UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE ENGENHARIA DE COMPUTAÇÃO

RAFAEL DE JESUS MARTINS

Virtual Functions Orchestration Costs: from Identification to Prediction

Work presented in partial fulfillment
of the requirements for the degree of
Bachelor in Computer Engineering

Advisor: Prof. Dr. Lisandro Zambenedetti Granville
Coadvisor: Prof. Dr. Juliano Araújo Wickboldt

Porto Alegre
December 2018

ACKNOWLEDGEMENTS

First and foremost, I would like to thank my advisors, professors Lisandro Granville, Juliano Wickboldt, and Cristiano Both, for all the guidance they offered me not only in this work, but throughout last years. I would also like to extend my gratitude to my first advisor, from when I first embarked the joys (and pains) from the academic world, professor Alberto Schaeffer-Filho.

Also, this work could not be possible without the persevering help from my labs colleagues, who were always open to discussion and insightful not only in technical aspects, but also in every matter known to man. A non-exhaustive list includes Anderson Santos, Andrei Rodrigues, Augusto Bardini, Iulisloi Zacarias, Lucas Bondan, and Lucas Castanheira. Past colleagues, from networks labs and graduation course, are too many to mention, but also deserving of likewise praise.

I would especially like to thank my parents, Oscar Morency Otto Martins and Paula de Jesus Martins, and also my sister, Luísa de Jesus Martins, and my grandparents, Nair and Paulo de Jesus, for being the entire cornerstone upon which I can build all my dreams. Hopefully, you already know it full well.

Last, but not least, I would like to thank my girlfriend Caroline Grazioso, who is always by my side and gives me all the emotional support I could ever need. This is not the last time I will thank you.

ABSTRACT

The advent of function virtualization concept, especially that of network functions (Network Function Virtualization, or NFV), presents critical benefits for networks of the future. Offering scalable solutions for network requirements that are dynamic by nature, virtualized functions can be expanded or withdrawn along the network infrastructure according to instantaneous demands. Although the orchestration of virtualized functions present gains for network operators and clients alike, the overhead for moving functions has not been thoroughly explored so far, especially considering functions virtualized by newer container technologies. In this work, we present the most relevant bibliographic references found in the subject, considering the differences in available virtualization technologies, and possible network setups. To assert the associated cost posed by the orchestration of virtualized functions, we utilize a well-known container platform to perform a series of experiment in a controlled environment. Quantitative analysis of the results are then regressed to a mathematical relation for the prediction of time and data transferred associated with the orchestration of virtual functions. Finally, a use case is presented, in which the predictor estimations are validated against observed measurements. Obtained results indicate that predictions can be accurate within reasonable range, and therefore future orchestration algorithms may benefit from considering such predictions when defining the best orchestration plan available.

Keywords: Container Virtualization. Virtual Function Orchestration. NFV. Performance Analysis.

Custos para orquestração de funções virtualizadas: da identificação à predição

RESUMO

O surgimento do conceito de virtualização de funções, em especial as de funções de rede (NFV, do inglês *Network Function Virtualization*), traz benefícios críticos para as redes do futuro. Oferecendo soluções escaláveis para requisitos naturalmente dinâmicos da rede, as funções virtualizadas podem ser expandidas ou retraídas ao longo da estrutura de rede de acordo com demandas instantâneas. Embora esta orquestração de funções represente ganhos tanto para os operadores quanto para os clientes da rede, o próprio custo da movimentação das funções foi pouco explorado até o momento, especialmente considerando-se a relativamente nova virtualização de funções por *containers*. Neste artigo, são apresentados os principais trabalhos encontrados sobre o tema, considerando as diferenças presentes entre tecnologias de virtualização disponíveis, e entre estruturas de rede possíveis. Para afirmar o custo imposto pela orquestração de funções virtualizadas, uma série de experimentos são executados em um ambiente controlado, utilizando uma plataforma de virtualização por *container* bem conhecida. A análise quantitativa dos resultados é então reduzida para uma relação matemática, que auxilia na predição dos custos de tempo e banda associados à orquestração de funções virtualizadas. Finalmente, um estudo de caso é apresentado, no qual as estimativas fornecidas pelo preditor são validadas com os resultados observados. Os resultados obtidos indicam que as predições podem ser precisas dentro de uma faixa razoável, e que portanto futuros algoritmos de orquestração podem beneficiar-se ao considerar tais predições quando estiverem definindo o melhor plano de orquestração disponível.

Palavras-chave: Virtualização por *Container*, Orquestração de Função Virtualizada, FRV, Análise de Desempenho.

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS AND ACRONYMS

ARP	Address Resolution Protocol
App	Application
Btrfs	B-tree file system
CDN	Content Delivery Network
COPA	Container Orchestration and Provisioning Architecture
CPU	Central Processing Unit
CRIU	Checkpoint/Restore In Userspace
FRV	Função de Rede Virtualizada
FUTEBOL	Federated Union of Telecommunications Research Facilities for an EU-Brazil Open Laboratory
GTP	General Packet Radio Service Tunneling Protocol
HPC	High-Performance Computing
IoT	Internet of Things
IP	Internet Protocol
LXC	Linux Containers
LXD	Linux Containers Daemon
MAE	Mean Absolute Error
NFV	Network Function Virtualization
OS	Operating System
QoE	Quality of Experiment
QoS	Quality of Service
RAM	Random-access Memory
Rsync	Remote Synchronization
SDN	Software-Defined Network
SC	Service Chaining

SGW	Serving Gateway
SSE	Sum of Squared Errors
TCP	Transmission Control Protocol
UFRGS	Federal University of Rio Grande do Sul
VF	Virtual Function
VM	Virtual Machine
VNF	Virtual Network Function
VNFC	VNF Component
ZFS	Zettabyte File System

CONTENTS

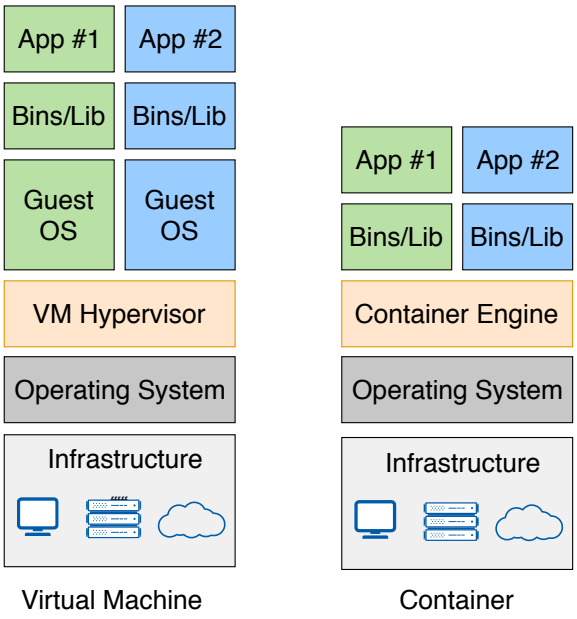
1 INTRODUCTION

Future networks, as exemplified by the forthcoming 5G network and its services for Extreme Mobile Broadband (xMBB) and Massive MTC (mMTC) [Droste et al. 2015], are being built upon technologies such as Software-Defined Networking (SDN) and network virtualization concepts such as Network Functions Virtualization (NFV) [Ksentini, Bagaa and Taleb 2016]. Specifically in the case of NFV, network applications that would traditionally be run only by specialized equipment, appliances known as middle-boxes, can also be executed by software running on top of general purpose hardware. Therefore, possibilities offered to network management are massively expanded, since the deployment of a given network function (*e.g.*, a firewall) does not depend on the acquisition and physical installation of a specialized hardware; instead, not only the Virtual Network Function (VNF) can be easily deployed along the network, computational resources for the VNF can be scaled in and out, for example, according to the perceived network load in any given moment.

Resource virtualization offers users a series of benefits, such as hardware independence, easiness of deployment, and scalability of used resources. Traditionally performed by Virtual Machines (VM), newer container virtualization technologies emerge with the promise of lighter, more efficient application virtualization [Soltesz et al. 2007]. An overview of the architecture of both systems is shown in Figure 1.1. Several studies have compared the effectiveness, particularly regarding performance and scalability, of virtualization by VMs and by modern container virtualization technologies [Felter et al. 2015, Joy 2015]. By sharing the Operating System (OS), and notably its kernel, with the underlying host, container technologies reduce much of the virtualization overhead making container virtualization a feasible option for network environments from IoT Cloud/-Fog [Celesti et al. 2016] to High-Performance Computing (HPC) [Xavier et al. 2013].

In the context of NFV, studies have been carried out to try to solve problems for VNF composition, chaining, and placement. Because VNFs can be composed of smaller, reusable components, known as VNF Components (VNFCs), a single VNF can be composed in multiple ways [Herrera and Botero 2016]. In addition, 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 in commodity hardware, NFV-enabled networks tend to offer multiple host options for each VNF or VNFCs; selecting the best host to run each function is an open research problem, usually

Figure 1.1: Architectural comparison of virtualization technologies.



Source: Author

referred to as the placement problem. Previous studies have investigated the problem for VNF composition [Dalla-Costa et al. 2017], chaining [Luizelli et al. 2017], and placement [Cohen et al. 2015] through a theoretical-based approach. To this end, our work focuses on experimental-based research. We argue that, even if theoretical research is able to produce increasingly optimized solutions, hardware and software constraints must be taken into account when adopting such solutions in the real world.

VNF orchestration, *i.e.*, the decision-making to initialize, stop, or move a VNF from one host to another, plays an important role in optimizing the available network resources. The decision of which function to orchestrate, and in which way, has been subject for several studies [Riggio, Rasheed and Narayanan 2015, Li and Qian 2016, Khedher et al. 2017], each offering a different take on the matter at hand. However, orchestration algorithms consistently fail to consider the overhead introduced by the orchestration itself when determining the best available orchestration options. For example, when applying a load-balancing algorithm, the network transfer required to migrate a function between two hosts may run on top of an already congested link, reducing the effectiveness of performing such migration. In order to support the decision-making process, an orchestrator should thus consider what will be the expected cost for implementing such actions.

In the scenario of VNF orchestration, our work as collaborators of the FUTEBOL project [FUTEBOL 2018] focused on the experimental analysis of orchestration solutions. FUTEBOL project is a Europe-Brazil partnership to develop and deploy research infras-

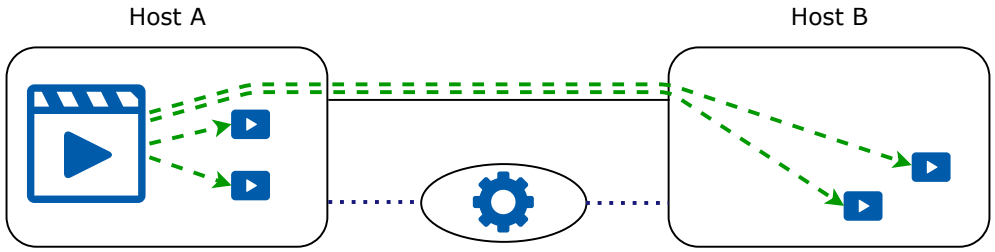
tructure, particularly in the convergence point between optical and wireless networks. Three use cases with five experiments are being developed in the project since its inception to showcase how researchers could benefit from experimenting in the testbeds. In this context, we developed studies in an experiment titled "Heterogeneous network management with SDN and virtualization", precisely in the NFV and service orchestration part. Function placement and Service Chaining problems were among the research questions proposed to be studied by the experiment. An overview of one of the experiments proposed in our studies can be seen in Figure 1.2. An early version of the experiment deployed the VNF (in this case, the video server) in a VM. The migration occurred seamlessly for the video clients; yet, the introduced load on the network from issuing a VM migration could not be ignored: as shown in Fig 1.2c, in order to ease the traffic congestion from Host A to Host B, the VNF must be migrated through the same network link, further congesting it until migration is completed. In a later improved version, container virtualization was used in the experiment, in order to minimize the virtualization and migration overhead. Still, the orchestration algorithm developed for the experiment was naive on the migration costs, and the question of how costly can a migration be remained an open one.

In this work, we describe how understanding the costs associated with VNF migration, posed to us during our research in FUTEBOL, was approached. First and foremost, to better understand the associated costs, a systematic review of the relevant peer-reviewed literature was conducted in order to confirm that the issue was one of interest in the area, and to identify which metrics were most commonly used. We then chose a well-known container virtualization platform, namely, LXD[1], to carry out our experimental studies, as we observed the impact of a number of variables in the result for migration costs. Combining the theoretical background with the observed experimental results, we derive a mathematical model to predict the costs associated with VNF migration. To verify the accuracy of our model, a use case is presented, in which our proposed model indicates to provide accurate estimations under certain conditions. Based on our results, it is thus indicated that future NFV orchestration algorithms can benefit from including some prediction model feature, as a method to better weigh in available orchestration alternatives.

