, can make lighter virtualization technologies such as containers a need, rather than an option [8].

In the context of NFV, studies have been carried out to try to solve problems of VNF composition, chaining, and placement [9], as described next. Because VNFs can be composed of smaller, reusable components, known as VNF Components (VNFCs), a single VNF can be composed in multiple ways [10]. Furthermore, two or more VNFs can be interconnected to provide a higher-level service in a process known as Service Chaining (SC). Moreover, because VNFs and VNFCs are typically designed to run on top of commodity hardware, NFV-enabled networks tend to offer multiple host options for each VNF or VNFC; selecting the best host to run each function is a research topic usually referred to as the placement problem. While promising heuristics to solve the place-

* Corresponding author.
E-mail addresses: rjmartins@inf.ufrgs.br (R.d.J. Martins), cbboth@unisinos.br (C.B. Both), jwickboldt@inf.ufrgs.br (J.A. Wickboldt), granville@inf.ufrgs.br (L.Z. Granville).

https://doi.org/10.1016/j.comnet.2020.107429
Received 10 March 2020; Received in revised form 8 June 2020; Accepted 15 July 2020
Available online 21 July 2020

ment problem have been proposed so far, the jury is still out on a definitive solution due to the problem being an NP-complete one [11]. Previous studies have investigated the issue for VNF composition [12], chaining [11], and placement [13] through a theoretical-based approach. Although theoretical research has been producing increasingly optimized solutions, we argue that hardware (*e.g.*, network bandwidth) and software (*e.g.*, memory dirtying frequency) constraints must be taken into account when adopting such solutions in the real world [14].

VNF orchestration, *i.e.*, the decision-making to initialize, stop, or migrate a VNF from one host to another, plays a fundamental role in optimizing the usage of available network resources, and improving energy efficiency [15]. In particular, the decision to migrate a VNF can be based on several factors, such as due to a load-balancing algorithm or hardware maintenance, for example [16]. The decision of which functions to migrate, and in which way, has been subject of several studies [17–19]. However, orchestration algorithms proposed so far consistently fail to consider the exact overhead introduced by migrations themselves when defining an orchestration plan. Previous works have considered the overhead in performing orchestration steps, such as attaining a system's state prior to determining an orchestration plan [20], and designed orchestration algorithms similarly tend to disfavour successive VNF migrations due to their intuitive cost [21,22]. Still, determining what the costs really are and estimating how impactful they will be for an individual migration have not received as much attention thus far. For example, when applying a load-balancing algorithm, the network transfer required by a migration may run on top of an already congested link, reducing the migration effectiveness. To improve the decision-making process, an orchestrator should thus consider the expected cost of carrying out an orchestration plan, especially the cost of migrating a VNF [23].

Proposals for VNF orchestration optimization often include some predictive model to enhance their performance. When used as input for the orchestrator, these predictions can allow a more proactive orchestration, in contrast to the usual reactive approach, thus further improving algorithms' performance. For example, the expected variance on demand for a specific VNF can be used as input for a better-informed orchestration decision [24]. Similarly, other virtualized environments have used power consumption prediction with high accuracy [25]. Regarding Service Function Chains, orchestrators that try and predict how functions placement can interfere with their performance, and therefore how to optimally place them, have been proposed with promising results [26]. However, although the trade-off for VNF migration is sometimes considered [27], to our best knowledge there is still no study on developing a prediction model for the migration costs themselves.

In this work, we investigate the problem of costs associated with VNF migration, and further, as the main contribution of this work, propose a model to estimate migration costs before the operation is performed. We chose a well-known container virtualization platform, namely, LXD[1], to carry out our experimental studies, analyzing how multiple variables affect the migration process. By combining the theoretical background with the observed experimental results, we propose a mathematical analysis based on linear regressions to predict the costs in terms of time and data transferred associated with VNF migration. To test our predictor, a cloud computing use case is presented. Based on our results, cost prediction for VNF migrations is possible with high accuracy (upwards of 80%), considering a reasonably narrow prediction range, and thus future VNF orchestrators can benefit from including similar prediction model features to weigh in not only the expected benefit from migration alternatives, but also the expected costs for executing each one of them.

The remainder of this article is structured as follows. In Section 2, we present related work on cost-estimation for VNF migration. In Section 3, the theoretical background and the experimental setup used throughout this study is presented, as is an initial experiment to define a baseline for migration costs. In Section 4, we present experiments that demonstrate how the nature of a function can impact the migration costs. In Section 5, a quantitative analysis of the experiments' results is discussed, and a mathematical model derived for predicting migration costs. In Section 6, we present a use case to evaluate our predictor. In Section 7, we discuss our conclusions and present future work.

2. Related work

In this section, we present the state-of-the-art on VNF migration costs. This overview not only assists in confirming that the problem exists, but also allows the identification of metrics we later use to determine migration costs, namely, time and data transferred. To do so, a systematic literature review was performed. Selected studies were grouped according to network environment, and laid out as follows. Cloud computing studies are discussed in Section 2.1. Content Delivery Networks (CDNs) studies are shown in Section 2.2. SDN, 5G, and other network environments are discussed in Section 2.3.

2.1. Cloud-Fog-Edge computing

Cloud/fog/edge computing are among the most critical scenarios for Virtual Function (VF) orchestration, where cloud operators attempt to meet the services agreement while minimizing their operating expense. One way to achieve that is to deploy an orchestrator in the cloud to migrate VFs along cloud, fog, and edge infrastructure dynamically. This orchestration allows a more efficient use of cloud/fog/edge resources, leveraging their scalability in response to demand fluctuations.

In this context, the work by Cerroni et al. [28] focus on the performance analysis of live migrating VNFs running inside dedicated VMs, where the run-time state of the function is transferred from the source host to the destination host before the migration is completed. As a result, function downtime perceived by the end-user is drastically reduced, even to negligible levels [29]. In their work, Cerroni et al. considered as performance indicators the total time elapsed in a migration and the service's downtime. Similarly to most of the remaining considered researches [30–32], their work uses virtualization through VMs to deploy the VNFs. The reduced virtualization overhead for using containers can positively impact the effectiveness of VFs orchestration when compared to their VMs counterpart [4]. Since we are interested in determining the orchestration impact for the container alternative, the quantitative performance analysis for container orchestration stands as a central research question in our work.

In the same line of work, Tao et al. [30] tackled the problem of dynamically migrating VMs in cloud computing. The authors introduce a proposal to optimize migration plans, *i.e.*, plans for a series of migrations to improve the system performance. Once again, total elapsed time for migration is taken into account when establishing the performance for each plan. The performance evaluation also accounts for the data transferred by VMs migration along the network.

A different approach for the performance analysis of migrating VMs in the cloud was proposed by Lin et al. [32]. While the authors also considered elapsed time for migration to determine its cost, the downtime observed by users during the migration process was also taken into account. The researchers focused on the number of memory pages that needed to be synchronized between hosts as a determining factor for the performance evaluation. The reasoning for doing so is that since the VNF migration relies on memory synchronization from source to destination, VNFs with higher memory dirtying rates may require additional iterations for synchronization, depending on the migration optimization in place. As a result, the migration process for VNFs that produce higher memory dirtying rates requires additional iterations for memory synchronization, which can be of utmost importance when considering migration for applications with such behaviour.

The work by Liu et al. [33] focuses on assisting live streaming in mobile cloud networks. In their scenario, video service migrations along

[1] https://linuxcontainers.org/.

the network are performed to maximize the Quality of Experience (QoE) of the users, while also minimizing the costs for network operators. To achieve that, an orchestrator migrates the service in response to users demand. Due to the application's nature, timing and bandwidth requirements are of vital importance in this scenario, *i.e.*, the maximum acceptable delay and the minimum bandwidth available must be ensured (within reasonable cost) at all times. Therefore, the overhead introduced by a migration must be carefully weighed against the expected benefits. The traffic introduced in the network and the resulting downtime for the service are considered by the authors as the costs associated with the migration.

Container migration in fog computing was subject of study by Tang et al. [34]. The study presents a model that uses container migration in the fog to account for user mobility, and deep learning is used in the migration decision process. Expected duration for a migration, estimated from the size of VF and the available bandwidth between hosts, is used in the migration decision. Finally, a recent work by Ouyang et al. [21] investigates service placement for Mobile Edge Computing, proposing a novel mechanism that optimizes both users' perceived-latency and service migration costs. Even with no assumptions on the virtualization technology used to host the service, the authors considered the main costs for migrating the service to be the bandwidth usage, the service interruption, and the possibility of failure in migrating a service. The NFV relationship with different network environments is not always as straightforward as with cloud/fog/edge computing. Still, network paradigms that focus on data replication and reallocation along the network can also benefit from NFV concepts, particularly when not only the content, but the services and the network themselves can be virtualized and dynamically reallocated. Therefore, in the next subsection, we delve into studies on CDNs.

2.2. Content delivery network

CDNs improve applications and services by distributing available content (*e.g.*, multimedia files) along the network according to user demand [35]. On top of that, virtual CDNs virtualize servers and networks themselves, using SDN and NFV paradigms, further improving services scalability [36]. By improving the management of resources, the employment of SDN and NFV by virtual CDN can positively impact users' QoE without burdening the network operating cost.

In a series of studies [19,37,38], Ibn-Khedher et al. investigated the optimization of orchestration algorithms for CDNs. The authors propose cost-efficient algorithms for the placement and orchestration problem of virtual nodes in a CDN. In their experiments, the researchers used VMs to virtualize the network's content. The main costs considered for orchestration was the total elapsed time, and the downtime of the migrated service.

Jahromi et al. [39] present a significant study for VNFs placement and chaining in CDNs. In their work, authors consider a multiple cost model that takes into account VNF instances and migrations when performing network alterations. In particular, migration cost is discussed in the placement model, and defined as the high transportation cost for moving a VM.

SDN and NFV benefits for CDNs are a direct result for their ability to maximize the efficiency in content reallocation and replication over the CDN hosts. By combining both paradigms, a network service that relies on continuous network monitoring and content orchestration can minimize its operating costs, while improving customers satisfaction. Other selected studies for different network environments that can also benefit from SDN and NFV, individually or in conjunction, are discussed in the next subsection.

2.3. SDN, 5G, And other network paradigms

In SDNs, a centralized controller with global view of the programmable network is used by network carriers to improve network scalability, Quality of Service (QoS), and service management [40]. In conjunction with NFV, traffic steering performed through SDN controllers can both prevent the necessity for VNF migrations, by better balancing traffic load between links, and ease up the VNF migration process, by optimizing the VNF transfer through the network, and assisting the subsequent re-establishment of VNF connections.

Employing SDN in a 5G scenario, Ksentini et al. [2] investigated costs associated with Serving Gateway (SGW) reallocation. SGW in a 5G network is responsible for forwarding users' data traffic [41], making the SGW placement problem an important one in this context. In their work, multiple VNF instances are deployed in the network, and an SDN controller is responsible for migrating users between the instances. The proposed cost for these migrations is the required signaling messages at the SDN controller to synchronize the migration. This solution, therefore, is not available for every network environment (*e.g.*, networks with no SDN support). In another study, similar results were found by Wang et al. [42], using a more generic network with SDN support to deploy a load-balance algorithm in the network.