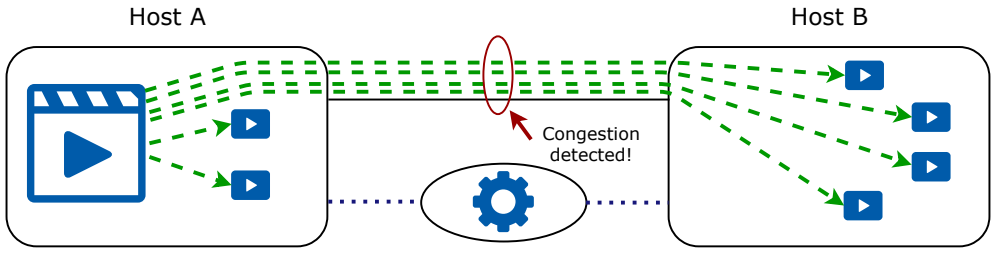
The remainder of this document is structured as follows. In Chapter 2, we present the related work on the Virtual Function orchestration problem. In Chapter 3, we present the experiments conducted to better understand the orchestration-cost relationship. In

[1] https://linuxcontainers.org/

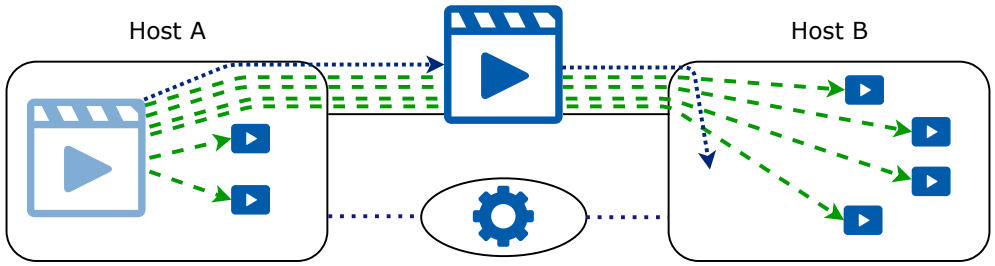
Figure 1.2: Orchestration experiment conducted as part of FUTEBOL research.



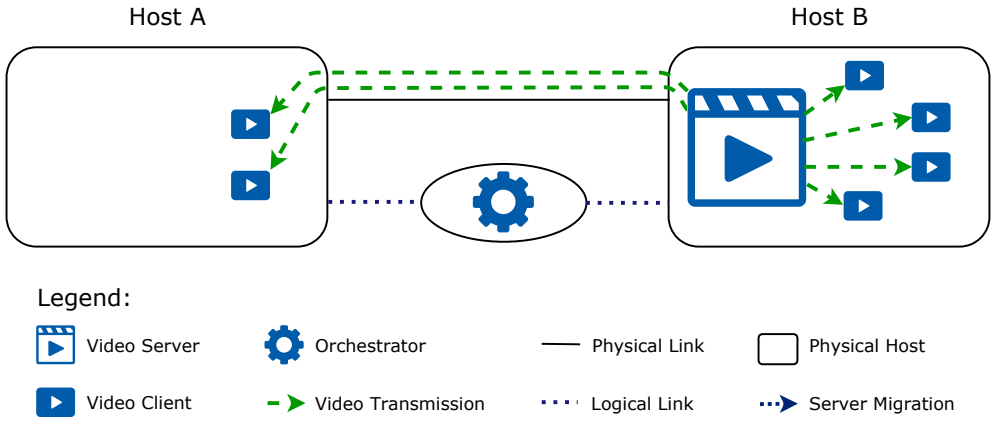
(a) A video server placed in Host A services a small number of video clients in Hosts A and B.



(b) As new clients are initialized in Host B, traffic between hosts increases, congesting the physical link.



(c) The service orchestrator responds to the increased demand in Host B, by triggering a video server migration.



(d) Server is migrated to Host B, mitigating the physical link congestion.

Source: Author, adapted from FUTEBOL deliverable

Chapter 4, a quantitative analysis of the experiments' results is discussed, and we introduce a mathematical model to predict migration costs from an orchestration plan. In Chapter 5, we present an use case where our prediction model is used and compared to observed results. In Chapter 6, we present our conclusion to this document and the future work.

2 RELATED WORK

In this Chapter, we present the methodology used to conduct a literature review on the subject of interest. The relevant results of our research are laid out concerning the network area of each research. The importance of our work in comparison to the ones presented is also underlined. Finally, the take away from the bibliography to our work is highlighted by the end of this chapter.

2.1 Systematic Literature Review

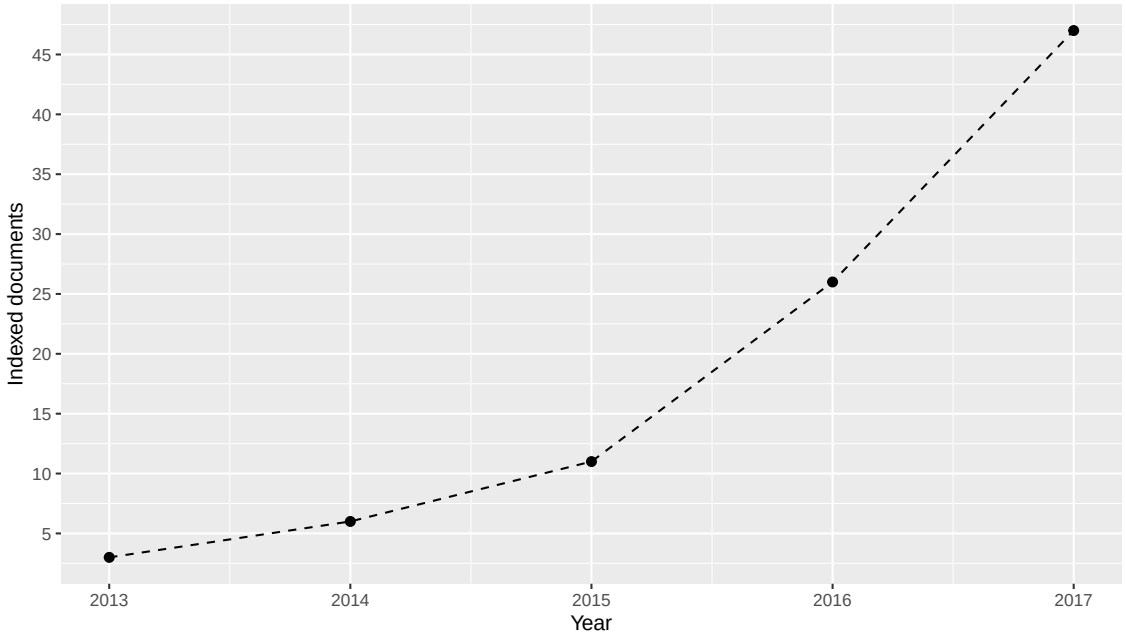
To start this work, a systematic literature review was performed. The objective of this review is twofold: (I) to confirm that the orchestration cost is a relevant topic in the field; and (II) having confirmed the relevance of the problem, identify what metrics are used to determine these costs. To do so, we chose to utilize Scopus[1] research tool and database of peer-reviewed literature. Being one of the largest available database for academic research, Scopus has the added benefit of providing curatorship for indexed documents, which thus eliminates most of the scientifically irrelevant results. After some refining, the following search query was made:

TITLE-ABS-KEY (((nfv OR vnf OR container OR "function virtualization" OR "virtualized function") AND ((migration OR orchestration OR deployment OR placement) W/3 (cost OR penalty OR tradeoff OR risks)) ANDNOT (ship OR cargo OR sea OR disease OR patient OR food))) AND (LIMIT-TO (PUBYEAR , 2018) OR LIMIT-TO (PUBYEAR , 2017) OR LIMIT-TO (PUBYEAR , 2016) OR LIMIT-TO (PUBYEAR , 2015) OR LIMIT-TO (PUBYEAR , 2014) OR LIMIT-TO (PUBYEAR , 2013))

The key aspect of the query is to filter recent documents that explicitly cites costs associated with the orchestration of virtual functions. A time window of five years was used since the technologies we consider in this work, such as container virtualization, were not prominent prior to the considered years. On January 26, 2018, this query resulted in a total of 94 documents. All the indexed documents were passed on to a sheet[2], where each document was systematically categorized [Kitchenham 2004]. The number of indexed documents by year is shown in Figure 2.1 to exhibit the evolution of interest in the area. Three filters were used in sequence to achieve the categorization of all documents:

[1] <https://www.scopus.com>
[2] <https://goo.gl/wsMiZq>

Figure 2.1: Query result: number of indexed documents by year.



Source: Author; 2018 is excluded from visualization for being a partial year.

1. In the first filter, the title and abstract of all results were read; results found to be clearly out of scope in this filter are then marked as such, and the remainder results pass on to the second filter. From the 94 query results, a total of 60 studies passed the first filter.
2. In the second filter, the sections for Introduction and Conclusion of the remaining documents were considered; once again, documents considered in the overall scope were passed on to the third filter, and the remaining results were discarded from consideration. From the 60 studies that passed on the first filter, 25 also passed this second filter.
3. Finally, the third filter induces the complete read of the article. Documents that go through the third filter are grouped by relevance to our research problem, categorized according to networking and virtualization aspects, and, when deemed necessary, summarized for easier recapitulation afterward. A total of 12 studies passed all three filters.

In the following sections, the articles that passed all three filters are discussed. The studies are grouped in sections according to their network environment. The studies on the network environment most prominent found in the filtered results, that is, Cloud Computing, is discussed on the next section.

2.2 Cloud Computing

Among the most important scenarios for virtual function orchestration, we can cite Cloud Computing. In the Cloud Computing area, there is great interest by the Cloud operator to meet the services agreement with their clients, while minimizing the operating costs of the Cloud. To do so, it is useful, for example, to deploy an orchestrator in the Cloud, which migrates virtual functions along the Cloud. An orchestrator thus allows the more efficient use of Cloud's resources, by making use of the Cloud scalability in response to demand fluctuations.

In the context of Cloud Computing, the work by Cerroni *et al.* [Cerroni and Callegati 2014] focus on the performance analysis of live migrating VNFs. With live migration, the running state of the function is transferred from the source host to the destination host before migration is finished. As a result, function downtime noticed by the user is drastically reduced, even to negligible levels [Liu et al. 2011]. In their work, Cerroni *et al.* considered as performance indicators the total time elapsed in a migration, and the service's downtime. It is worth noting that, in the same way as most other researches considered do, the work uses virtualization through virtual machines to deploy their VNFs. As previously mentioned, the reduced overhead for using container virtualization can positively impact the effectiveness of virtual functions, when compared to their virtual machines counterpart [Felter et al. 2015]. Since we are interested in determining the orchestration impact for some of the lighter virtualization available, namely the container one, the quantitative performance analysis for container orchestration lies as a central research question.

In the same line of work, Tao *et al.* [Tao et al. 2016] tackled the problem of dynamically migrating VMs in Cloud Computing. The authors introduce a proposal to optimize migration plans, *i.e.*, the intention of performing a series of migrations which are compared regarding their performance. Once again, total time elapsed for migration is taken into account when establishing the performance for each plan. The addition of the data transferred by VMs migration along the network compose the performance computation.

A different approach was proposed by Lin *et al.* [Lin et al. 2016] in the performance analysis of migrating VMs in the Cloud. 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 to be synchronized between hosts as a determining factor for

the performance evaluation. The explanation is that, since the VNF migration relies on memory synchronization from origin to destination, VNFs with different memory dirtying rates may require more or less iterations for synchronization, depending on migration optimization used. Here, we understand as a dirty memory page a page whose content is not consistent from source to destination. Therefore, VNFs with patterns for higher memory dirtying rates result in more iterations of the memory synchronization loop in the migration algorithm, which can be of utmost importance when considering migration for applications with such patterns.

Lastly, the work by Liu *et al.* [Liu et al. 2017] focused on utilizing Cloud technology to assist live streaming in mobile networks. In their scenario, migrations along the network are used 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 video service along the network, in response to users demand. Due to the application's nature, timing and bandwidth requirements are of vital importance in this scenario. That is, the maximum acceptable delay and the minimum bandwidth available must be ensured (in reasonable costs) 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 migration.

The relation between Cloud Computing and NFV can be straightforwardly seen, but the same can not be said for every other network environment. While Cloud Computing offers remote computational resources, network paradigms that focus on the replication and reallocation of any information along the network can benefit from NFV concepts. In the next section, we delve into studies on Content Delivery Networks (CDNs).

2.3 Content Delivery Network

Content Delivery Networks improve applications and services by distributing available content (*e.g.*, multimedia files) along the network, in response to user demand [Vakali and Pallis 2003]. A step further is taken by virtual CDNs, where servers and networks themselves are virtualized, using SDN and NFV paradigms, to further improve services scalability [Herbaut et al. 2017]. By improving the management of resources, the employment of NFV and SDN by virtual CDN can positively impact users' QoE, without burdening the network operating cost. In this topic, the work we developed in FUTE-

BOL, as presented in Figure 1.2, can be viewed as a simplified virtual CDN (simplified because traditional CDN methods, such as replication of content, was not used in the experiment).

In a series of studies [Ibn-Khedher et al. 2016, Ibn-Khedher et al. 2017, Khedher et al. 2017], Ibn-Khedher *et al.* investigated the problem of optimizing orchestration algorithms for CDNs. The authors propose cost-efficient algorithms for the placement and orchestration problem of virtual nodes in a CDN. When running their experiments, the researchers used VMs to virtualize network's content. The authors considered as main costs of orchestration the total elapsed time, and the downtime of the migrated service.

The benefits offered by the employment of SDN and NFV by CDNs are a direct result for maximizing the efficiency in reallocating and replicating content over the CDN hosts. By combining both paradigms, a network service that relies on permanent monitoring and decision-making of available content and users' patterns can minimize its operating costs, while improving customers satisfaction. Filtered studies for other network environments that can also benefit from SDN and NFV, individually or in conjunction, are discussed in the next section.

2.4 SDN and Other Network Paradigms

In Software-Defined Networks, a centralized controller with the view of the programmable network is used by network carriers to improve network scalability, Quality of Service (QoS), and service management [Sezer et al. 2013]. 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.

Using SDN in a 5G scenario, Ksentini *et al.* [Ksentini, Bagaa and Taleb 2016] looked into costs associated with Serving Gateway (SGW) reallocation. SGW in a 5G network is responsible for forwarding users' data traffic [Shariatmadari et al. 2015], making the SGW placement problem an important one in the area. In their work, multiple instances of VNF 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 signalling 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.* [Wang et al. 2017] using a more generic network with SDN support, in deploying a load-balance algorithm in the network.

Three other investigations tackled the problem for VNF migration in generic networks. The study by Sun *et al.* [Sun et al. 2016] investigates the problem with NFV Service Chaining (SC). Service Chaining 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 pertaining to a 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 in demand for VNFs in an SC, thus reducing the need for subsequent migrations in response to demand increase. The authors present results from simulations to compare the performance for different proposals.

Based on the related work presented in the present and previous sections, a summary is laid out in the next section. We also highlight how the results affect our research.

2.5 Influence in Our Research

The main aspects of the Related Work are summarized in Table 2.1. 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, as indicated in Figure 2.1, so investigations tend to be mostly theoretical, resorting to simulation experiment to analyze their proposals. When using a real NFV-enabled network, researchers preferred method for virtualization was using Virtual Machines. In this work, we focus on evaluating the migration costs in a real network, using container virtualization, as offered in FUTEBOL testbeds for external experimenters. The results for conducted experiments are presented in Chapter 3.

As for the factors that compose the costs associated with an orchestrated migration, 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 both were also the two variables most recurrently found in related work. Moreover, our experimental network did not offer SDN support, so the inclusion of SDN in our setup would require some network emulation (*e.g.*, using Mininet[3]), which would defeat our purpose of experiment-

[3] http://mininet.org/overview/

Table 2.1: Summary of Related Work.

Reference	Network Environment	Considered as Orchestration Cost			
		Migration Time	Data Transferred	Service Downtime	Message Signaling
[Cerroni and Callegati 2014]	Cloud	X		X	
[Ksentini, Bagaa and Taleb 2016]	5G				X
[Sun et al. 2016]	Generic		X		
[Tao et al. 2016]	Cloud	X	X		
[Ibn-Khedher et al. 2016]	Virtual CDN	X	X		
[Wang et al. 2017]	Generic				X
[Xia, Cai and Xu 2016]	Generic				X
[Xia et al. 2016]	Generic	X	X		
[Ibn-Khedher et al. 2017]	Virtual CDN	X	X		
[Lin et al. 2016]	Cloud	X		X	
[Liu et al. 2017]	Cloud	X	X	X	
[Khedher et al. 2017]	Virtual CDN	X	X		

ing strictly with real equipment. Concerning service downtime, the early investigation indicated that its measuring would require an active agent[4], 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.

[4]https://discuss.linuxcontainers.org/t/how-to-monitor-lxc-running-container-metrices-during-migration-phase/2112

3 EXPERIMENTS: QUANTIFYING MIGRATION COSTS

In this Chapter, we present the theoretical background on the migration process and optimization. By dividing the migration problem into smaller parts, we can better understand how each part weigh on total costs for migration. Based on this knowledge, we propose a series of experiments to gauge how some factors affect the migration costs.

3.1 Decomposing the Migration Process

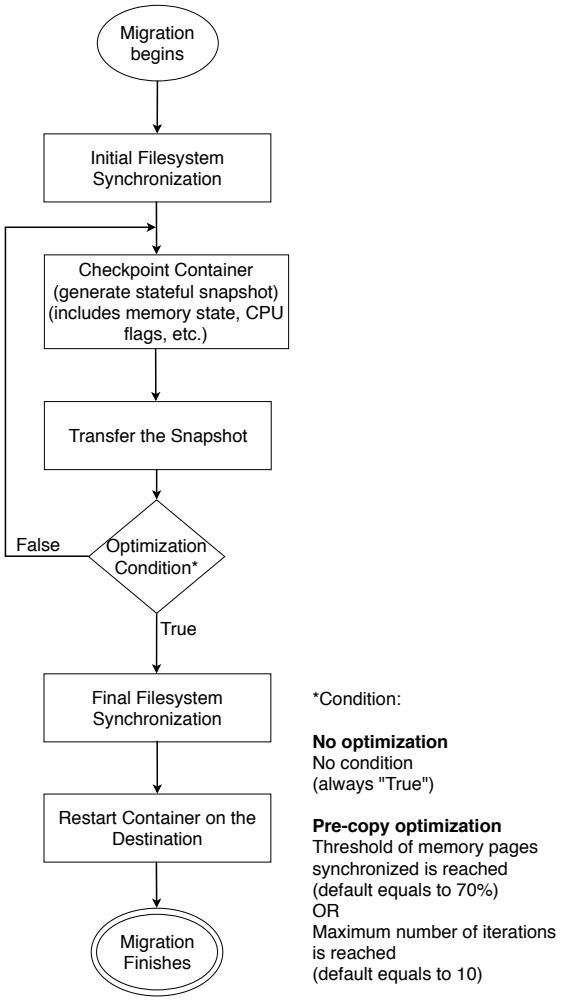
Live migration[1] is a network feature for virtualization which enables VNFs reallocation by transferring a virtual node's persistent and run-time state between hosts [Bradford et al. 2007]. The benefits from the process include minimal disruption of the service, since the downtime for the service, that is, the interval for when the node is turned off in both source and destination, is kept to minimal levels. To perform such migration, the host system must synchronize with the destination host all data required for the virtual node to resume its operation, which includes the file system, the memory pages, TCP connections, etc.

For functions virtualized by LXC containers, migration is largely composed 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 increasing the total time for migration, when adopting memory pre-copy optimization for a migration. The migration work-flow is shown in Figure 3.1.

To synchronize the operation between hosts, LXD makes use of three traffic channels: the control stream, the CRIU stream, and the file system stream[2]. The control stream initiates with the migration, and is responsible for exchanging information about the container, its configuration, and result for the restore operation. Checkpoint/Restore In Userspace (CRIU) is the tool used by LXD to save and resume a container's running state. Therefore, the CRIU stream is used to send over the container checkpoint data, synchronizing containers' run-time state. The file system stream is used to synchronize the container file system between hosts, as one would expect.

[1] In this work, unless explicitly stated otherwise, migration refers to live migration
[2] https://lxd.readthedocs.io/en/latest/migration/

Figure 3.1: Flowchart of the migration process.



Source: Author

3.2 Experimental Environment

Due to constraints that range from hardware availability to compatibility between hosts for migration, some parameters were fixed throughout the experiments. The topology used in the experiments is presented in Figure 3.2. Two VMs are configured in each physical host, and their specs, unless stated otherwise, are identical. The Virtual Function (VF) running is specified in each experiment. The most important specs for the hosts utilized are found in Table 3.1.

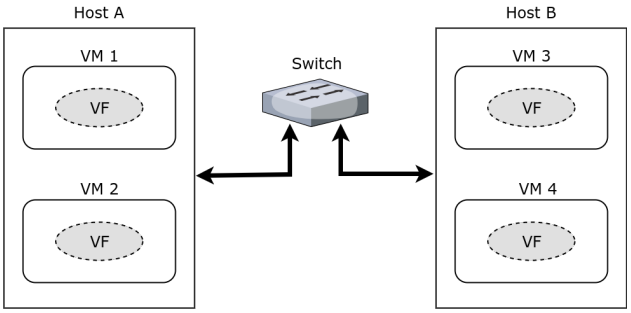
Finally, differences in the container platform versions and configuration can have a direct impact on observed metrics. For example, storage back-end options provided by LXD include Directory, B-tree file system (Btrfs), and Zettabyte File System (ZFS). Different storage back-end can offer different features[3] and therefore differ in performance

[3]https://lxd.readthedocs.io/en/stable-2.0/storage-backends/

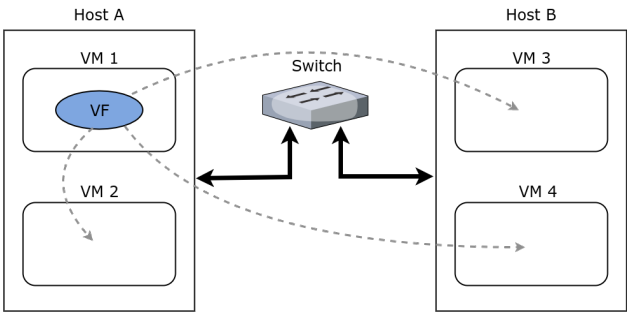
Table 3.1: Test Environment Configurations.

Physical Hosts (Hosts A, B)	
OS Version	Ubuntu 18.04.1 LTS
Linux Kernel	4.15.0-32
Network Link	Gigabit Ethernet
Virtual Hosts (VMs 1-4)	
OS Version	Ubuntu 18.04.1 LTS
Architecture	x86_64
Linux Kernel	4.15.0-36
Memory	16 GB
CPU Cores	6
CPU Frequency	2.0 GHz

Figure 3.2: Overview of the experimental setup.



(a) System topology for the experiments.



(b) Migration alternatives for a VF running in VM 1.

Source: Author

Table 3.2: Shortlist for available Linux distributions, with respective image and filesystem sizes.

Linux Distribution	Image Size (MB)	Filesystem Size (MB)
Alpine 3.8	2.34	4.4
Debian 7	91	144
Debian 10	122	187
Ubuntu 16.04	105	173
Ubuntu 17.10	118	186
Ubuntu 18.04	119	194
Ubuntu 18.04 (i686)	121	196

particularly for the tasks of container checkpointing and synchronization, relevant to this work. We utilized LXD 3.0.1, CRIU 3.10, and ZFS 0.7.5 for containers' storage back-end.

3.3 Preliminary Considerations

To advance our experiments towards estimating migration costs, we first propose two important considerations. First, we look into the choice of Linux distribution to be used when implementing a VNF. Second, we observe the impact different container variations have on each migration step.

3.3.1 Linux Distributions

LXC differs from application containers (*e.g.*, Docker[4]) 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* [Rosen 2013]. As a result, hundreds of distribution images are offered by LXD when deploying a new container, for multiple OSs and architectures. A non-exhaustive list for available distributions is shown in Table 3.2.

Ideally, containers with applications that are expected to be frequently reallocated should rank lighter distributions higher, as to minimize the reallocation overhead; rather static containers could prefer distributions that are easier to work with, as the overhead introduced by unwanted pre-installed services, for example, is not as big of a concern.

[4]https://www.docker.com/

While it is clear that the difference in filesystem size between distributions will impact on 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.

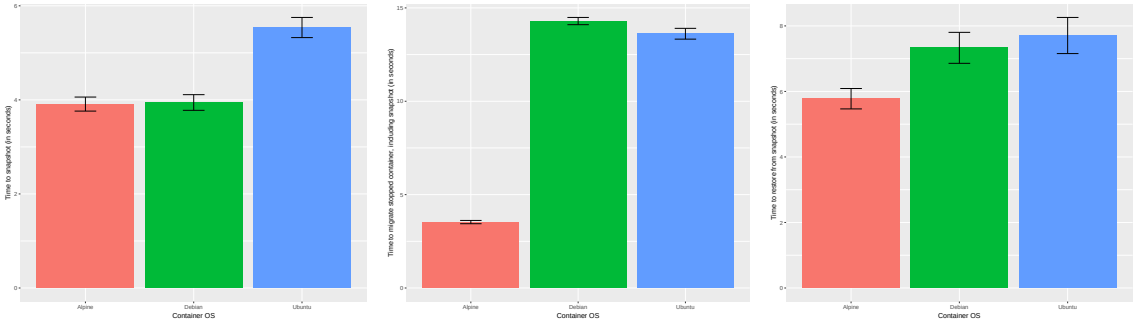
3.3.2 Cold Migration x Live Migration

Unlike the live migration process, cold migration occurs when there is a non-negligible interval for stopping the VNF on the origin, moving it to the destination, and then resuming its operation. A series of experiments were conducted to expose the cost incurred by each migration step (*i.e.*, snapshotting, moving, and restoring). As discussed in previous Chapter, we are interested in determining the costs for migration time and data transferred. Because it would be unfeasible in a timely fashion to consider too many different distributions, we restricted this experiment to three available OSs: Alpine, Debian, and Ubuntu.

Each container was kept in their minimal state, *i.e.*, no additional applications or services were installed or run on top of their installation. Each experiment was run 30 times. The results for the time consumed by snapshotting, transferring, and restoring the containers are shown in Figure 3.3. In Figure 3.3a, it is shown that the time to snapshot Alpine and Debian containers are virtually the same 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 results. In Figure 3.3b, the time to migrate the stopped container, including the run-time snapshot to be afterward restored, shows that Debian and Ubuntu containers are proportionally much more costly than the Alpine alternative; that is, of course, expected, as a direct result for the disparities in file system and image sizes between distributions, as presented in Table 3.2. The difference in migrating each distribution between hosts, both through live and cold migration, is further testified by the comparison of data transferred, shown in Figure 3.4. Figure 3.3c for the restore times shows a similar pattern to the snapshotting one, albeit the Debian container fared proportionally worse than the Alpine container.