An important study by Moradi et al. [43] propose SoftBox, a promising novel 5G core network architecture based on SDN and NFV. In the proposed architecture, VNF migrations are leveraged in provisioning a flexible and scalable cellular network, with consideration for the migration costs, which are summarized as the control load on the global controller. Still on 5G core network design, a study by Jin et al. [44] considered NFV and SDN for optimizing mobility management through a virtual Mobility Management Entity (MME). The optimization method developed is designed to minimize overhead migration cost for VNF components of MME. The overhead is defined as the data migrated between hosts, and the proposed method evaluated through simulation.

Other investigations tackled the problem for VNF migration in less specific networks. The study by Sun et al. [24] investigates the issue with NFV SC. SC is useful when two more VNFs require a connection between them (even when functions run on the same host). The optimization for the placement problem of a VNF on chaining is thus tied to their VNF neighbors in the chain. To minimize the transferring of VNFs on the network, which Sun et al. consider as the main orchestration cost, the authors look to forecast variances on demand for VNFs in an SC, thus reducing the need for subsequent migrations in response to demand increase. The authors show results from simulations to compare the performance for different proposals.

Optimizing the migration process for VMs across Wide Area Network (WAN) boundaries has been subjected to a study by Le et al. [45]. In their proposal, the new Internet standard of Multi-Path TCP is leveraged to optimize the live migration process, improving it, particularly for the WAN case. Metrics considered by the authors for the migration overhead are the migration time, the service downtime, the application overhead (due to the additional load by migrating the application), and the amount of data transferred.

Lin et al. [46] present a mechanism for VNF placement in data-centers, including VM migration between physical hosts. The authors consider the network throughput added from mirroring during migration as the overhead for migration. Furthermore, in the context of VM migration in a data-center, Fernando et al. [47] studied the failure in performing said migrations using a post-copy technique. In this scheme, the CPU execution state of the VM is migrated and resumed first, and memory pages are retrieved from the source when needed (*i.e.,* upon page-faults). This technique thus allows for diminished migration time and service downtime when migrating a service, although an increased risk for migration failure is introduced. Therefore, the authors propose a solution that combines pre and post-copy migration techniques, and a mechanism of incremental checkpointing, to combine the best of both worlds while reducing the migration overhead. Furthermore, the solution requires additional setup by available hosts and does not include a prediction mechanism for determining the expected cost for a given VM migration. Optimizing the migration process across data-centers was investigated by Shi et al. [48]. The authors propose a system that predicts

Table 1
Summary of Related Work.

		Considered as Migration Cost			
Reference	Network Environment	Migration Time	Data Transferred	Service Downtime	Message Signaling
[28]	Cloud/Fog/Edge	X		X	
[30]	Cloud/Fog/Edge	X	X		
[32]	Cloud/Fog/Edge	X		X	
[33]	Cloud/Fog/Edge	X	X	X	
[34]	Cloud/Fog/Edge	X	X		
[21]	Cloud/Fog/Edge	X	X	X	
[37]	Virtual CDN	X	X		
[38]	Virtual CDN	X	X		
[19]	Virtual CDN	X	X		
[39]	Virtual CDN		X		
[2]	5G				X
[43]	5G				X
[44]	5G		X		
[42]	SDN				X
[51]	SDN				X
[49]	SDN	X			X
[50]	SDN	X	X		
[31]	Generic	X	X		
[24]	Generic		X		
[45]	WAN	X	X	X	
[46]	Data-center		X		
[47]	Data-center	X		X	
[48]	Data-center	X	X		

disk read and memory write working sets before migrating a VM, and combines pre- and post-copy migration to get the best from both worlds. Their results show that migration costs in terms of service downtime and total migration time are shortened with their system. While the system tries to predict the nature of the working set before performing a migration, the cost of migrating the service itself is not predicted by the system.

The study by Zhang et al. [49] analyses distance between the source and the destination servers in the migration costs, in addition to the delay of updating flow rules along with the network, which is done gradually to avoid network congestion. Zhou et al. [50] present an essential study on VNF migration effect, and how to minimize it, in SDN environments. The authors consider that the live migration of a VM consumes most of the bandwidth and computational resources, which can impact negatively in the QoS of the service during migration. Additionally, to the network effect, time to migrate is also considered a migration cost in the study. Simulation results show that the proposed algorithm can reduce some of the migrations costs while optimizing QoS. Finally, Xia et al. [51] also considers the associated costs of migrating a VNF in a VM. The aggregated traffic on the SDN controller is pointed at the overhead for the migrations, as the authors show in simulation results. In the next subsection, a summary for the related work presented in this section is laid out. We also highlight how the results affect our research.

2.4. Influence in our research

The main aspects of the state-of-the-art are summarized in Table 1. As shown by our research, the problem of determining orchestration costs is one that has been drawing attention in the area. Nevertheless, the research field is still young, so investigations tend to be mostly theoretical, resorting to simulated experiments to analyze their proposals. When using a real NFV-enabled network, virtualization was mostly done using VMs, highlighted by the fact that only three [2,34,43] studies explicitly considered container virtualization in their evaluation. In this work, we focus on evaluating the migration costs in a real network, using container virtualization. Our findings indicate that migration costs can be measured and predicted accurately, something that has been overlooked by other works. Considerations on the experimental setup used, an algorithmic view of the migration process, and baseline experiments, are presented in Section 3.

Further, we elected migration time and migration data transferred to compose the migration cost for this work. Not only they align with our initial suspicion, but also both were the two variables most recurrently found in related work. Moreover, our experimental network offered limited SDN support, so the inclusion of SDN in our setup would require some network emulation (*e.g.*, using Mininet [52]), which would defeat our purpose of experimenting strictly with real equipment. Concerning service downtime, the early investigation indicated that its measuring would require an active agent, which could interfere with the cost measurements for the main variables (*i.e.*, time and data transferred), and therefore is singled out for future work. Finally, it is important to notice that performing other aspects of function orchestration such as re-chaining a service chaining may also incur in additional overhead. These are also singled out for future work, as this work focuses on the migration process and the costs it introduces by themselves.

3. Container migration and baseline experiment

In this section, we present some theoretical background on the migration process and the setup used in our experiments. Although several benchmarks for VMs and containers are found in the literature, we goal to draw a baseline for the costs of migration for comparison. In Section 3.1, we divide the migration process into smaller parts, so we can better understand how each part weighs in the total cost for a migration. We then lay out the experimental setup we utilized in our experiments, in Section 3.2. Considerations on available distribution for container's OS and their impact on migration costs are presented in Section 3.3. Finally, in order to create a baseline for the experiments conducted in the following sections, we present preliminary experiments in Section 3.4 that show the minimal cost expected from each migration step.

3.1. Decomposing the migration process

Live migration[2] is a network feature for virtualization which enables seamless VNFs reallocation by transferring the virtual node's persistent

[2] In this work, unless explicitly stated otherwise, migration refers to live migration.

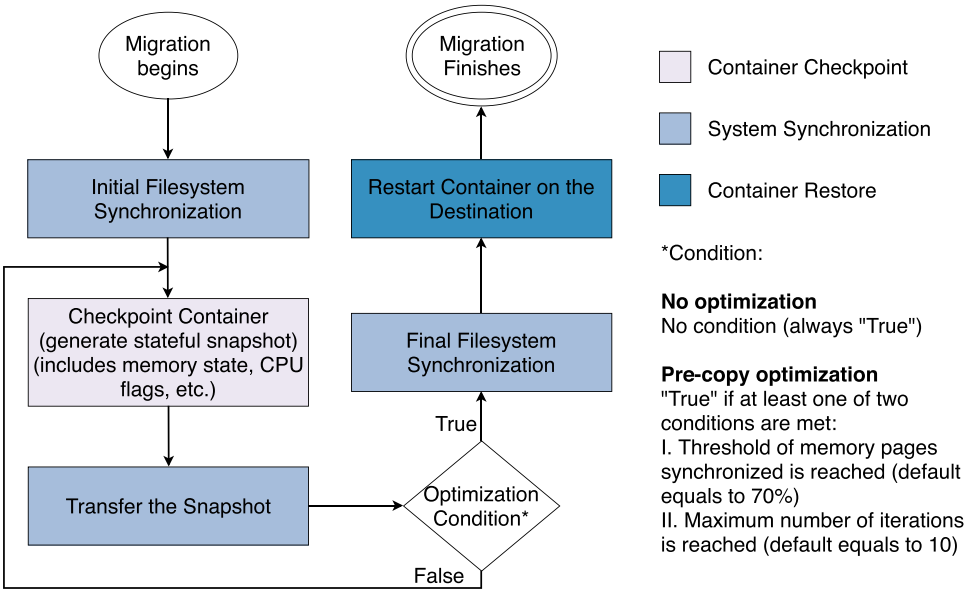


Fig. 1. Flowchart of the migration process.

and run-time state between hosts [53]. The benefits from this process include minimal disruption of the service, since the downtime for the service, that is, the time interval between the node being stopped in the source and resumed in the destination, is kept to minimal levels. In contrast, cold migration requires the service to be fully stopped in the origin before migration, causing a disruption that may be unacceptable for some services. To perform a live migration, origin and destination hosts must synchronize all data required for the service to resume its operation, which includes the file system, memory pages, TCP connections, etc. As stated previously, migration optimizations have been proposed to diminish its costs in specific scenarios, such as with NFV and SDN pairing [54]. Although some proposed optimization techniques hold promise for the container migration landscape, our work is centered on the general, mostly-applicable context, *i.e.*, with LXD built-in pre-copy optimization.

For functions virtualized by LXD containers, migration is composed mainly of three parts: container checkpoint (also known as a container snapshot), system synchronization, and container restore. Parallelism and optimization can be utilized to improve migration efficiency; as an example, service downtime can be reduced at the expense of total migration time by enabling memory pre-copy optimization. The migration work-flow is shown in Fig. 1.

To synchronize the container between hosts, LXD makes use of three traffic channels: the control stream, the Checkpoint/Restore In Userspace (CRIU)[55] stream, and the file system stream[3]. The control stream initiates the migration, and is responsible for exchanging information about the container, its configuration, and the result for the restore operation. CRIU is the tool used by LXD to checkpoint and resume a container's run-time state, and therefore the CRIU stream is used to transfer the container checkpoint data, synchronizing the container run-time state. The file system stream is used to synchronize the container file system between hosts, as one would expect.

3.2. Experimental environment

The topology used in this work's experiments is presented in Fig. 2, where the VF running is experiment-specific. Four virtual servers are configured in two physical hosts, and in our setup, these are realized by conventional VMs. The virtual servers specs, unless stated otherwise, are identical, as seen in Table 2. The simplicity in the design of the experimental setup is intended so that we can single out the interfering variables in the experiments and develop an initial cost analysis and prediction model, that later can be expanded to more complex scenarios.