Finally, we compared the time performance for the sum of individual cold migration steps with live migration. Because the snapshotting and the synchronizing processes in live migration are to run in parallel, as seen in Section 3.1, the component with the

Figure 3.3: Time results for the decomposed migration experiments.



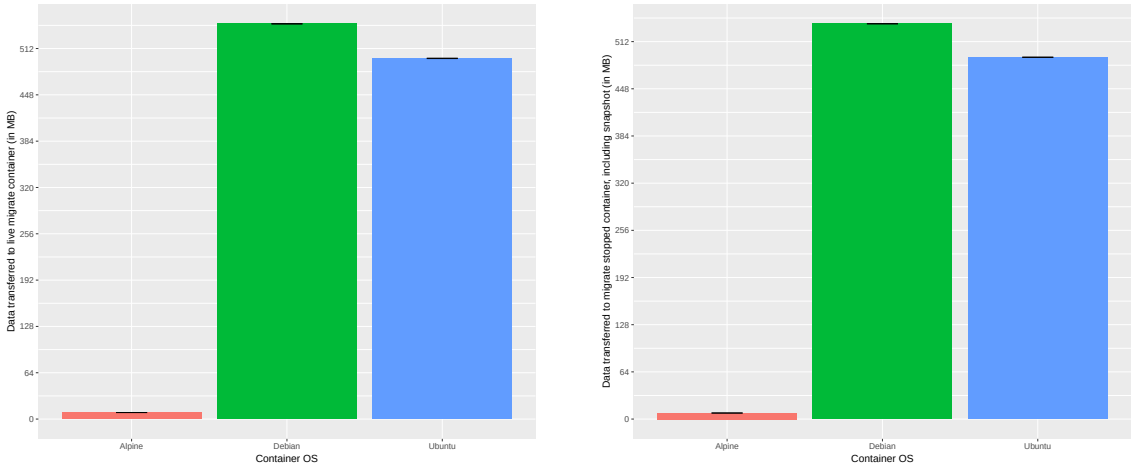
(a) Time elapsed in checkpointing the containers' run-state.

(b) Time elapsed in migrating the stopped container.

(c) Time elapsed in restoring the run-state of the container.

Source: Author

Figure 3.4: Data transferred comparison for the live and cold migration experiments.



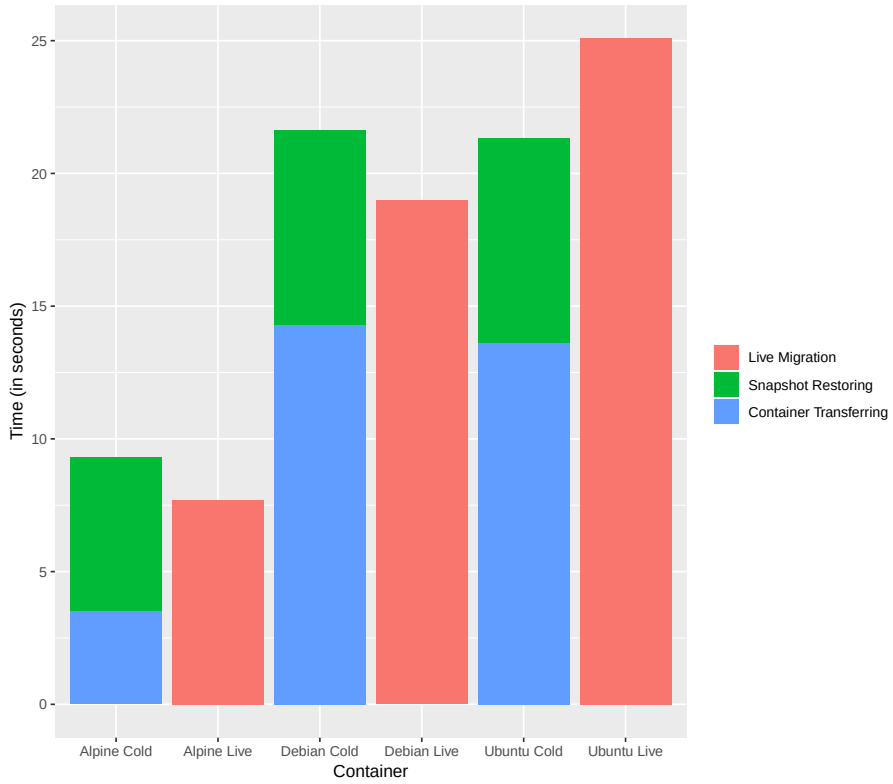
(a) Data transferred during live migration of the containers.

(b) Data transferred for the stopped container, including snapshot.

Source: Author

worse result (*i.e.*, the higher time to complete) was considered in the cold migration composition. The result is shown in Figure 3.5. 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; although a definitive answer is yet to be agreed on, optimization for live migration ignored by cold migration can be a factor in this equation, while the underlying OS for the host system, *i.e.*, Ubuntu, could help explain the anomalous behavior for the Ubuntu container.

Figure 3.5: Comparison of time results for live migration with composed cold migration.



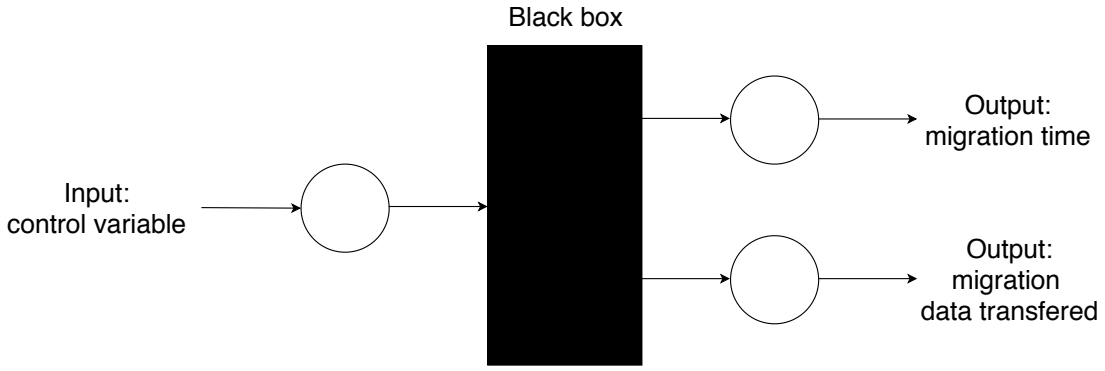
Source: Author

3.4 File Growing Experiment

In both Section 3.4 and Section 3.5, we present results for experiments conducted and analyzed in a simplified black box approach. The premise in this approach is that the output variance is a direct response to the varying of the input, *i.e.*, the controlled variable for each experiment. The experiments' design is depicted in Figure 3.6. For both experiments, the considered output is the migration costs, *i.e.*, migration time and data transferred.

In this first experiment, we focus on the stateless part of the migration, namely the required filesystem synchronization between hosts. To do so, we create a file inside the container, which is gradually increased in size, before migrating it between hosts. The growing file starts at 1MB and increases exponentially (2^n) until 1GB. For each file size considered, migration was performed five times (due to the low variance expected), *i.e.*, the experiment part for each file size was repeated five times, and migration costs for each repetition were observed. A confidence interval of 95% is used, and error bars may seem absent due to low variance observed. The results are shown in Figure 3.7.

Figure 3.6: Experiments initial design: variances in output (migration costs) are seen as result from changes in input (experimental variable).



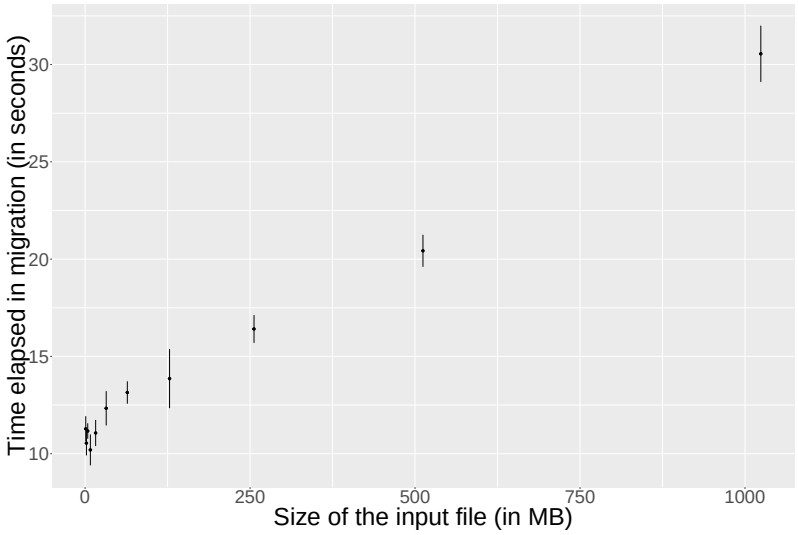
Source: Author

As observed in the results, variance in the measurements for data transferred was negligible, but was noticeable for the time elapsed by migrations. The almost non-existent variance in data transferred is somewhat expected, since the output is mostly defined by our control variable, *i.e.*, the input file size. For the time cost, however, the run-time synchronization is not as ruled by our control variable, and therefore variances of up to 5 seconds, and representing 25% of the total time, have been observed. Still, both results present a clear linear pattern, that is, the relationship between output and input, which will be further explored by our initial prediction model in Section 3.6.

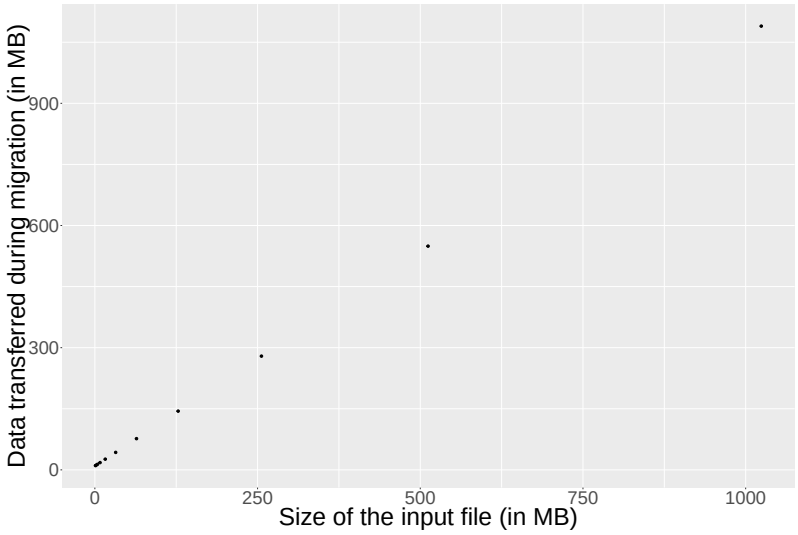
3.5 Apache Processes Experiment

Complementary to the file size experiment presented in the previous section, this second experiment focuses on the stateful part of the migration, which is performed by synchronizing the run-state of the container between hosts. For this experiment, an Apache Web Server [Apache Software Foundation 2018] is used to summon an increasing number of processes inside the container. The choice for Apache arises as a mean to minimize migration failures, commonly thrown by CRIU when migrating lesser-supported applications. The experiment starts with one Apache process summoned and increases all the way to 1000. Ten runs were performed for each processes count (the number of runs was increased in comparison to the previous experiment due to the larger variance expected), *i.e.*, the experiment part for each number of Apache process was repeated ten times, and migration costs for each repetition were observed. As with previous experi-

Figure 3.7: Results for the growing file experiment.



(a) Result for time (output) versus file size (input).



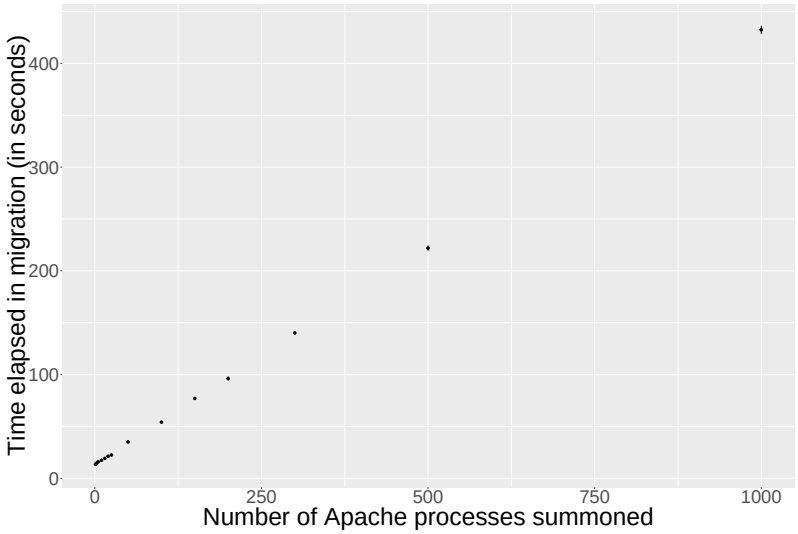
(b) Result for data transferred (output) versus file size (input).

Source: Author

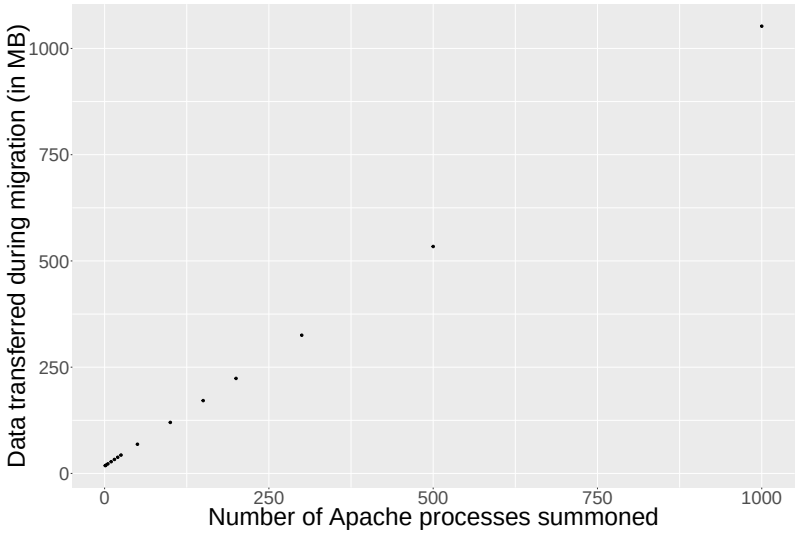
ment, a confidence interval of 95% is used, and again and error bars may seem absent due to low variance observed. The results are shown in Figure 3.8.

Even though experiments are orthogonal by nature, the results show a clear pattern for a linear component, and even with reduced relative variability in the time result. It is also clear that, while the results for data transferred in both experiments presented aligned results, the impact on migration time was much more prominent in the second experiment; that is aligned with the notion that synchronizing run-time state can be an iterative process that is costly to the migration.

Figure 3.8: Results for the Apache processes experiment.



(a) Result for time (output) versus Apache processes (input).



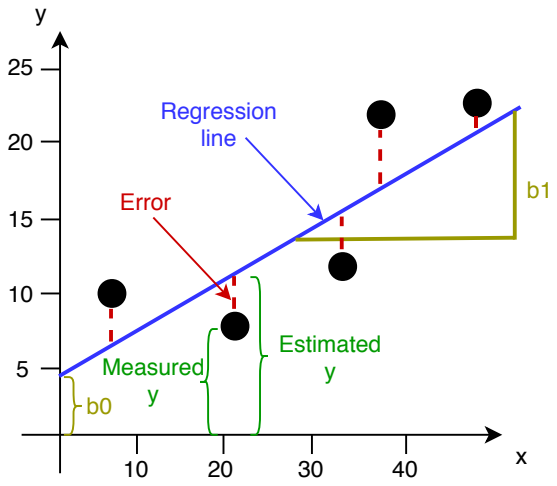
(b) Result for data transferred (output) versus Apache processes (input).

Source: Author

3.6 Simple Linear Regression

A regression model is used to estimate or predict a chosen variable as a function of one or more other variables [Jain 1990]. As a result for the linearity observed in results for experiments presented in Figures 3.7 and 3.8, we utilize a simple linear regression model to estimate costs for migration on both experiments as results for input variables.

Figure 3.9: Model for linear regression.



Source: Author

A mathematical definition of the linear model is presented in Equation 3.1. Figure 3.9 shows a graphical visualization for the model.

$$y_i = b_0 + b_1 x_i + \varepsilon_i \tag{3.1}$$

In Equation 3.1, 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. Using a criterion known as least-squares, our objective is to find the line given by Equation 3.1 that minimizes the Sum of Squared Errors (SSE). In SSE, residuals are squared before being summed, so that 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 large errors, at the expense of having 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. The 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. For the p-value, two exclusive hypotheses are considered: the null hypothesis, *i.e.*, there is no such correlation between variables; and the alternative hypothesis, *i.e.*, the correlation between variables exist. Assuming the

Table 3.3: Summary of the regression fit for the file size experiment.

File size Regression	Adjusted R-squared	p-value	Residual Standard Error
Time	0.9443	$<22E^{-17}$	1.416
Data Transferred	1	$<22E^{-17}$	0.120

null hypothesis as true, the p-value indicates the probability of observing an effect at least as extreme as the one observed in the sampled data, *i.e.*, the correlation found in the sample is due to chance, and is not found in the population. Small p-values (<0.05) therefore indicate strong evidence to reject the null hypothesis. Finally, the residual standard error measures how close the regression line is to the observed results. Residual plots are also used to confirm further that the model is correct.

3.6.1 Regression Applied to the Growing File Experiment

Applying the linear regression to the first experiment, we derive Equation 3.2 for the migration time and Equation 3.3 for the data transferred during a migration. The summary for the fits, on 53 degrees of freedom, is presented in Table 3.3.

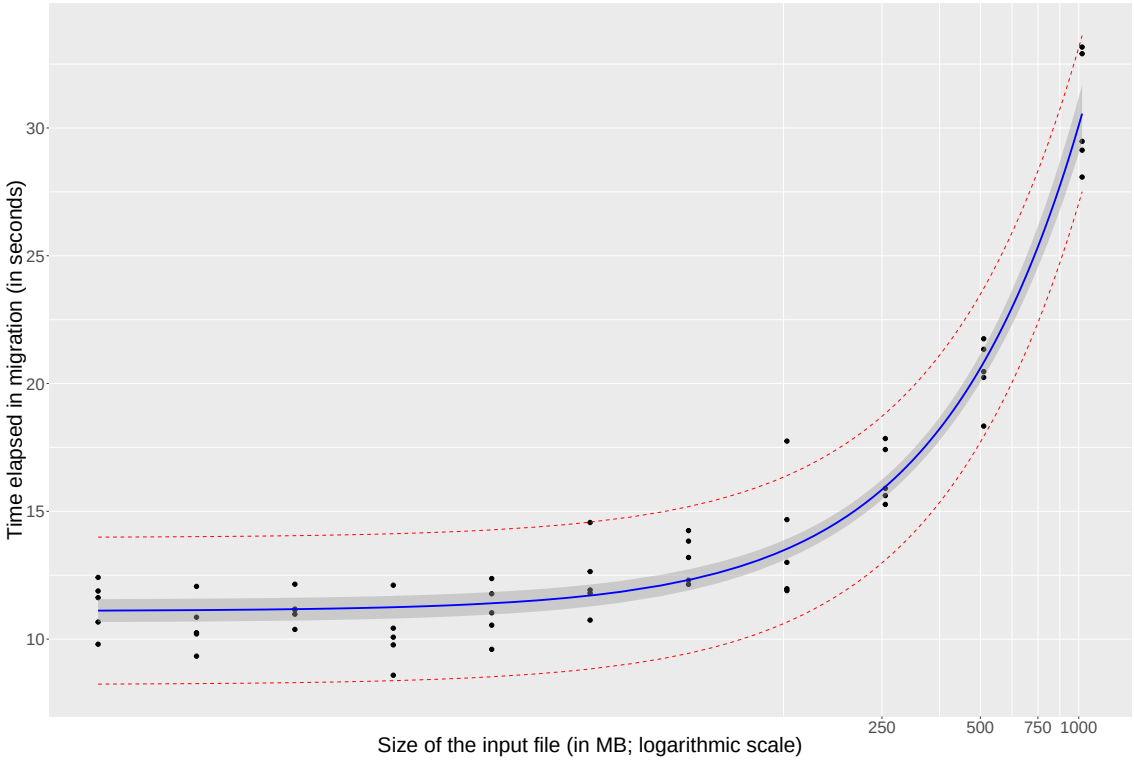
$$Y(seconds) = 11.02 + 0.02 * FileSize(MB) \tag{3.2}$$

$$Y(MB) = 9.3 + 1.05 * FileSize(MB) \tag{3.3}$$

The result for Equation 3.2 is plotted in Figure 3.10. A log-scale is used for the x-axis since the growth of the input file is exponential. The blue line in the figure 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 offer 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 run the experiment again, 95% of the sampled results are expected to fall into this approximate 2.5 seconds range from the regression line.

Similarly, the result for Equation 3.3 is plotted in Figure 3.11. Due to the low variance in the output as a response for each input value, the points for observed values stack on top of each other; for the same reason, visualization for the standard error and the

Figure 3.10: Linear regression for migration time as response for inputted file size.



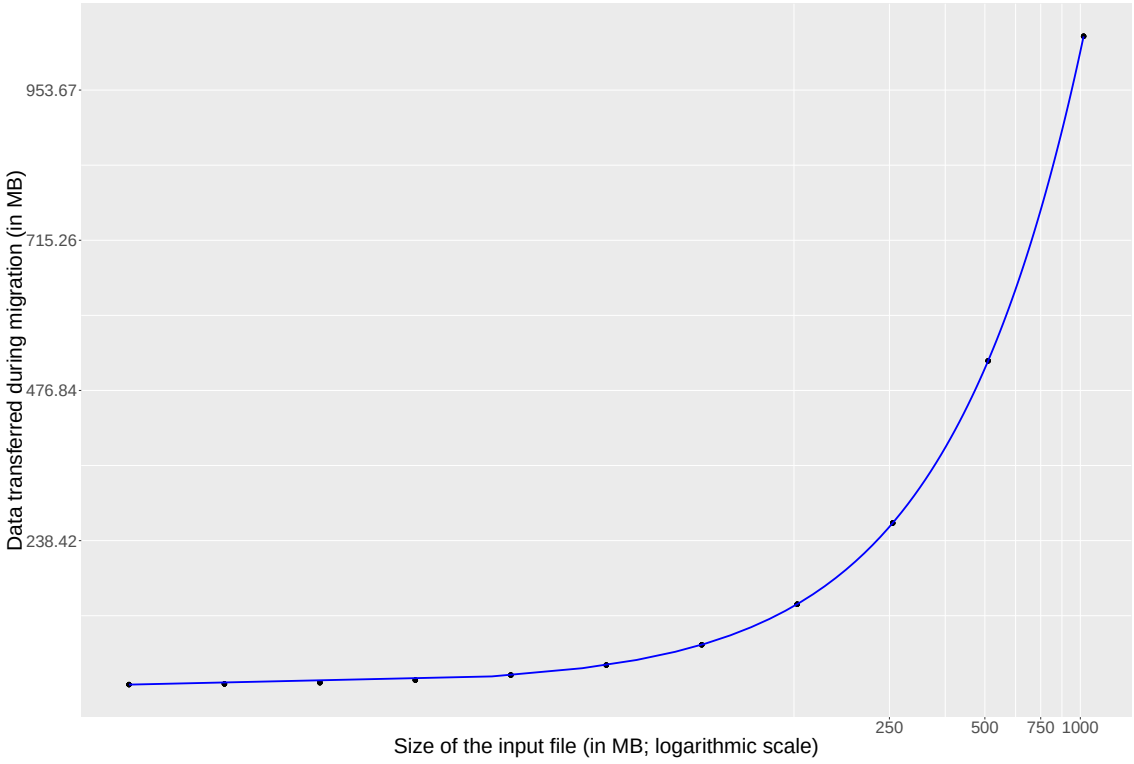
Source: Author

prediction interval are omitted in the graph. The result shows that the migration cost in terms of data transferred is almost perfectly predictable considering solely the controlled variable, for this experiment. The perfect fit is also somewhat expected, as discussed in Section 3.4, since the data transferred in the experiment is mostly defined by the stateless synchronization.

Finally, we analyze the residuals for each regression in attempt to assess that the proposed model is appropriate. The residual plots show how each observation differed from the result expected by the regression. Therefore, the plots help to spot patterns in error along the regression line, which could suggest that a different model could be a better fit for the regression. For example, the regression analysis can help spotting outliers in the results, and correctly dealing with those could help improve the regression for remaining points; if the residual plot shows a parabola pattern in the results, a quadratic component may have to be considered in the regression. Therefore, a good fit for the regression should produce residuals that are close to the regression line (low error), and randomly dispersed throughout the horizontal axis (model is appropriate).

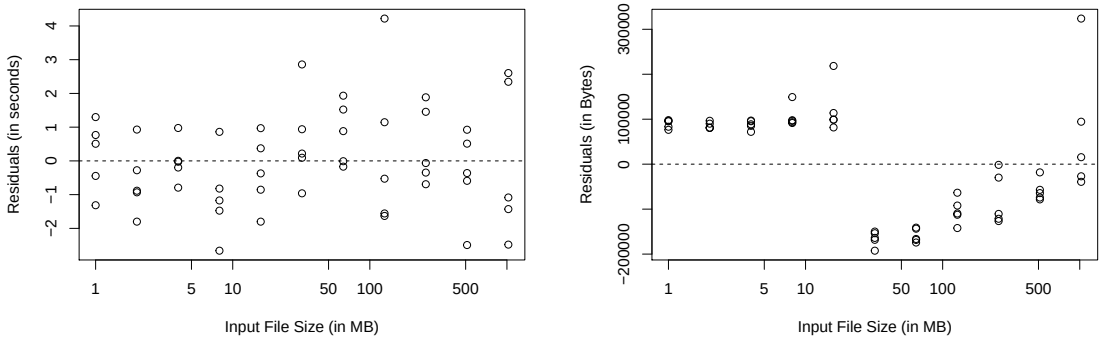
The residuals observed in Figure 3.12a, *i.e.*, the vertical distance between each observed value (dots) and the horizontal dotted line (regression line), show a rather random pattern, indicating that the proposed model is adequate. In the case of Figure 3.12b,

Figure 3.11: Linear regression for data transferred during migration as response for inputted file size.