Finally, differences in the container platform configuration can impact the proposed metrics. For example, storage back-end options provided by LXD include Directory (Dir), B-tree file system (Btrfs), and Zettabyte File System (ZFS). Distinct storage back-end can offer different features and therefore differ in performance, particularly for the tasks of container checkpoint and synchronization. In our experiments, we utilized LXD 3.0.3, CRIU 3.10, and ZFS 0.7.5 for containers' storage back-end.

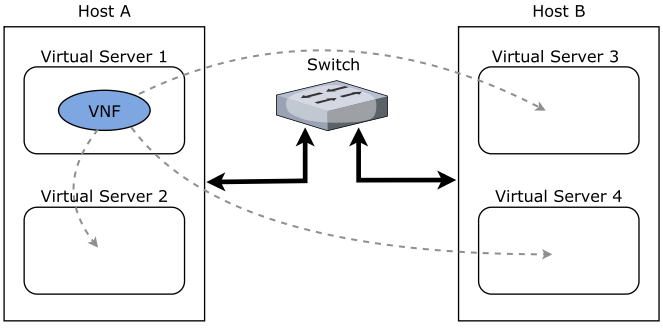


Fig. 2. Overview of the experimental setup.

Table 2
Test Environment Configurations.

Physical Hosts (Hosts A, B)	
OS Version	Ubuntu 18.04.4 LTS
Linux Kernel	4.15.0–101
Network Link	Gigabit Ethernet
Virtual Hosts (VMs 1–4)	
OS Version	Ubuntu 18.04.4 LTS
Architecture	x86_64
Linux Kernel	4.15.0–101
Memory	16 GB
CPU Cores	6
CPU Frequency	2.0 GHz

[3] https://lxd.readthedocs.io/en/latest/migration/.

Table 3
Shortlist for available Linux distributions, with respective image and filesystem sizes.

Linux Distribution	Image Size (MB)	Filesystem Size (MB)
Alpine 3.8	1.92	4.1
Debian 7	91	144
Ubuntu 16.04	105	173
Ubuntu 18.04	119	194
Ubuntu 18.04 (i686)	121	196

3.3. Container operating system

LXD differs from application containers (*e.g.*, Docker [56]) by offering OS-level virtualization, and therefore a virtualization environment similar to that of a full-fledged VM, but with reduced overhead. This lighter VM-like virtualization is achieved by the sharing of the host's kernel among containers, which are isolated, secured, and have their resources managed through kernel *Namespaces* and *cgroups* [57]. Several distribution images are thus offered by LXD when deploying containers, including multiple OSs and architectures. A non-exhaustive list for available distributions is shown in Table 3.

When deploying a new container, the trade-off between services offered and image size should be made accounting for the nature of the virtualized service. Ideally, applications that are expected to be frequently reallocated should rank lighter distributions higher, as to minimize the reallocation overhead; rather static applications could prefer distributions that are easier to work with, as the overhead introduced by pre-installed services, for example, is not as big of a concern. Regarding the lighter container images, studies have shown its performance to be comparable (often better) than Unikernels and other light virtualization alternatives [58]. While it is clear that the distributions difference in filesystem size will impact migration costs (for data transferred, and as result, for time), it is also noteworthy that a migration will also have to copy the base image to the destination host if it had not been previously done so, thus further increasing the costs for such migration. Moreover, the specifics of the applications running in a container will determine how meaningful to the migration costs the filesystem synchronization is in comparison to the run-time state synchronization.

3.4. Baseline experiment: Cold migration x live migration

Contrary to live migration, in cold migration there is a non-negligible service downtime stopping the VNF on the origin, moving it to the destination, and then resuming its operation. A series of experiments comparing cold and live migration was conducted to expose the cost incurred by each migration step (*i.e.*, snapshotting, moving, and restoring). For every experiment, we measure migration cost in time and data transferred. Because it is unfeasible to consider every distribution available, this experiment was restricted to three OSs: Alpine 3.8, Debian 7, and Ubuntu 16.04.

Each container was kept in their minimal state, *i.e.*, no additional applications or services were installed or run on top of their base image. By not installing nor running any additional application in any container, we can single out how the differences in distributions impact each step of the migration process. Each experiment was run 30 times. The cold migration results for the time consumed by snapshotting, transferring, and restoring the containers are shown in Fig. 3a. Regarding the snapshot time, Alpine and Debian containers are virtually tied at just under four seconds, with Ubuntu container taking about 35% more time to complete. Pre-installed services on each distribution, which results in a different number of processes running in each system (5 for Alpine, 4 for Debian, and 10 for Ubuntu), could help explain the difference observed in the results.

Based on the time taken to migrate the stopped container, including the checkpoint state, Debian and Ubuntu containers are much more costly than the Alpine alternative; that is expected, of course, as a direct result from the disparities in file system and image sizes between distributions, as presented in Table 3. The restore time shows a similar pattern to the snapshotting one, although the Debian container fared proportionally worse than the Alpine. The difference in cold migrating each distribution between hosts is further testified by the comparison of data transferred, as shown in Fig. 3b. The standard deviation for the data transferred is negligible for all distributions, as expected since the containers are kept at their minimal state. For the same reason, virtually no variation was observed in data transferred comparing cold and live migrations.

Finally, we compare the time performance for live migration with the sum of individual cold migration steps. Because the snapshotting and the synchronizing processes in live migration can run in parallel, the component with the worst result (*i.e.*, the highest completion time) was considered in the cold migration composition. The result is shown in Fig. 3c. It is noteworthy that the cold migration under-performed compared to the live migration for the Alpine and Debian containers, while the opposite was true for the Ubuntu container. Live migration optimization, absent in individual cold migration steps, can be a factor in this discrepancy. Additionally, the underlying OS for the host system, *i.e.*, Ubuntu, could help explain the anomalous behavior for that container for their shared distribution.

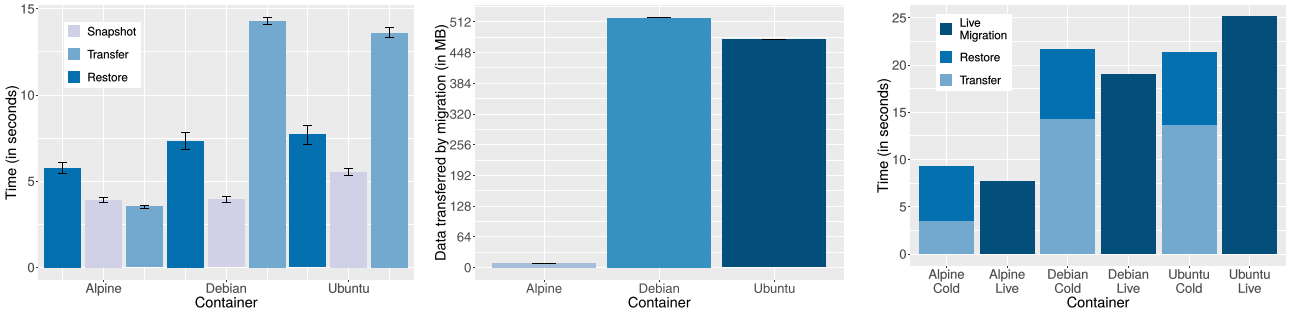
In this section, we described the migration process and the experimental setup used throughout this work, and analyzed how the steps necessary in a container migration are impacted by the OS choice for a container. Since containers were kept at their minimal state for this section's experiments, the results offer a baseline for when a container is used to virtualize any function. Therefore, in the next section, we use one of the tested OSs to carry out experiments evaluating how different virtualized functions impact the migration costs.

4. Black-box analysis: Quantifying migration costs

In this section, we propose two experiments that investigate orthogonal aspects of the migration process: in the first experiment, described in Section 4.1, we measure the impact of the stateless aspect of the migration; complementarily, we describe in Section 4.2 the second experiment, where we measure the effects of the stateful aspect of the migration. Since VNFs can be abstracted as containerized processes that consume resources (*e.g.*, CPU, memory, disk), we consider simplified functions for each experiment. For the first one, we consider a simplified version for a function that primarily consumes stateless resources (*e.g.*, a CDN cache consuming disk space). For the second one, we consider a simplified version for a function that primarily consumes running-time resources (*e.g.*, Deep Packet Inspection (DPI) under heavy traffic). Both experiments are evaluated as black-boxes, where we look to the output response (*i.e.*, migration costs) when varying a VNF controlled variable as input. For each experiment, 10 preliminary trials are performed, so that the mean and standard deviation for every input can be estimated. Then, we set a confidence interval at 95% and a precision of 2% to determine the minimal number of trials N required based on the central limit theory [59]. The largest value of N found in each experiment is used to determine the number of repetitions for every input, so that the sample size remains constant in each experiment. The results observed in both experiments motivate our first prediction model for migration costs, presented in Section 4.3.

4.1. File growing experiment

In this first experiment, we focus on the stateless part of the migration, namely the required filesystem synchronization between hosts. To do so, a file inside the container is created, and gradually increases in size along the experiment. The growing file starts at 1MB and increases exponentially (2^n) until 1GB. Based on the results from the preliminary

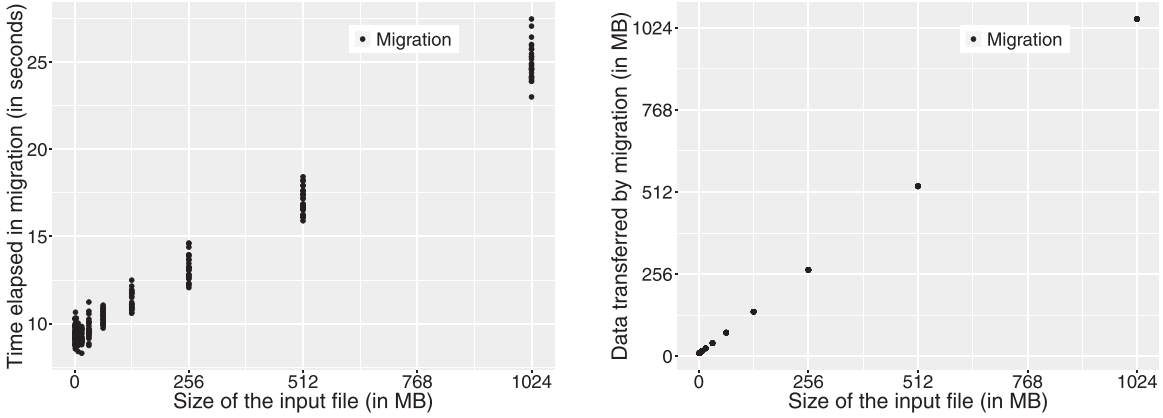


(a) Time elapsed by each step in cold migration.

(b) Data transferred during migration.

(c) Comparison of time between live and composed cold migration.

Fig. 3. Time results for the decomposed migration experiments.



(a) Time (output) versus file size (input).

(b) Data transferred (output) versus file size (input).

Fig. 4. Results for the growing file experiment.

runs, 26 repetitions were performed for this experiment. The results are shown in Fig. 4.