Source: Author

Figure 3.12: Residual plots for the file size linear regression.



(a) Residuals for time versus file size. (b) Residuals for data transferred versus file size.

Source: Author

albeit errors are relatively low (<2% in every case), it is possible to identify a pattern: file sizes of 16MB or less are consistently under-predicted by the model, while the opposite happens for file sizes of 32MB or more. This pattern could be due to a relevant variable being ignored by the model, *e.g.*, an optimization for larger files being triggered by synchronization tools.

Table 3.4: Summary of the regression fit for the Apache processes experiment.

Apache processes Regression	Adjusted R-squared	p-value	Residual Standard Error
Time	0.9995	$<22E^{-17}$	2.638
Data Transferred	1	$<22E^{-17}$	0.730

3.6.2 Regression Applied to the Apache Processes Experiment

Applying the linear regression to the second experiment, we derive Equation 3.4 for the migration time and Equation 3.5 for the data transferred in a migration. The summary for the fits, on 138 degrees of freedom, is presented in Table 3.4.

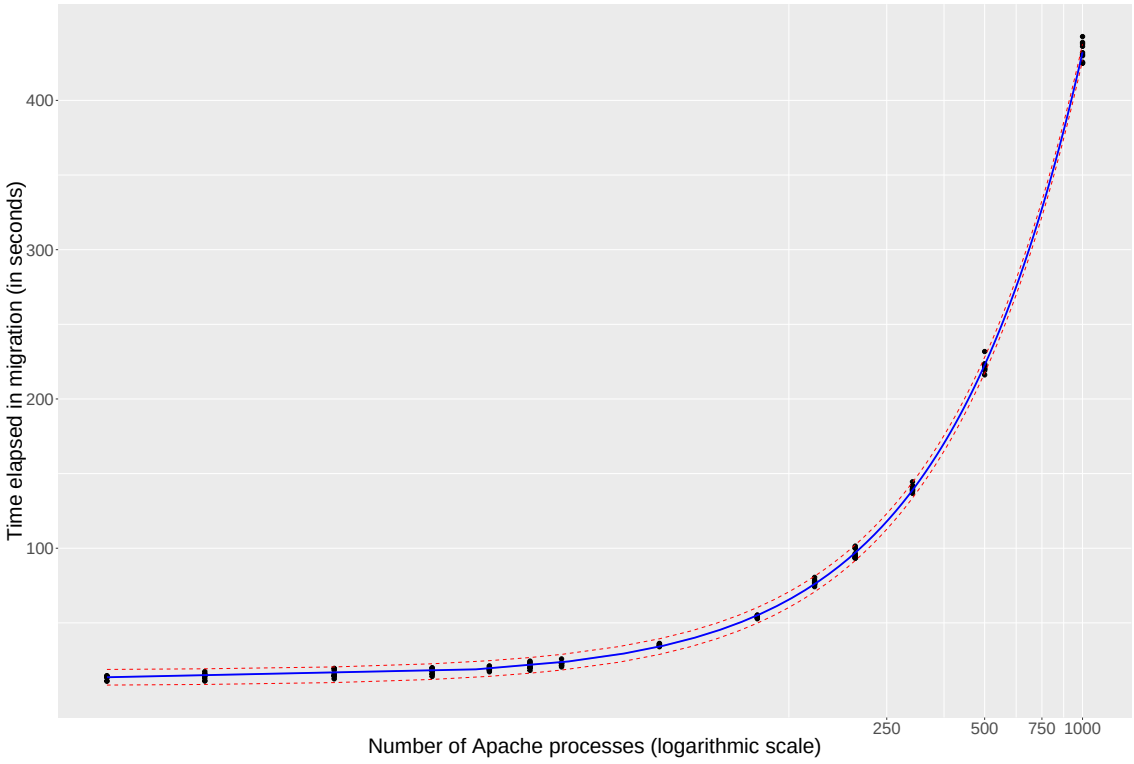
$$Y(seconds) = 13.16 + 0.42 * ApacheProcesses \tag{3.4}$$

$$Y(MB) = 16.93 + 1.035 * ApacheProcesses \tag{3.5}$$

It is noticeable that the intercept values are greater 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 and the size of the growing file impact similarly on the data transferred; also, that the number of Apache processes is significantly more costly to the migration time when compared to the size of the growing file, as previously discussed. As in with the previous experiment, results for the Equations 3.4 and 3.5 are shown in Figures 3.13 and 3.14, respectively.

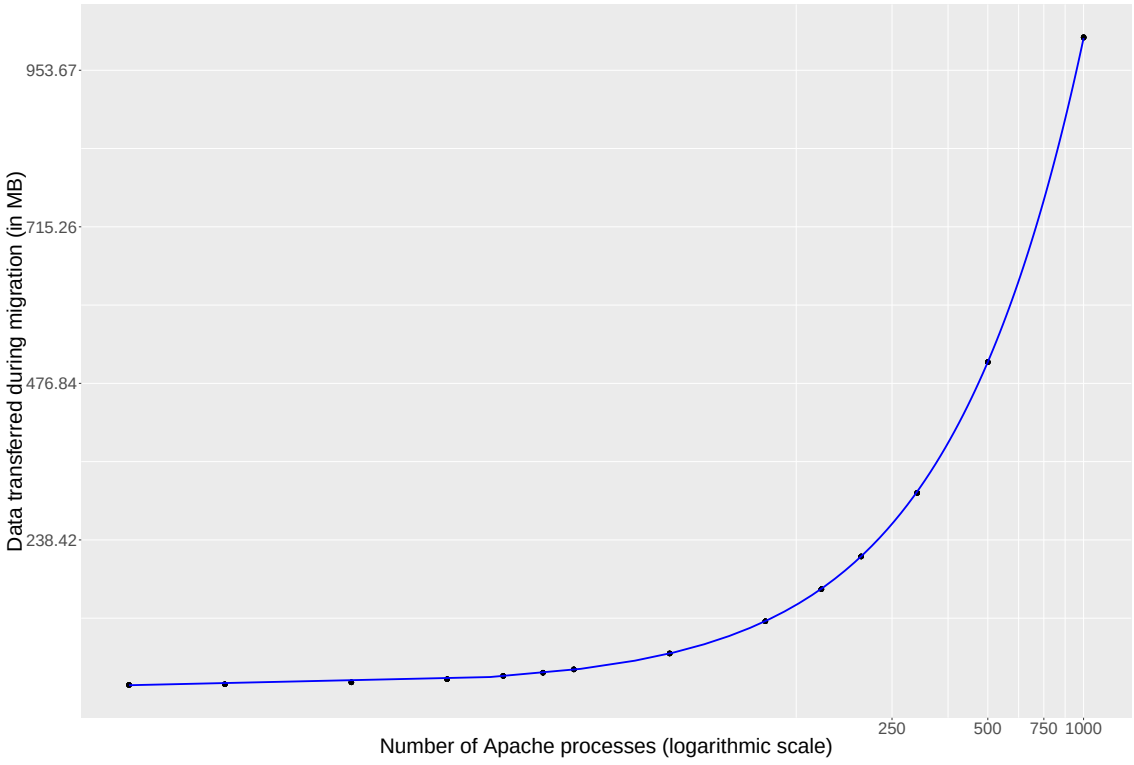
The residuals analysis for the results shown in Figure 3.15 is similar to that made for the previous experiment. Residuals for time were randomly distributed along the graph, while residuals for data transferred, again proportionally small, show a pattern for under-predicting results for the initial part of the experiment, where lower values for Apache processes are tested, and over-predicting latter results in the experiment, where the count for Apache processes reaches higher values.

Figure 3.13: Linear regression for migration time as response for inputted Apache processes.



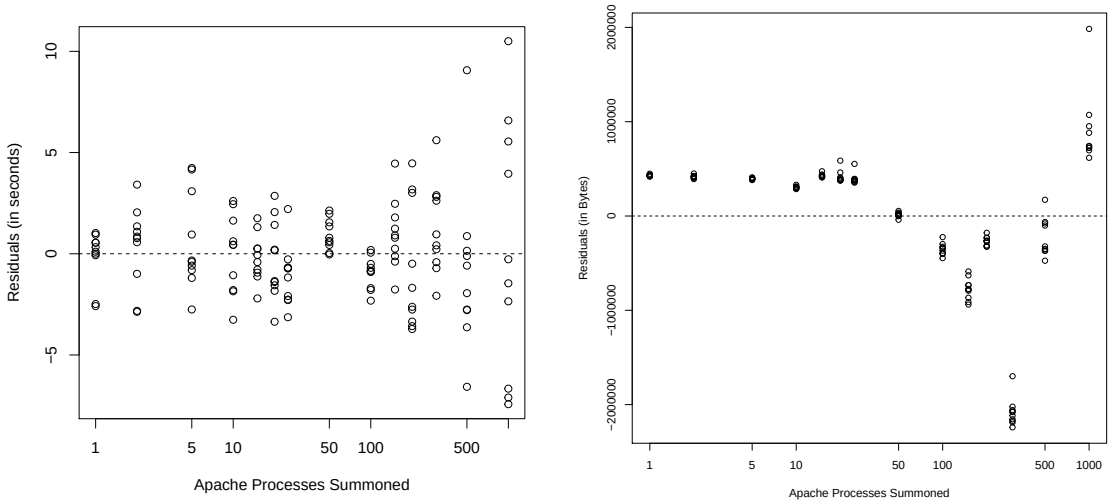
Source: Author

Figure 3.14: Linear regression for data transferred during migration as response for inputted Apache processes.



Source: Author

Figure 3.15: Residual plots for the Apache processes linear regression.



(a) Residuals for time versus Apache processes.

(b) Residuals for data transferred versus Apache processes.

Source: Author

3.7 Summary

In this Chapter, we analyzed the steps performed in a migration. By dividing the migration in smaller steps, namely the stateless and stateful synchronization, we can better understand how each step is affected by 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, showing good predictive promise for migration costs as a response for changes in experiments input.

While the regression lines obtained in this Chapter help understand the behavior observed in each experiment, the results are too constrained by the experiments' parameters to be useful in a general case. In the next Chapter, we open up the black box and try to determine which internal variables can be helpful when trying to predict migration costs for a general environment.

4 PREDICTOR MODELLING

In the previous Chapter, the black box analysis and the simple linear regression for each experiment provided promising results in the prediction of migration costs. The derived equations, however, are too narrowly fit to the scope of the experiments. To create a model that can provide cost predictions that are useful for generic applications, we must open up the black box, by now considering that the output is not directly affected by the input, but rather indirectly so. To understand the variables relationship, we thus look for the system's internal variables relevant to the migration costs prediction, and available to be used as input for predictions regardless of the application running. This process is depicted in Figure 4.1. In the next section, we discuss how internal variables were selected from each experiment, to compose the predictor model we propose for broader experimentation scenarios.

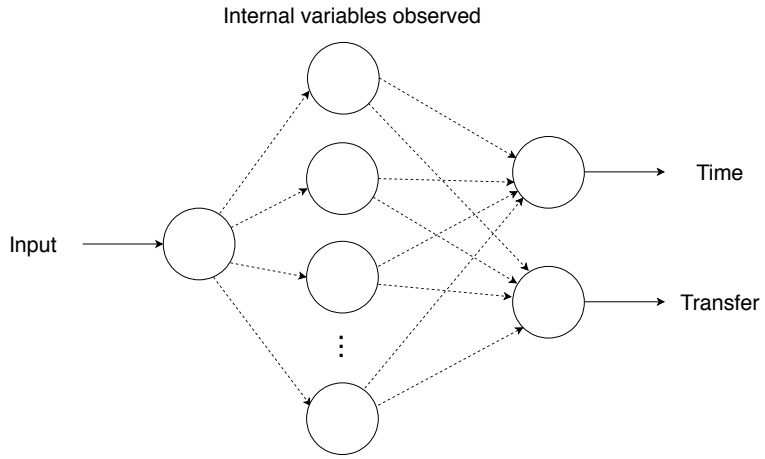
4.1 Variables Correlation

The breaking down of the growing file size and Apache processes experiments started with the monitoring of 16 different variables, including one for the experiment input, and two for the outputs (*i.e.*, migration costs, 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 for the experiments could not be measured as significant. By the end of the process, five internal variables were considered fit to be used as possible predictors for the migration costs.

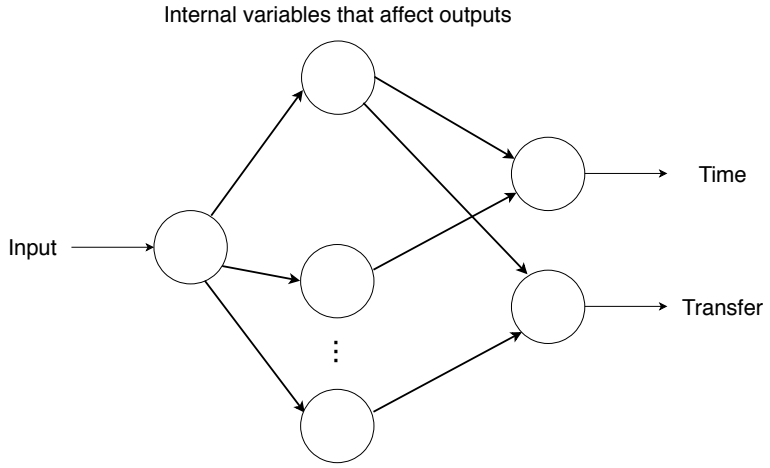
To find which internal variables are strongly correlated with the input and output for each experiment, a correlation matrix was calculated. The correlation results help to understand the dependency between each internal variable to the input from the experiments, and more importantly, the predictive association of these internal variables to the output. To assist in the visualization of the correlation results, they are presented in the form of matrix heat map. The result of the file size experiment is shown in Figure 4.2, and the result of the Apache processes is shown in Figure 4.3.