As observed in the results, variance in the measurements for the time elapsed by migrations was noticeable in Fig. 4a, but was negligible for the data transferred in Fig. 4b. For the time cost, the run-time synchronization is not strictly ruled in this experiment, and the coefficient of variation ranged from 0.036 to 0.057. In contrast, the negligible variance in data transferred is somewhat expected (coefficient of variation under 0.0015 for every input value), since this output is mostly defined by our control variable, *i.e.*, the input file size. Nevertheless, both results present a clear linear relationship between output and input, which will be further explored by our initial prediction model in Section 4.3.

4.2. Apache processes experiment

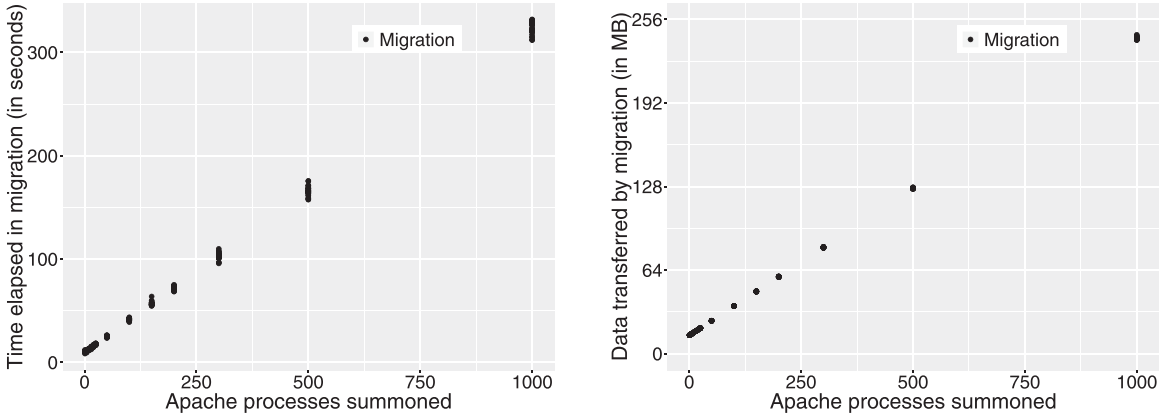
Complementary to the file size experiment presented in the previous subsection, this second experiment focuses on the stateful or run-time state synchronization part of the migration. For this experiment, an Apache Web Server [60] is used to summon an increasing number of processes inside the container. Apache choice arises as a mean to minimize migration failures, occasionally thrown by CRIU when migrating less-supported applications. The number of Apache process summoned in the experiment goes from one to 1000. Based on the results from the preliminary runs, 30 repetitions were performed for this experiment. The results are shown in Fig. 5.

The results in Fig. 4.2 show that, while the values for data transferred in both experiments were in the same order of magnitude, the impact on migration time was much more prominent in the second experiment (Fig. 5a), taking over 10 times as much for the most extreme cases; this result confirms that synchronizing run-time state is an iterative process that can be costly for the migration. Notwithstanding their orthogonal nature, the results from both experiments show a clear pattern for a linear component with both input and output growing exponentially, and with little variance in all results.

4.3. Simple linear regression

A regression model is used to estimate or predict a certain variable as a function of one or more other variables [59]. As a result of the linearity observed in migration costs for previous experiments, we utilize a simple linear regression model to estimate costs for migration on each experiment as function of their input variables. Although machine learning and other more complex techniques could also be valid choices for tackling the problem, the linearity observed in the results indicates that a simpler model may be the fittest candidate for the job. A linear model does not introduce computational overhead, and since it is a comprehensive model that allows the similar conclusion to that of more complex models, it is generally preferred [59]. A mathematical definition of the linear model is presented in Eq. (1).

$$y_i = b_0 + b_1 x_i + \epsilon_i \tag{1}$$



(a) Time (output) versus Apache processes (input).

(b) Data transferred (output) versus Apache processes (input).

Fig. 5. Results for the Apache processes experiment.

Table 4
Summary of the regression fit for the file size experiment.

File size Regression	Adjusted R-squared	p-value	Residual Standard Error
Time	0.9848	$<22E^{-17}$	0.581
Data Transferred	1	$<22E^{-17}$	0.018

In Eq. (1), y_i is the response variable, *i.e.*, the migration costs we considered as output for the experiments. x_i is the variables used as input in the experiments, *i.e.*, the size of the growing file, and the number of Apache processes run. The b_0 term is the y-intercept, *i.e.*, the intersection of the y-axis when x equals to 0, and the b_1 term is the slope coefficient, *i.e.*, the rate of change in y as a result of changes in x. The ϵ term represents the observed error, also known as residual, between expected and measured values. We use the least-squares criterion to find the line given by Eq. (1) that minimizes the Sum of Squared Errors (SSE). In SSE, residuals are squared before being summed, so positive and negative errors do not cancel each other out. This model minimizes the variance of errors, meaning that the best fit for the regression line should prioritize having fewer significant errors, at the expense of producing a higher number of smaller errors.

Results for R-squared, p-value, and residual standard error are considered to assess how well the regression fits the observed data. R-squared, also known as the coefficient of determination, measures the proportion of the variance in the dependent variable that is predictable from the independent variable. The p-value indicates the probability of finding the observed results when the null hypothesis (*i.e.*, the hypothesis complementary to the one considered) is correct; small p-values (<0.05) therefore indicate strong evidence to reject the null hypothesis. The residual standard error measures how close the regression line is to the observed results. Residuals density are plotted using a Gaussian kernel for the density estimation to confirm further that the model is appropriate.

4.3.1. Regression applied to the growing file experiment

Applying the linear regression to the first experiment, we derive Eqs. (2) and 3. The output variable (*i.e.*, y_i from Eq. (1)) in Eq. (2) gives the migration time, in seconds. In Eq. (3), the output variable gives the total data transferred during migration, in MB. The input variable (*i.e.*, x_i from Eq. (1)) in both equations is given by the size of the input file, in MB. The summary for the fits, on 285 degrees of freedom, is presented in Table 4.

$$\text{Migration Time} = 9.25 + 0.02 \cdot (\text{File size}) \quad (2)$$

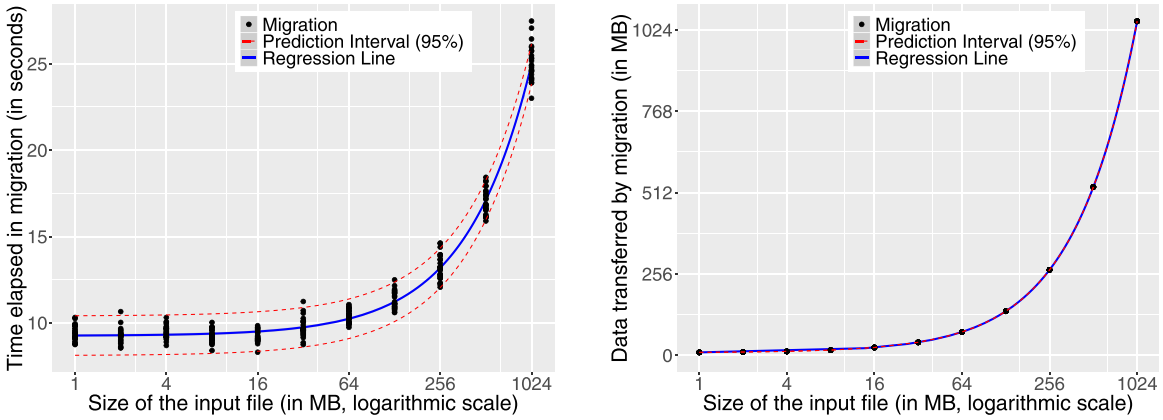
$$\text{Migration Transfer} = 8.16 + 1.02 \cdot (\text{File size}) \quad (3)$$

The result for Eq. (2) is plotted along the experiment's time result in Fig. 6a. A log-scale is used for the x-axis since the growth of the input file is exponential. The blue line represents the given equation; the greyed area around the line indicates the standard error. The closeness of the regression line to observed values, combined with the low standard error obtained, show that the regression offers a reliable prediction model for the experiment. The dotted red lines delimit a 95% prediction interval; this narrow interval indicates that, if we were to rerun the experiment, 95% of the sampled results are expected to fall into this approximate 2.5 seconds range from the regression line.

Similarly, the result for Eq. (3) is plotted in Fig. 6b with the experiment's data transferring result. Due to the negligible variance in the output as a response for each input value, the points for observed values stack on top of each other, as do the standard error and the prediction interval with the regression line. The result shows that the migration cost in terms of data transferred is almost perfectly predictable considering the controlled variable alone in this experiment. The perfect fit is also somewhat expected, as discussed previously, since in this experiment the data transferred is mostly defined by the stateless synchronization of the growing file.

Finally, we analyze the probability density of residuals for each regression in an attempt to assess that the proposed model is appropriate. The kernel density estimation approach used centers a smooth kernel function (in this case, a Gaussian) at each data point, and produce a density estimate by summing them, which helps to see where the values are concentrated. Since residuals represent the error in the regression model, plots ideally should have their peaks at $x = 0$ (most values have close to no error), be symmetrical (positive and negative errors are equivalent), and have the x range be as short as possible (errors are small).

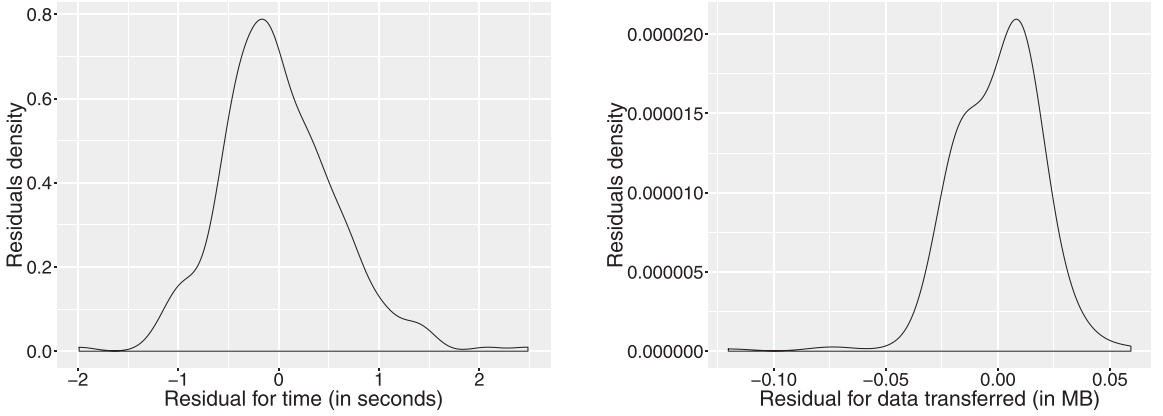
The peak around $x = 0$ in Fig. 7a indicates that the regression line obtained has close to no error with most of the experimental values. Moreover, the graph centered at $x = 0$ shows that positive and negative errors are equally likely. Finally, the vast majority of the residuals are within 1 second, an acceptable range for the predictor. In the case of Fig 7b, the peak around 0 and the short spread shows that most observed values will be close to perfectly on top of the regression line. The negative skew indicated by the longer tail towards the left could indicate some missing variable from the model, but since the values are negligible (less than 0.1% of the migration), it can be safely disregarded.