As shown in Figure 4.2, the heat map 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 each other. Strong direct or inverse correlations, *i.e.*, values

Figure 4.1: Process of determining system's internal variables which can be used as migration costs predictors.



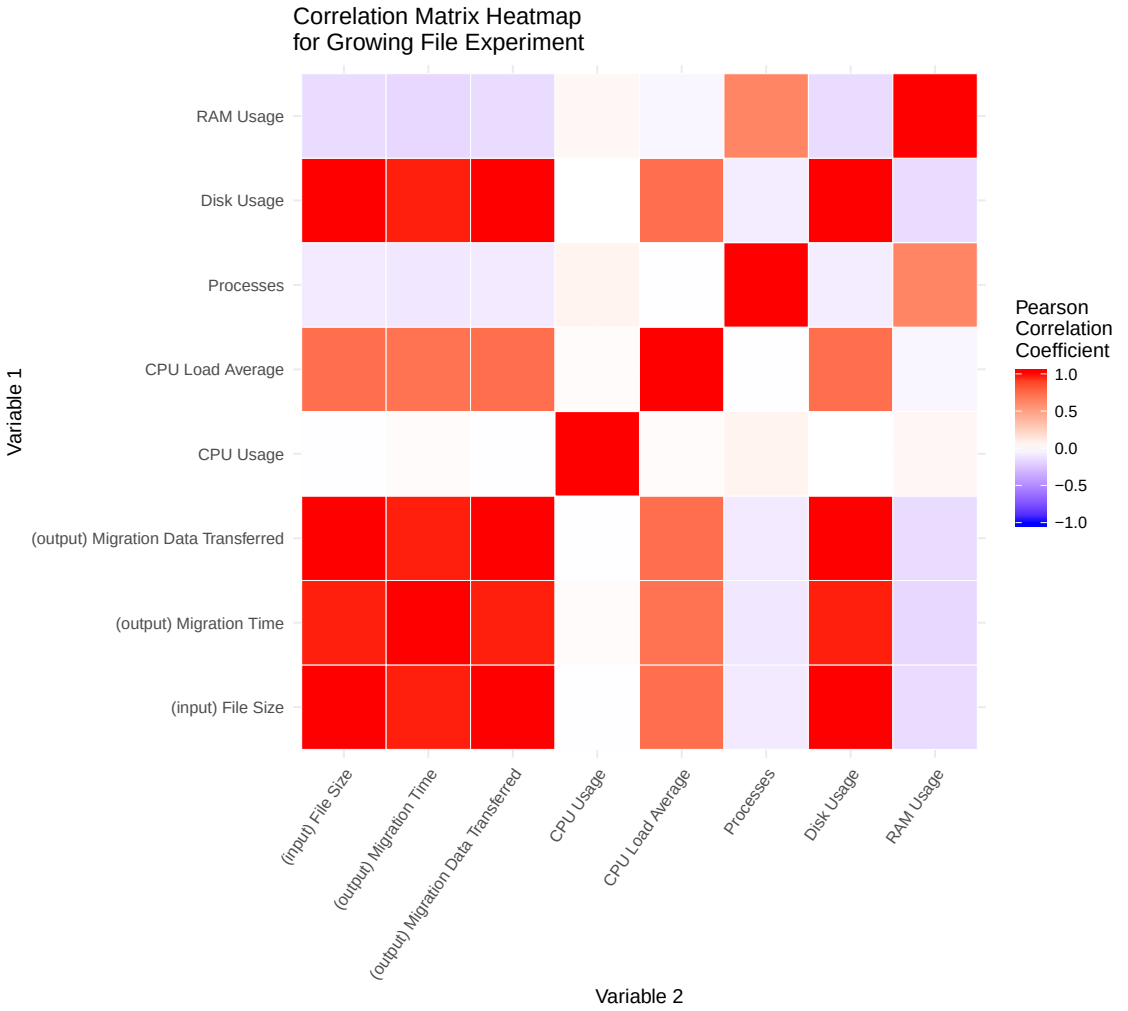
(a) A series of internal variables that could be useful as predictors are monitored during experiments.



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

Source: Author

Figure 4.2: Heatmap matrix for variables correlation in the growing file size experiment.

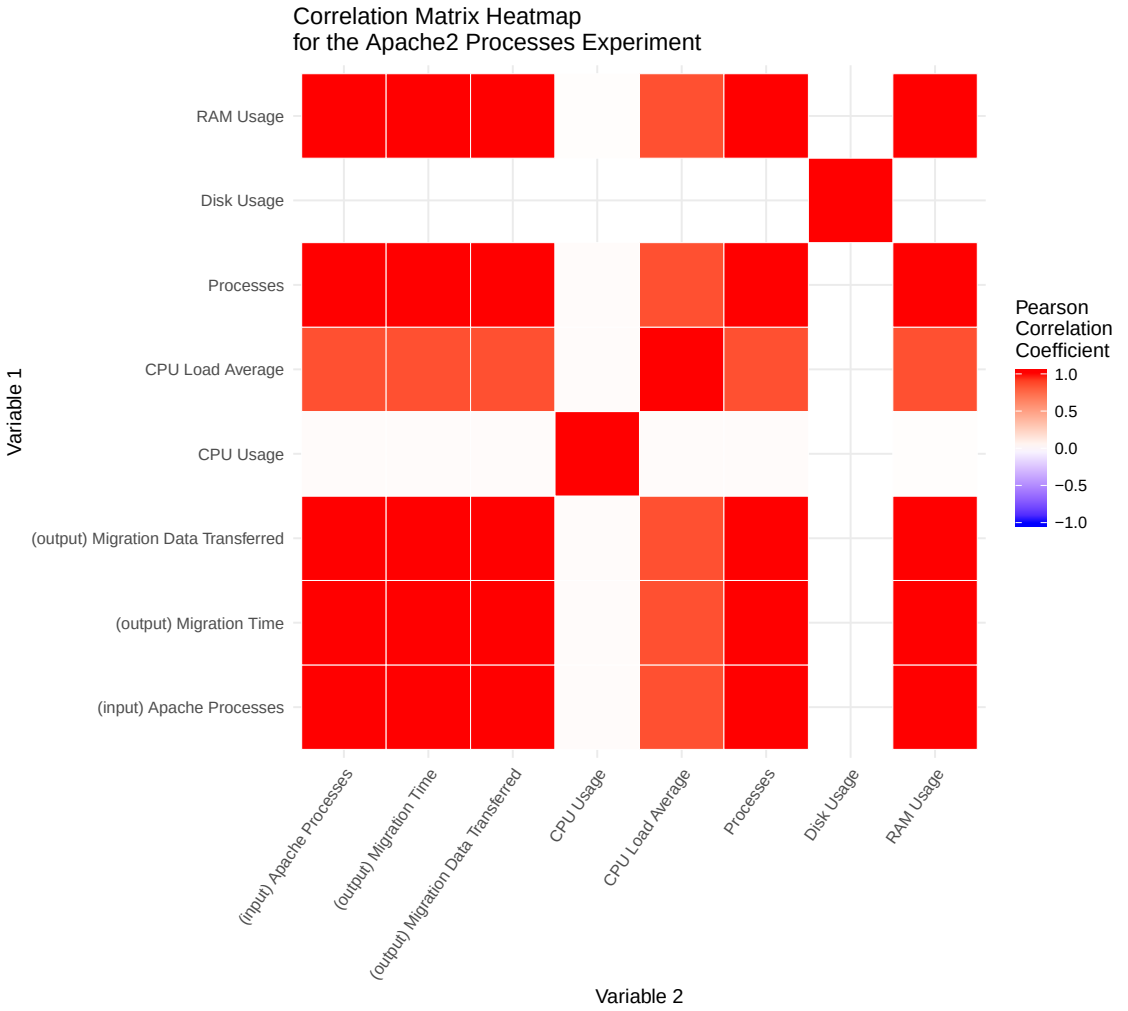


Source: Author

close to 1 or -1, respectively, are the ones we look for when composing the predictor model. In the figure, it is possible to see that "Disk Usage" was the variable with the strongest correlation to input and outputs. The correlation is also expected since the file size growth mainly affects the size of the filesystem, which in turn interferes with the stateless synchronization of the container.

For the variables in the Apache processes experiment, Figure 4.3 shows that both "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 process 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, introducing a new experiment to inde-

Figure 4.3: Heatmap matrix for variables correlation in the Apache processes experiment.



Source: Author

pendently verify the impact of each variable in the migration costs. Finally, "CPU Load Average" also showed a strong correlation, albeit weaker than that of "Processes" and "RAM Usage"; however, initial regression analysis showed a worse result for including it in the equation, which was therefore left out for the final predictor.

In this section, we detailed how the system's internal variables were selected to be used as input for our general-purpose predictor. In the next section, we investigate into the regression used to fit all the selected variables in a single prediction equation for each output.

4.2 Multivariable Regression

In the previous Chapter, it was explained how simple linear regression was used in the experiments to derive migration cost equations concerning input values. As detailed in the previous section, multiple variables are now to be used in the model for migration costs prediction [Teetor 2011]. The model for the multiple variable regression is shown in Equation 4.1.

$$y = b_0 + b_1 x_1 + b_2 x_2 + ... + b_k x_k + \varepsilon \tag{4.1}$$

As with Equation 3.1, y is the response variable, b_0 is the y-intercept, and ε is the error term. Each b_n term indicates the regression coefficient for a particular variable and is multiplied by the respective variable value, 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, so that the values found for each variable is scaled to the range of [0, 1]. The normalization formula is given by Equation 4.2. The result for the normalization equations for each variable is presented in Table 4.1.

$$Normalized(x) = \frac{x - min(x)}{max(x) - min(x)} \tag{4.2}$$

Table 4.1: Normalization formula for the input variables.

Input Variable	Normalization Formula
Disk Usage	$\frac{x-5.16}{994.84}$
Ram Usage	$\frac{x-4.11}{333.5}$
Number of Processes	$\frac{x-5}{1001}$

As discussed in the previous section, we initially consider all strongly correlated variables in our prediction model. Gradually, we fine-tuned the regression parameters, to obtain a model that, while keeping simplicity, better estimates the output values concerning the chosen input variables. We utilize R [R Core Team 2015] to perform the regressions, and ggplot2 package [Wickham 2016] to provide the visualization of the results

Table 4.2: Summary of the multivariable regression fit for the prediction model.

Multivariable Regression	Adjusted R-squared	p-value	Residual Standard Error
Time	0.9994	<22E-17 (<2E-16 for all terms)	2.562
Data Transferred	0.9992	<22E-17 (<2E-16 for all terms)	8.479

when needed. The best fit found for the migration time cost is presented in Equation 4.3, and for the data transferred in migration, in Equation 4.4.

$$Y\,(seconds) = 12.00 + 16.98 * Normalized\,Disk\,Usage + 420.68 * Normalized\,Processes \tag{4.3}$$

While the y-intercept is similar to what was obtained in the previous Chapter's 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 running state is not that of transferring the memory pages allocated by each process, but rather by iteratively confirming that every process is synchronized between hosts.

$$Y\,(MB) = 9.7 + 1073.4 * Normalized\,Disk\,Usage + 1039.6 * Normalized\,RAM\,Usage \tag{4.4}$$

Similarly to the previous result, Equation 4.4 shows a y-intercept consistent with what has been observed in Chapter 3. 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 Equation 4.3, regression coefficients do not show an as large discrepancy between variables, 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 further confirms that run-time state synchronization is disproportionally 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 confirm synchronization 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 4.2.

In this Chapter, we described how the predictor modeling evolved from experiment-aware ones, towards a generic, experiment-independent predictor. In the next Chapter, we present an use case developed in the FUTEBOL project, which we use to evaluate the accuracy of the proposed predictor model.

5 USE CASE: PREDICTIONS APPLIED IN A CLOUD NETWORK

In this Chapter, we present an use case as a way to evaluate the accuracy of our prediction model. A Cloud Computing experiment conducted in the FUTEBOL project is presented, and adaptations to our scenario are discussed before the predictor results are analyzed.