(a) Migration time as response for file size input.

(b) Data transferred during migration as response for file size input.

Fig. 6. Linear regression for migration costs in the growing file experiment (x-axis in log scale).



(a) Probability density of residuals for migration time.

(b) Probability density of residuals for data transferred.

Fig. 7. Residual plots for the file size linear regression.

Table 5
Summary of the regression fit for the Apache processes experiment.

Apache processes Regression	Adjusted R-squared	p-value	Residual Standard Error
Time	0.9993	$<22E^{-17}$	2.308
Data Transferred	1	$<22E^{-17}$	0.423

4.3.2. Regression applied to the apache processes experiment

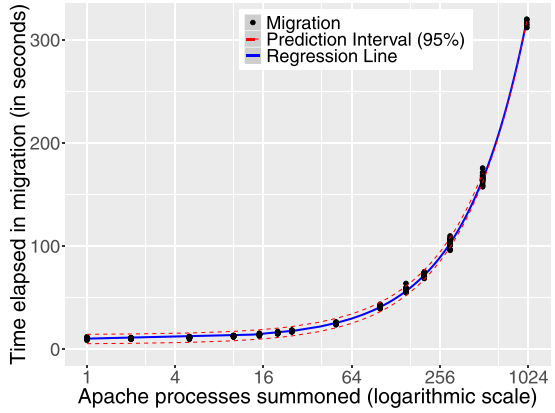
Applying the linear regression to the second experiment, we derive Eqs. (4) and (5). The output variable in Eq. (4) gives the migration time, in seconds, and in Eq. (5) the output variable gives the total data transferred during migration, in MB. The input variable in both equations is given by the number of Apache processes summoned in the experiment. The summary for the fits, on 418 degrees of freedom, is presented in Table 5.

$$\text{Migration Time} = 9.53 + 0.31 \cdot (\text{Apache Processes}) \tag{4}$$

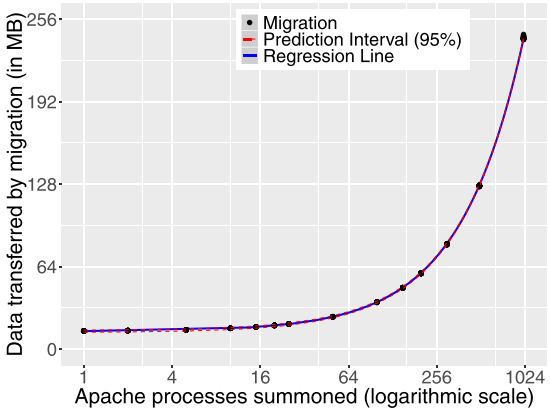
$$\text{Migration Transfer} = 13.72 + 0.23 \cdot (\text{Apache Processes}) \tag{5}$$

It is noticeable that the intercept values are higher in both equations in comparison to the ones in the previous experiment. That is expected, since the first experiment does not require any additional package installation from the minimal container, while in this experiment, Apache and other required packages had to be installed. When comparing the slope coefficients, it is clear that the number of Apache processes has a comparable but smaller effect on the data transferred; however, the number of Apache processes is significantly more costly (over ten times as much) to the migration time when compared to the size of the growing file. Results for the regressions presented in Eqs. (4) and (5) are shown in Fig. 8a and 8 b for the experiment's result regarding time and data transferred, respectively.

The difference in the slope coefficients can be perceived at the y-axis range in Fig. 8a; while in the first experiment the longest migration took just under 30 seconds, the second experiment shows migrations taking upwards of 300 seconds. The prediction of necessary time for a migration must therefore account properly for the overhead in run-time state synchronization. Moreover, the relatively narrow area for the standard error and the prediction interval show that the time variance is not only subject to the migration load, but also to time variances in the migration process itself. Although those large differences were present in the migration cost regarding time heavily dis-

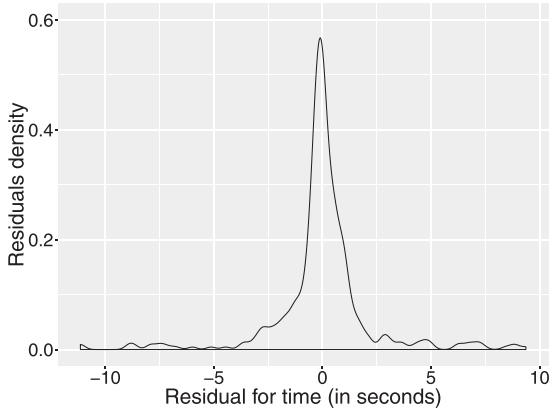


(a) Migration time as response for inputted Apache processes.

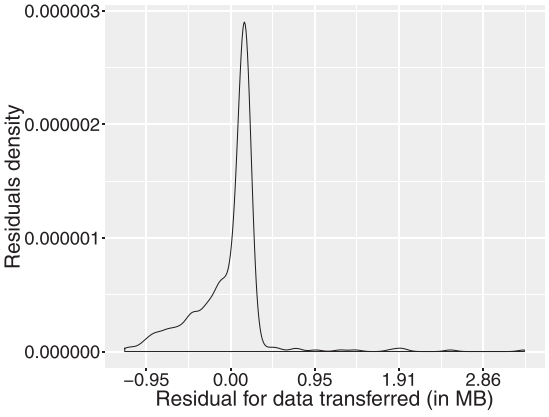


(b) Data transferred during migration as response for inputted Apache processes.

Fig. 8. Linear regression for migration costs in the Apache processes experiment (x-axis in log scale).



(a) Probability density of residuals for migration time.



(b) Probability density of residuals for data transferred.

Fig. 9. Residual plots for the Apache processes linear regression.

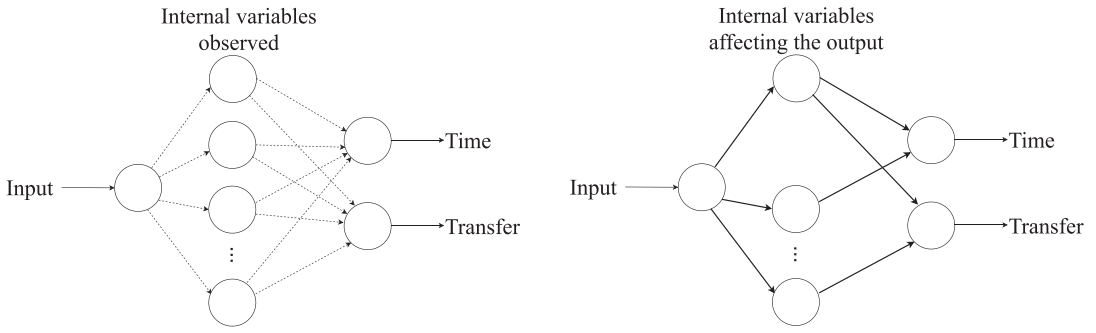
favouring the second experiment, in Fig. 8b it is shown that the cost in data transferred is closer to that observed in the first experiment, but with the first experiment being disfavoured. This behavior highlights the specific impact each experiment has on the migration steps previously presented. In the first experiment, the migration time is closely related to the transferring of the filesystem between hosts. Conversely, in the second experiment, this relationship is not as strictly defined, as a smaller amount of data transfer does not determine a longer time necessary for migration, but on the contrary, the much longer migration time can be explained by the costly process of synchronizing the runtime state in the container migration, as previously discussed regarding Fig. 3.

Finally, we analyze the density of residuals produced by the regressions in Fig. 8, shown in Fig. 9. Regarding the time results, the thin peak by $x = 0$ in Fig 9a shows that most observations have near-zero error. The x-axis span is larger than that obtained for the previous experiment, which is understandable since the absolute values for time migration were over ten times greater in this experiment. Therefore, the relative error is still within acceptable ranges, especially since the tails have close to zero density. Regarding the results for data transferred, the larger, shorter negative tail is contrasted by the thinner, longer positive tail. This result indicates that positive errors are less likely, but are (in magnitude) higher than negative errors. Finally, the value of the residuals (in the x-axis) increased orders of magnitude compared to the results of the previous experiment, which could further indicate a missing variable from the module, and that it affects run-time more than stateless state synchronization. Notwithstanding, the error is still relatively manageable (1% or less) compared to the measured cost; therefore, the model is considered fit.

In this section, we further analyzed the steps performed in a migration. By dividing the migration into two complementary parts, namely the stateless and stateful synchronization, we can better understand how each part is affected by the nature of a particular VNF. To quantitative determine the impact on migration, two separated experiments were performed, one focused on the stateless, the other focused on the stateful part of the migration. Simple linear regression was then done for each result, resulting in promising prediction models for migration costs as response for values in experiments input. While the models obtained help understand the behavior observed in each experiment, the results are too constrained by the experiments' parameters to be generally useful. In the next section, we look into the black-box to determine what internal variables can be used to predict migration costs for VNFs in general.

5. White-box analysis: Predictor modelling

In the previous section, the black-box analysis and the simple linear regression showed promising results for the prediction of migration



(a) A series of internal variables that could be useful as predictors are monitored during experiments.

(b) The relation between input, internal variables, and output, is set out.

Fig. 10. Process of determining system's internal variables which can be used as migration costs predictors.

costs. The derived equations, however, are too narrowly fit to the scope of the experiments. To create a cost prediction model that is useful for any application, we must look into the black-box, and look for system's internal variables that affect the migration costs, and that are available to be used as input for predictions regardless of the application. This process is depicted in Fig. 10. We thus discuss in Section 5.1 how internal variables were selected from each experiment to compose the predictor model, which we propose in Section 5.2 and is fit for a broader range of applications.

5.1. Variables correlation

We begin monitoring 16 variables in the growing file size and Apache processes experiments, including one for the experiment input, and two for the experiments output (*i.e.*, migration costs, namely time and data transferred). Variables like usage of swap and cache memories, and variables duplicated from different sources (*e.g.*, number of processes reported from inside the container and from LXD API information), were gradually discarded, as their effect on the migration costs was either negligible or redundant. By the end of the process, the following five internal variables were considered fit to be used as possible predictors for the migration costs: CPU usage, CPU load average, number of processes running, disk usage, and memory usage.

To find which internal variables are strongly correlated with the input and output for each experiment, a correlation matrix was calculated. The correlation results help to understand the dependency between each internal variable to the input from the experiments, and more importantly, the predictive association of these internal variables to the output. To assist in the visualization of the correlation results, a matrix heatmap for the correlation is presented in Fig. 11[4].

Fig. 11 a shows the result for the growing file size experiment, where the heatmap matrix shows how every considered variable, including input, *i.e.*, the file size, and output, *i.e.*, the time and data transferred for migrations, are correlated to every other variable. Strong direct or inverse correlations, *i.e.*, values close to 1 or -1, respectively, are the ones we look for when composing the predictor model. It is possible to see that "Disk Usage" was the variable with the strongest correlation to input and outputs. The correlation is expected since the file size growth mainly affects the size of the file-system, which in turn affects the stateless synchronization part in the container migration.

For the variables in the Apache processes experiment, Fig. 11b shows that "Processes" and "RAM Usage" are the strongest correlated variables to the input and outputs. The correlation between Apache summoned processes, and the total number of processes running on the system is, again, expected. Moreover, it is clear that, as the number of processes increases, so does the memory used by the system. When deriving the general-purpose equation for migration costs, it is important to evaluate how both variables (processes and memory) interact with each other, as different applications may greatly vary in processes to memory relationship. Finally, "CPU Load Average" also showed a strong correlation, although weaker than that of "Processes" and "RAM Usage". However, preliminary regression analysis showed worse results when including it in the equation, and therefore the variable was left out in the final prediction model, as described next.

Table 6
Normalization formula for the input variables.

Input Variable	Normalization Formula
Disk Usage	$\frac{x - 0.15}{1085.86}$
Ram Usage	$\frac{x - 4.23}{334.84}$
Number of Processes	$\frac{x - 5}{1001}$

5.2. Multivariable regression

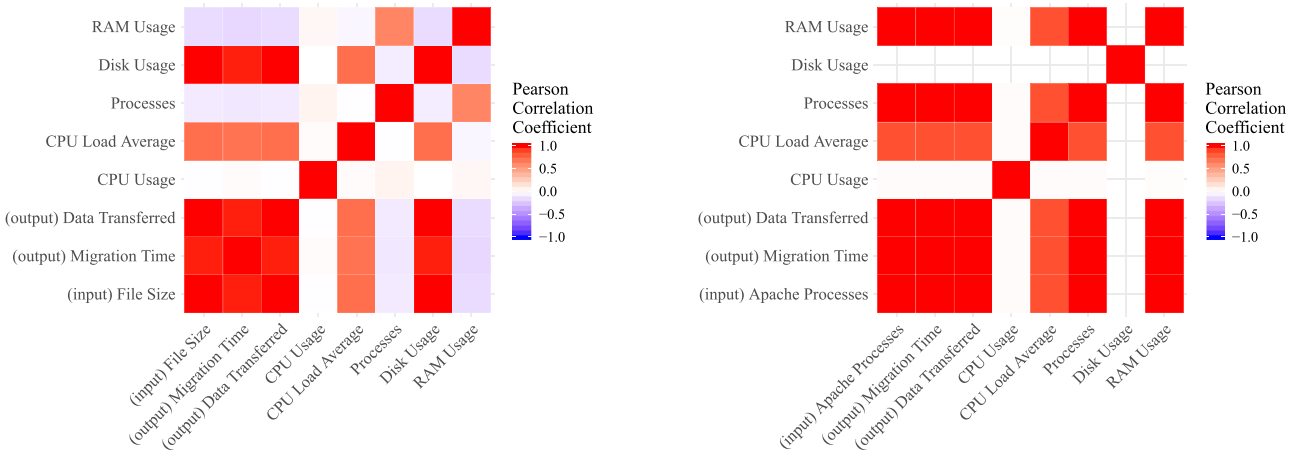
Multiple variables are now to be used in the prediction model for the migration costs prediction [61]. Therefore, we must resort to a regression model that considers more than a single input variable. We do so using the multiple variable linear regression model, which is shown in Eq. (6).

$$y = b_0 + b_1 x_1 + b_2 x_2 + \ldots + b_k x_k + \epsilon \tag{6}$$

In Eq. (6), y is the response variable, b_0 is the y-intercept, and ϵ is the error term. Each b_n term indicates the regression (or slope) coefficient for a particular variable, and thus is multiplied by the respective variable, x_n. More complex models, where interaction effects between terms are considered, were also tried. The increased complexity of the models, however, did not improve the accuracy of the predictor in our tests and were therefore discarded.

Chosen variables are orthogonal by nature, and thus to minimize the effect of the different scales we utilize normalization for feature scaling. Values are thus normalized prior to being used in the regression, therefore values for each variable are scaled to the range of [0, 1]. The normalization formula is given by Eq. (7). The result for the normalization equations for each variable is presented in Table 6. It is noteworthy that the normalization considered the extreme values encountered in the experiments, and should not be taken as absolute bounds. Extrapolation for more resource-intensive functions (*i.e.*, >1K processes, 1GB files) could

[4] For more on the heatmap matrix and correlation, see https://deeptools.readthedocs.io/en/develop/content/tools/plotCorrelation.html



(a) Growing file size experiment. (b) Apache processes experiment.

Fig. 11. Heatmap matrix for variables correlation in the two experiments.

require revision of the normalization and predictor's equations.

$$\text{Normalized}(x) = \frac{x - min(x)}{max(x) - min(x)} \tag{7}$$

We initially consider all strongly correlated variables in our prediction model. We then utilize backward elimination, iteratively removing the variable with the most significant associated p-value for as long as the value of R-squared does not decrease [62]. The best fit found for the migration time cost is presented in Eq. (8), and for the data transferred during migration, in Eq. (9).

$$\text{Migration Time} = 8.85 + 16.08 \cdot (\text{Disk Usage}) + 314.51 \cdot (\text{Processes}) \tag{8}$$

While the y-intercept is similar to what was obtained in the previous models, the slope rate is now determined by disk usage and number of processes executing. The disk and processes parameters are a direct result of the migration stateless and stateful synchronization processes, respectively. The regression coefficients show that the run-time state synchronization is potentially much more time-costly for migration than the stateless one. It is also noteworthy that the number of processes is used as a factor for the run-time state, but the memory usage is not; since both variables show a strong correlation internally, regression analysis for using both terms showed an increase in uncertainty. When considered independently, for the migration time prediction the processes variable performed better; that can be understood as the most time-consuming synchronization for the run-time state is not that of transferring the memory pages allocated by each process, but rather by iteratively synchronizing every process between hosts.

$$\begin{aligned}\text{Migration Transfer} = {} & 10.45 + 1042.06 \cdot (\text{Disk Usage}) \\ & + 241.39 \cdot (\text{RAM Usage})\end{aligned} \tag{9}$$

Similarly to the previous result, Eq. (9) shows a y-intercept consistent with what has been observed in the previous models. Disk and memory usage are the observed parameters for stateless and stateful data transfer, respectively. It is noticeable that, in contrast to the time model presented in Eq. (8), regression coefficients do not show a discrepancy between variables as large as shown in previous models, even when accounting for the normalization applied (*i.e.*, that the maximum value for disk usage is about three times the maximum for memory usage, before normalization). This result is in line with the fact that run-time state synchronization is disproportionately more costly to the migration time than it is to the migration data transferred. Finally, the representative variable for the run-time transfer is "RAM usage", in opposition to the "number of processes" used prior; while the number of processes impact on the time to synchronize run-time state between hosts, as already explained, the majority of data transferred is related to the copying of memory pages between hosts. The memory usage is, therefore, the best parameter to forecast the total amount of data being transferred during a migration. The statistical summary for both time and data transferred fits are presented in Table 7.

In this section, we described how the predictor modeling evolved from experiment-aware ones towards a more generic, experiment-independent predictor. In the next section, we present a use case of cloud computing which is used to evaluate the accuracy of the prediction model.

6. Validating the predictor model in a cloud computing scenario

In this section, we present a use case to evaluate the accuracy of our prediction model. A cloud computing scenario employing our prediction model is presented in Section 6.1. A discussion on the observed accuracy of the predictions is presented in Section 6.2.

6.1. Predictor applied to VNF-enabled cloud computing

To evaluate the accuracy of our prediction model, we propose a cloud computing use case in which a virtualized service hosted in the cloud is responsible for analyzing audio recordings from mobile users [63]. The analysis performed by the service is done using machine learning algorithms, and therefore can be memory and CPU intensive, particularly when client demand is high. An orchestrator is responsible for migrating the service along cloud virtual servers of various computing power in response to the varying demand. Virtual servers are host to cloud applications and can be realized by a bare-metal server, or by VMs and other virtualization technologies. This scenario is illustrated in Fig. 12.

In this use case, the number of connected clients varies between low (10 to 25 clients), medium (26 to 40 clients), and high (41 to 60 clients) demand. An audio file is associated with each connected client, and the size for each file is randomly chosen from 1 to 25 MB. The file transferring is abstracted from this evaluation since the predictor does not consider it. Virtual servers are realized through VMs with configurations as per Table 2, but with a less powerful one offering half the CPU cores and memory (*i.e.*, 3 cores and 8 GB of RAM). Once an upper/lower threshold for the virtual server resource usage is reached, the orchestrator consults our predictor to estimate the expected migration costs before performing a migration to a more/less resourceful virtual server,

Table 7
Summary of the multivariable regression fit for the prediction model.

Multivariable Regression	Adjusted R-squared	p-value	Residual Standard Error
Time	0.9993	<22E-17 (<2E-16 for all terms)	1.857
Data Transferred	0.9999	<22E-17 (<2E-16 for all terms)	1.909

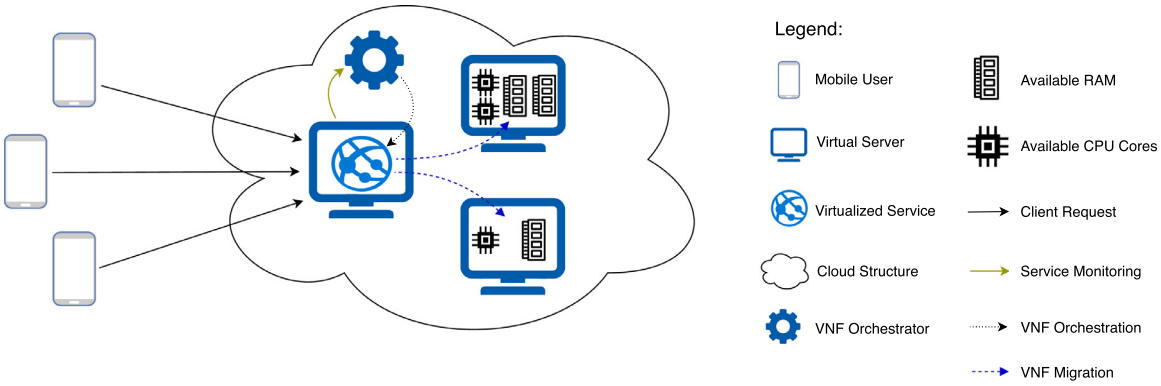
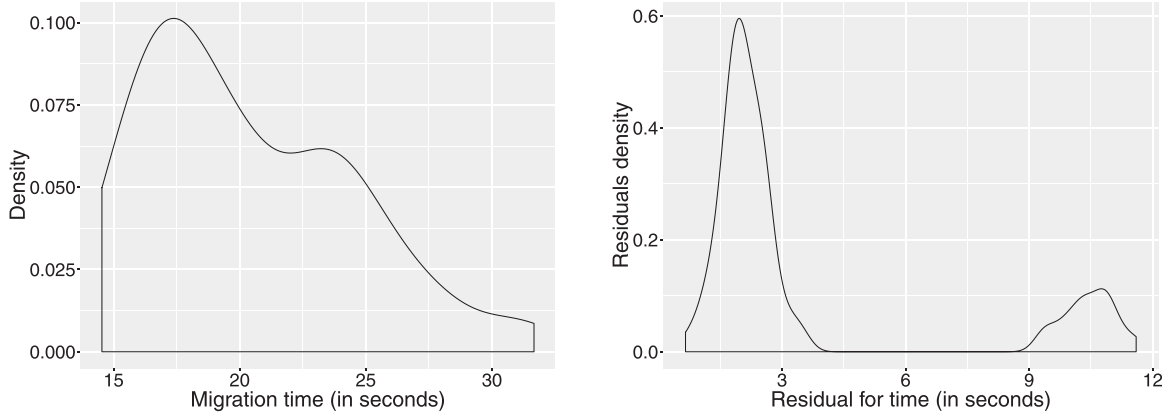


Fig. 12. Overview of the cloud experiment with VNF orchestration.



(a) Probability density of time cost.

(b) Probability density of residuals for migration time.

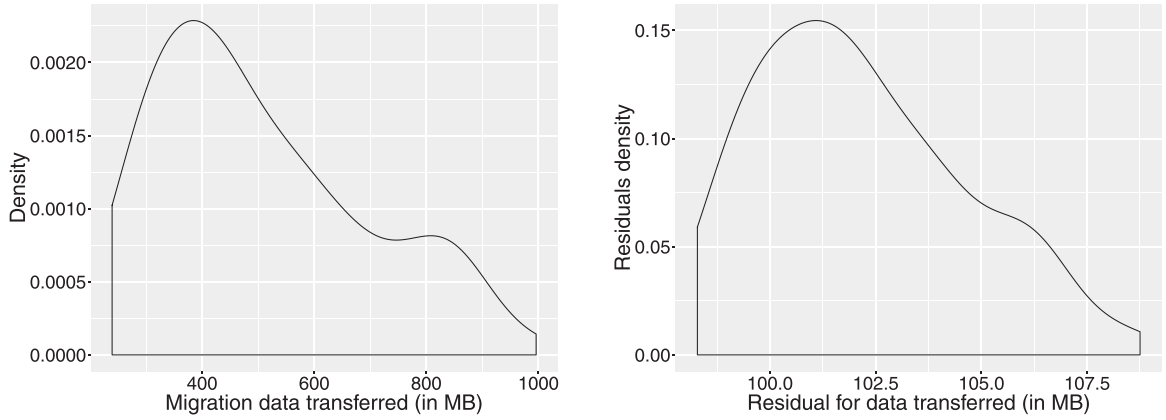
Fig. 13. Migration time cost and residuals for the use case.

accordingly. Because our intention is solely to evaluate the accuracy of the predictor, and not the orchestration algorithm, migration intentions are always performed by the orchestrator. On the same note, our analysis is focused on the results of the predictions and disregards the migrations impact on service performance, and thus clients demand is set to vary so that migrations are frequently triggered. The result for migrations prediction and measurements performed during the use case experiment is shown in the form of probability density of residuals in Fig. 13, concerning time, and in Fig. 14, for data transferred.

6.2. Discussion of results

The result for time costs are shown in Fig. 13. As shown in Fig. 13a, migrations lasted from 15 to 31 seconds, with a mean of 20.4 seconds and a standard deviation of 4.2 seconds. When considering the difference between the predicted value and the observed results, the density of residuals shown in Fig. 13b shows that over 80% of the measurements fell within a narrow 3 seconds range from the predicted values. Therefore, we argue that the results are within a reasonable range from the prediction for most applications. If the same accuracy is needed in a narrower prediction range, however, the inclusion of more parameters in the regression analysis may be required. For example, the analysis of the load in both source and destination host systems, and not only in the container, may provide additional resources to fine-tune the prediction model. Nevertheless, the density plot shows a second peak near the 10 seconds mark that indicates that some migrations were less predictable than most. Because the use case application was not part of the predictor modeling, some of its interactions with the migration tools (*e.g.*, requiring more synchronization iterations when the application is under heavy stress) may not be completely considered by the predictor. Analysis with respect to each application can assist the development of more precise predictors.

With respect to the data transferred costs, Fig. 14a shows that data transferred by migrations varied from 240 MB to 1 GB. The mean was of 506.2 MB, with a standard deviation of 188.1 MB. The density of residuals are presented in Fig. 14b, which shows the little range observed: from 98.3 MB to 108.8 MB. Despite being skewed and demonstrating that the predictor consistently under-predicts the data transferred for this application, the low variance of the residuals (6.1 MB) indicates that a simple tweak in the predictor's independent term (y-intercept) can cover most migrations cost with respect to data transferred in a narrow range. For example, considering a +100MB in all predictions



(a) Probability density of data transferred cost.

(b) Probability density of residuals for migration data transferred.

Fig. 14. Migration data transferred cost and residuals for the use case.

would represent a residual mean of 2.2 MB, with a standard deviation of 2.5 MB. Since the migration mean cost was of 506.2 MB, correctly predicting over 90% of transferring costs in a 1% range (about 5 MB) can be useful and reliable for orchestrators to determine migration plans. Again, as with the previous result (regarding time cost), the exact application effect on the migration and its costs may not always be perfectly captured by the predictor, more so because the application was not part of the modeling. The reasons to why such difference was observed in the data transferred predictions are the theme for further investigation, but an experimental-based adjustment could be used in the predictor to help the adaptation to various scenarios. For example, a feedback loop extension could gradually reduce the observed bias in the experiment by increasing or decreasing the y-intercept term in the prediction model, according to observed errors patterns.

Finally, an additional module was included in the predictor so it can be further used by an application-aware orchestrator to estimate the costs for future migrations. In this case, an orchestrator that expects a later increase/decrease in the resource usage from an application can query the predictor providing the expected variation as input, and thus work out from multiple estimations how to proceed. Furthermore, if the orchestrator can alter the service behavior itself, the conditional predictions allow a greater refinement of orchestration algorithms to include new possibilities. In the use case, for example, deciding to cache new requests for ~ 10 seconds while performing a rather inexpensive migration, and then resuming the requests after migration completes could be acceptable for the service and clients standpoint, while reducing the migration costs to a fraction of what was originally expected.

7. Conclusion and future work

In this work, we investigated the costs associated with the orchestration of VFs along the network. NFV is expected to play an important role in future networks, and a fundamental feature to be provided by these networks is the ability to dynamically reallocate VNFs. Through a systematic bibliographic review, we were able to identify what costs are expected when migrating a function between hosts. Using a well-known container virtualization platform, we performed a series of experiments to better understand the relation between several parameters and the resulting migration costs. The theoretical background for the migration implementation is used to separate the analysis in two major steps: stateless (*i.e.*, regarding the filesystem) and stateful (*i.e.*, regarding the memory pages, processes, etc.) synchronization.

Using linear regressions for the observed results in two experiments conducted, we were able to derive a prediction model for the time and data transferring costs due to a migration. The model is evaluated in a use case, in which a cloud computing setup regularly migrates a virtualized service between available hosts. Results for the predictions indicate that, under certain conditions, high accuracy is possible for the predictions of both migration costs. The predictor is extended to also offer the option for an application-aware orchestrator to forecast what are the costs expected for migration in the future, so that a well-informed decision for orchestration can be made.

Because several parameters can impact the migration costs, and it would be impractical to consider the effect of each individually, this work keeps fixed or ignores some of the parameters that can help to improve the accuracy and the adaptability of the model. For example, the network topology and links bandwidth remained constant throughout our analysis, and were therefore left out of the proposed models. Although the cost for data transferred in a migration may not be significantly affected by this variable, the cost in time for performing the migration is obviously affected by the available bandwidth, for example. Therefore, future work for improving the predictor requires the removal of fixed constraints, and the consideration for variance in each parameter is an iterate process until the model is generic and accurate enough to be used in a variety of scenarios. Additionally, the inclusion of machine-learning algorithms in the predictor can further push forward the predictor accuracy and adaptability in the future.

Declaration of Competing Interest

The authors certify that they have NO affiliations with or involvement in any organization or entity with any financial interest, or non-financial interest in the subject matter or materials discussed in this manuscript.

CRediT authorship contribution statement

Rafael de Jesus Martins: Writing - original draft, Methodology, Software, Data curation. **Cristiano Bonato Both:** Conceptualization, Writing - original draft, Visualization. **Juliano Araújo Wickboldt:** Conceptualization, Investigation, Writing - review & editing. **Lisandro Zambenedetti Granville:** Supervision, Writing - review & editing.

Acknowledgment

This research received funding from the H2020-BR programme under grant agreement no. 688941, as well as from the Brazilian Ministry of Science, Technology, Innovation, and Communication (MCTIC) through RNP, CTIC, and CNPq.

References

[1] H. Droste, et al., The METIS 5G architecture: A summary of METIS work on 5G architectures, in: Vehicular Technology Conference (VTC Spring), 2015 IEEE 81st, 2015, pp. 1–5.

[2] A. Ksentini, M. Bagaa, T. Taleb, On Using SDN in 5G: The Controller Placement Problem, in: IEEE Global Communications Conference (GLOBECOM), 2016, pp. 1–6.

[3] S. Soltesz, et al., Container-based operating system virtualization: a scalable, high-performance alternative to hypervisors, in: ACM SIGOPS Operating Systems Review, 41, 2007, pp. 275–287.

[4] W. Felter, et al., An updated performance comparison of virtual machines and linux containers, in: IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), 2015, pp. 171–172.

[5] A.M. Joy, Performance comparison between linux containers and virtual machines, in: International Conference on Advances in Computer Engineering and Applications (ICACEA), 2015, pp. 342–346.

[6] A. Celesti, et al., Exploring Container Virtualization in IoT Clouds, in: IEEE International Conference on Smart Computing (SMARTCOMP), 2016, pp. 1–6.

[7] M.G. Xavier, et al., Performance evaluation of container-based virtualization for high performance computing environments, in: 21st Euromicro International Conference on Parallel, Distributed, and Network-Based Processing, 2013, pp. 233–240.

[8] R. Cziva, D.P. Pezaros, Container network functions: bringing NFV to the network edge, IEEE Commun. Mag. 55 (6) (2017) 24–31.

[9] M.C. Luizelli, et al., Piecing together the NFV provisioning puzzle: Efficient placement and chaining of virtual network functions, in: IFIP/IEEE International Symposium on Integrated Network Management (IM), 2015, pp. 98–106.

[10] J.G. Herrera, J.F. Botero, Resource allocation in NFV: A Comprehensive survey, IEEE Trans. Netw. Serv. Manage. 13 (3) (2016) 518–532.

[11] M.C. Luizelli, et al., A fix-and-optimize approach for efficient and large scale virtual network function placement and chaining, Comput. Commun. 102 (2017) 67–77.

[12] A.G. Dalla-Costa, et al., Maestro: An NFV Orchestrator for Wireless Environments Aware of VNF Internal Compositions, in: IEEE 31st International Conference on Advanced Information Networking and Applications (AINA), 2017, pp. 484–491.

[13] R. Cohen, et al., Near optimal placement of virtual network functions, in: IEEE Conference on Computer Communications (INFOCOM), 2015, pp. 1346–1354.

[14] A.S. Rajan, et al., Understanding the bottlenecks in virtualizing cellular core network functions, in: The 21st IEEE International Workshop on Local and Metropolitan Area Networks, 2015, pp. 1–6.

[15] B. Yi, et al., A comprehensive survey of network function virtualization, Comput. Networks 133 (2018) 212–262.

[16] H. Ibn-Khedher, et al., Network issues in virtual machine migration, in: International Symposium on Networks, Computers and Communications (ISNCC), 2015, pp. 1–6.

[17] R. Riggio, T. Rasheed, R. Narayanan, Virtual network functions orchestration in enterprise WLANs, in: IFIP/IEEE International Symposium on Integrated Network Management (IM), 2015, pp. 1220–1225.

[18] X. Li, C. Qian, An NFV orchestration framework for interference-free policy enforcement, in: IEEE 36th International Conference on Distributed Computing Systems (ICDCS), 2016, pp. 649–658.

[19] H. Ibn-Khedher, et al., Optimal and Cost Efficient Algorithm for Virtual CDN Orchestration, in: IEEE 42nd Conference on Local Computer Networks (LCN), 2017, pp. 61–69.

[20] G. Einziger, M. Goldstein, Y. Sa'ar, Faster placement of virtual machines through adaptive caching, in: IEEE Conference on Computer Communications (INFOCOM), 2019, pp. 2458–2466.

[21] T. Ouyang, et al., Adaptive user-managed service placement for mobile edge computing: An online learning approach, in: IEEE Conference on Computer Communications (INFOCOM), 2019, pp. 1468–1476.

[22] V. Farhadi, et al., Service placement and request scheduling for data-intensive applications in edge clouds, in: IEEE Conference on Computer Communications (INFOCOM), 2019, pp. 1279–1287.

[23] F. Bari, et al., Orchestrating virtualized network functions, IEEE Trans. Netw. Serv. Manage. 13 (4) (2016) 725–739.

[24] Q. Sun, et al., Forecast-Assisted NFV Service Chain Deployment Based on Affiliation-Aware vNF Placement, in: IEEE Global Communications Conference (GLOBECOM), 2016, pp. 1–6.

[25] G. Dhiman, et al., A system for online power prediction in virtualized environments using gaussian mixture models, in: Proceedings of the 47th Design Automation Conference, 2010, pp. 807–812.

[26] Z. Zheng, et al., Octans: Optimal placement of service function chains in many–core systems, in: IEEE Conference on Computer Communications (INFOCOM), 2019, pp. 307–315.

[27] B. Andrus, et al., Live Migration Downtime Analysis of a VNF Guest for a Proposed Optical FMC Network Architecture, in: Proceedings of Photonic Networks; ITG-Symposium, 2016, pp. 1–5.

[28] W. Cerroni, F. Callegati, Live migration of virtual network functions in cloud-based edge networks, in: IEEE International Conference on Communications (ICC), 2014, pp. 2963–2968.

[29] H. Liu, et al., Performance and energy modeling for live migration of virtual machines, in: Proceedings of the 20th international symposium on High performance distributed computing, 2011, pp. 171–182.

[30] F. Tao, et al., BGM-BLA: A New Algorithm for dynamic migration of virtual machines in cloud computing, IEEE Trans. Serv. Comput. 9 (6) (2016) 910–925.

[31] J. Xia, et al., Reasonably Migrating Virtual Machine in NFV-Featured Networks, in: IEEE International Conference on Computer and Information Technology (CIT), 2016, pp. 361–366.

[32] C.C. Lin, et al., A practical model for analyzing push-based virtual machine live migration, in: 7th International Conference on Cloud Computing and Big Data (CCBD), 2016, pp. 347–352.

[33] J. Liu, et al., Migration-based dynamic and practical virtual streaming agent placement for mobile adaptive live streaming, IEEE Trans. Netw. Serv. Manage. (2017).

[34] Z. Tang, et al., Migration modeling and learning algorithms for containers in fog computing, IEEE Trans. Serv. Comput. 12 (5) (2019) 712–725.

[35] A. Vakali, G. Pallis, Content delivery networks: status and trends, IEEE Internet Comput. 7 (6) (2003) 68–74.

[36] N. Herbaut, et al., Dynamic deployment and optimization of virtual content delivery networks, IEEE Multimedia 24 (3) (2017) 28–37.

[37] H. Ibn-Khedher, et al., Scalable and Cost Efficient Algorithms for Virtual CDN Migration, in: IEEE 41st Conference on Local Computer Networks (LCN), 2016, pp. 112–120.

[38] H. Ibn-Khedher, et al., OPAC: An optimal placement algorithm for virtual CDN, Comput. Networks 120 (2017) 12–27.

[39] N.T. Jahromi, S. Kianpisheh, R.H. Glitho, Online VNF Placement and Chaining for Value-added Services in Content Delivery Networks, in: IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN), 2018, pp. 19–24.

[40] S. Sezer, et al., Are we ready for SDN? implementation challenges for software-defined networks, IEEE Commun. Mag. 51 (7) (2013) 36–43.

[41] H. Shariatmadari, et al., Machine-type communications: current status and future perspectives toward 5G systems, IEEE Commun. Mag. 53 (9) (2015) 10–17.

[42] C. Wang, et al., A switch migration-Based decision-Making scheme for balancing load in SDN, IEEE Access 5 (2017) 4537–4544.

[43] M. Moradi, et al., Softbox: A Customizable, low-Latency, and scalable 5G core network architecture, IEEE J. Sel. Areas Commun. 36 (3) (2018) 438–456.

[44] H. Jin, et al., NFV And SFC: A Case study of optimization for virtual mobility management, IEEE J. Sel. Areas Commun. 36 (10) (2018) 2318–2332.

[45] F. Le, E. Nahum, Experiences Implementing Live VM Migration over the WAN with Multi-Path TCP, in: IEEE Conference on Computer Communications (INFOCOM), 2019, pp. 1090–1098.

[46] P.-C. Lin, C.-F. Wu, P.-H. Shih, Optimal Placement of Network Security Monitoring Functions in NFV-Enabled Data Centers, in: IEEE 7th International Symposium on Cloud and Service Computing (SC2), 2018-January, 2018, pp. 9–16.

[47] D. Fernando, et al., Live Migration Ate My VM: Recovering a Virtual Machine after Failure of Post-Copy Live Migration, in: IEEE Conference on Computer Communications (INFOCOM), 2019, pp. 343–351.

[48] B. Shi, H. Shen, Memory/Disk Operation Aware Lightweight VM Live Migration Across Data-centers with Low Performance Impact, in: IEEE Conference on Computer Communications (INFOCOM), 2019, pp. 334–342.

[49] J. Zhang, et al., Joint Optimization of Virtual Function Migration and Rule Update in Software Defined NFV Networks, in: IEEE Global Communications Conference, 2018-January, 2018, pp. 1–5.

[50] X. Zhou, et al., Approach for minimising network effect of VNF migration, IET Commun. 12 (20) (2018) 2574–2581.

[51] J. Xia, Z. Cai, M. Xu, Optimized Virtual Network Functions Migration for NFV, in: IEEE 22nd International Conference on Parallel and Distributed Systems (ICPADS), 2016, pp. 340–346.

[52] B. Lantz, B. Heller, N. McKeown, A network in a laptop: Rapid prototyping for software-defined networks, in: Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks, 2010, pp. 1–6.

[53] R. Bradford, et al., Live wide-area migration of virtual machines including local persistent state, in: Proceedings of the 3rd international conference on Virtual execution environments, 2007, pp. 169–179.

[54] A. Gember-Jacobson, et al., Opennf: enabling innovation in network function control, SIGCOMM Comput. Commun. Rev. 44 (4) (2014) 163–174.

[55] CRIU, Checkpoint/restore in userspace, 2019, (https://criu.org/Main_Page). [Online; accessed 6-June-2019].

[56] Docker, Enterprise container platform for high-velocity innovation, 2019, (https://www.docker.com/), [Online; accessed 6-June-2019].

[57] R. Rosen, Resource management: linux kernel namespaces and cgroups, Haifux, May 186 (2013).

[58] P. Enberg, A performance evaluation of hypervisor, unikernel, and container network I/O virtualization, MS thesis, Faculty Sci., Univ. Helsinki, Helsinki, Finland, 2016 Ph.D. thesis.

[59] R. Jain, The art of computer systems performance analysis: Techniques for experimental design, measurement, simulation, and modeling, John Wiley & Sons, 1990.

[60] Apache Software Foundation, Apache http server, 2018.

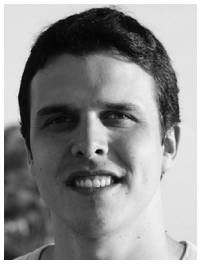
[61] P. Teetor, R cookbook: Proven recipes for data analysis, statistics, and graphics, O'Reilly Media, Inc., 2011.

[62] L. Skovgaard, Applied regression analysis. 3rd edn. n. r. draper and h. smith, wiley, new york, 1998. no. of pages: xvii + 706. price: £45. isbn 0-471-17082-8, Stat. Med. 19 (22) (2000) 3136–3139, doi:10.1002/1097-0258(20001130)19:22<3136::AID-SIM607>3.0.CO;2-Q.

[63] C.F. Bublitz, et al., Unsupervised segmentation and classification of snoring events for mobile health, in: IEEE Global Communications Conference (GLOBECOM), 2017, pp. 1–6.

Rafael de Jesus Martins received the B.Sc. degree in Computer Engineering from Federal University of Rio Grande do Sul (UFRGS), in Brazil, where he is currently pursuing the M.Sc. degree. His research interests include network virtualization and cloud management.

Cristiano Bonato Both is an associate professor of the Applied Computing Graduate Program at the University of Vale do Rio dos Sinos (UNISINOS), Brazil. He coordinators research projects funded by H2020 EU-Brazil, CNPq, FAPERGS, and RNP. His research focuses on wireless networks, next generation networks, softwarization and virtualization technologies for telecommunication network, and SDN-like solutions for the Internet of Things. He is participating in several Technical Programme and Organizing Committees for different worldwide conferences and congresses.



Juliano Araujo Wickboldt is an associate professor at the Institute of Informatics of the Federal University of Rio Grande do Sul (UFRGS), in Brazil. He holds a Ph.D. degree from the Federal University of Rio Grande do Sul (UFRGS), in Brazil. He achieved his M.Sc. degree from UFRGS in a joint project with HP Labs Bristol and Palo Alto. He also received a B.Sc. degree in computer science at the Pontifical Catholic University of Rio Grande do Sul in 2006. Juliano was an intern at NEC Labs Europe in Heidelberg, Germany from 2011 to 2012. His current research interests include cloud resource management and software-defined networking.

Lisandro Zambenedetti Granville is a full professor at the Institute of Informatics of the Federal University of Rio Grande do Sul (UFRGS), in Brazil. He received his M.Sc. and Ph.D. degrees, both in Computer Science from UFRGS, in 1998 and 2001, respectively. He is a member of the Brazilian Computer Society (SBC). He has served as a TPC member for many events in the area of network management (e.g., IM, NOMS, and CNSM) and was TPC co-chair of DSOM 2007 and NOMS 2010. management.