5.1 Use Case Motivation: FUTEBOL Vertically-Scalable Cloud Experiment

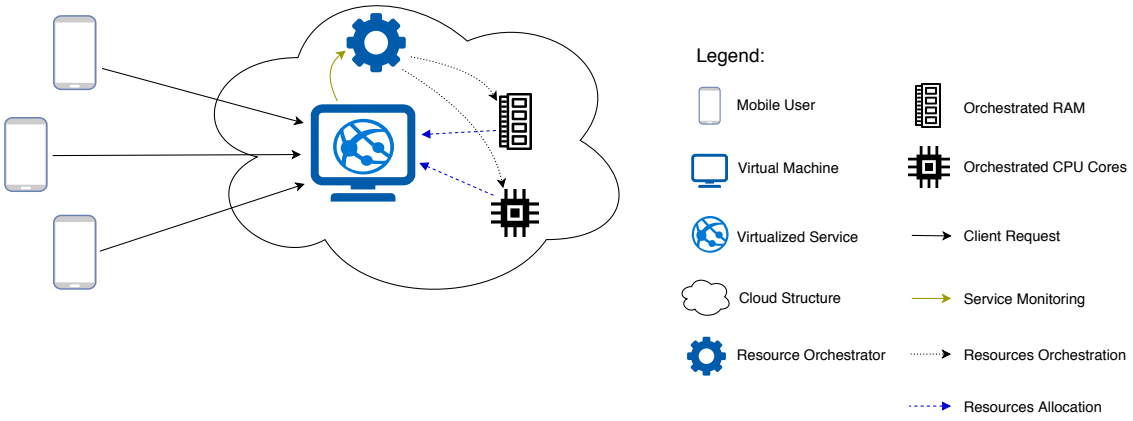
Initially, we proposed to utilize a different FUTEBOL experiment for the use case, as can be seen in Appendix A, in which core elements from a 5G network are migrated in response to user demand. Unfortunately, the core functions use General Packet Radio Service Tunneling Protocol (GTP) for connection, which by the time of this work was not supported by CRIU[1]. We thus consider another FUTEBOL experiment, in the context of Cloud Computing, to be used in our predictor evaluation.

The proposed Cloud experiment introduces an orchestrator of computational resources, namely CPU cores, and RAM, to provide vertical scaling for a virtualized service running in a container. This service is a Web application that analyzes sound signals provided by mobile users. In this application, users worried with the quality of their sleep can use a smartphone to record the ambient sound for a night of sleep and have the audio analyzed remotely. In this context, recorded sound signals are sampled, and sent to the Web service to be analyzed. Received audio samples are processed through a machine learning algorithm, which identifies patterns consistent with sleep disorders [Bublitz et al. 2017]. Clients, therefore, enter and leave the service according to their sleep schedule, which for the service's load roughly translates into periodic rises, followed later by matching falls, for the number of clients connected to the service. Additionally, due to intrinsic variances in audio sampling, the time interval between requests from a client is not constant, and the difference in processing required by any two samples can vary significantly (*e.g*, samples with fewer frequencies are easier to process than those with more frequencies). An overview of the scenario for the experiment is presented in Figure 5.1.

In this experiment, the number of concurrent clients is linearly increased over time, to a maximum of 50, at which point it briefly stabilizes, before starting to decrease until the end of the experiment, analogously. The experiment duration is of approximately 24

[1] https://github.com/checkpoint-restore/criu/issues/405

Figure 5.1: Overview of the Virtual-Scaling Cloud Experiment.

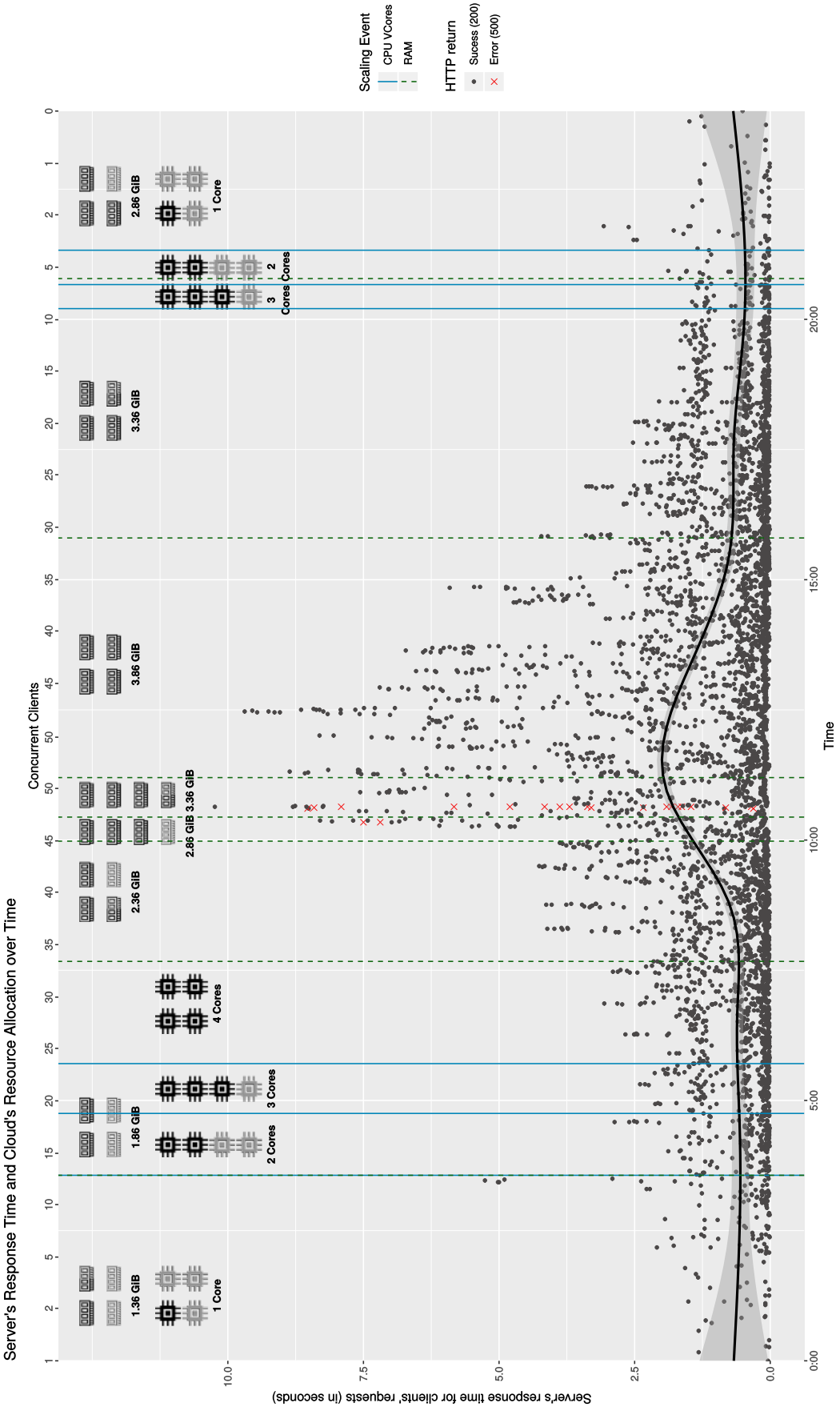


Source: Author

minutes, with clients stepping in or out every one minute. The orchestration performed scales server's resources up and down, in response to perceived utilization; orchestrated resources are RAM, from 1 to 4 GB, and virtual CPU cores, from 1 to 4. Results for the experiment are presented in Figure 5.2, where the time evolution is registered in the lower x-axis, and the clients' ladder, in the upper x-axis; the y-axis in the graph shows the processing time for every request performed along the experiment. Scaling events, *i.e.*, time instant when orchestration actions increase or decrease server's resources, are marked with vertical lines; after such event, status on the allocation of resources are depicted in the upper part of the graph.

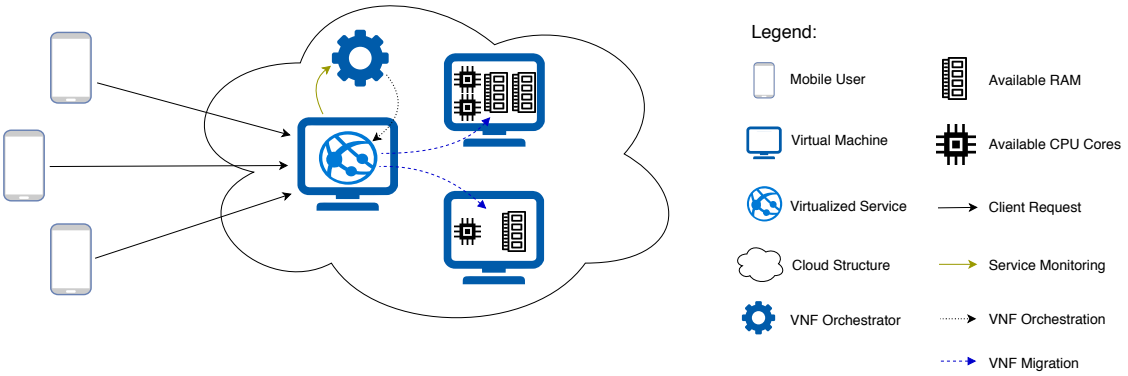
An important aspect of the experiment result is showing that the vertical scaling allows for a more efficient use of resources by the Cloud. Throughout the experiment, the average resource allocation for the server was of 2.97 virtual CPU cores, and of 2.71 GB of RAM. In a non-orchestrated environment, running the same service would require the constant allocation of 4 virtual CPU cores, and of 3.86 GB of RAM. This vertical-scaling represents an average resource-saving about 1.03 virtual CPU cores, and of 1.15 GB of RAM, over the 24 minute period of the experiment. In the next section, we adjust the experiment so that the orchestration towards efficient resource allocation is performed horizontally, rather than vertically.

Figure 5.2: Result for orchestration of Cloud vertical scaling experiment.



Source: Author; expected to appear in a future FUTEBOL journal publication

Figure 5.3: Overview of the Cloud Experiment with VNF Orchestration.



Source: Author

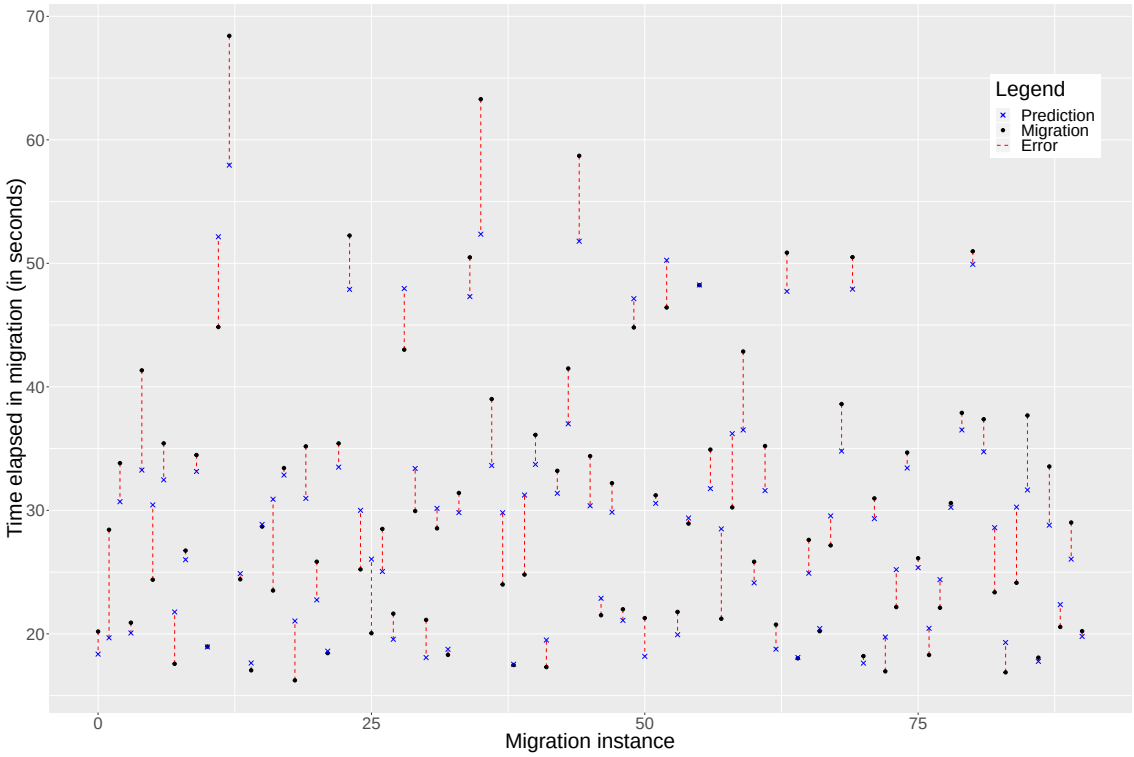
5.2 Predictor Applied to a Non-Scalable Cloud

To evaluate the accuracy of our prediction model, we use a variation of the Cloud experiment presented in the previous section. Since we are interested in VNF migrations, we propose a Cloud where VMs with different configuration are available hosts for a given VNF; analogously to the FUTEBOL Cloud experiment, an orchestrator reallocates the VNF in response to the users' demand. The virtualized service is the same as presented in Section 5.1. The proposed scenario is shown in Figure 5.3.

In the 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. The range for each workload is based on the results from FUTEBOL experiment, as presented in the previous section. 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. Once a minimum or a maximum threshold for the VM resource usage is reached, an orchestrator consults our predictor to estimate the expected migration costs, before performing a migration to a more/less resourceful VM, accordingly. Because our intention is solely to evaluate the accuracy of the predictor, and not the orchestration algorithm, migration intentions are always confirmed by the orchestrator. On the same note, our analysis is focused on the results of the predictions disregarding the impact on service performance, and thus total clients pattern is chosen, so migrations are frequently triggered. An example of a single prediction against measured migration costs is provided in Figure 5.6. The result for migrations prediction and measurements performed during

the use case experiment is shown in Figure 5.4, concerning time, and in Figure 5.5, for data transferred.

Figure 5.4: Result for time prediction and migration in the use case.



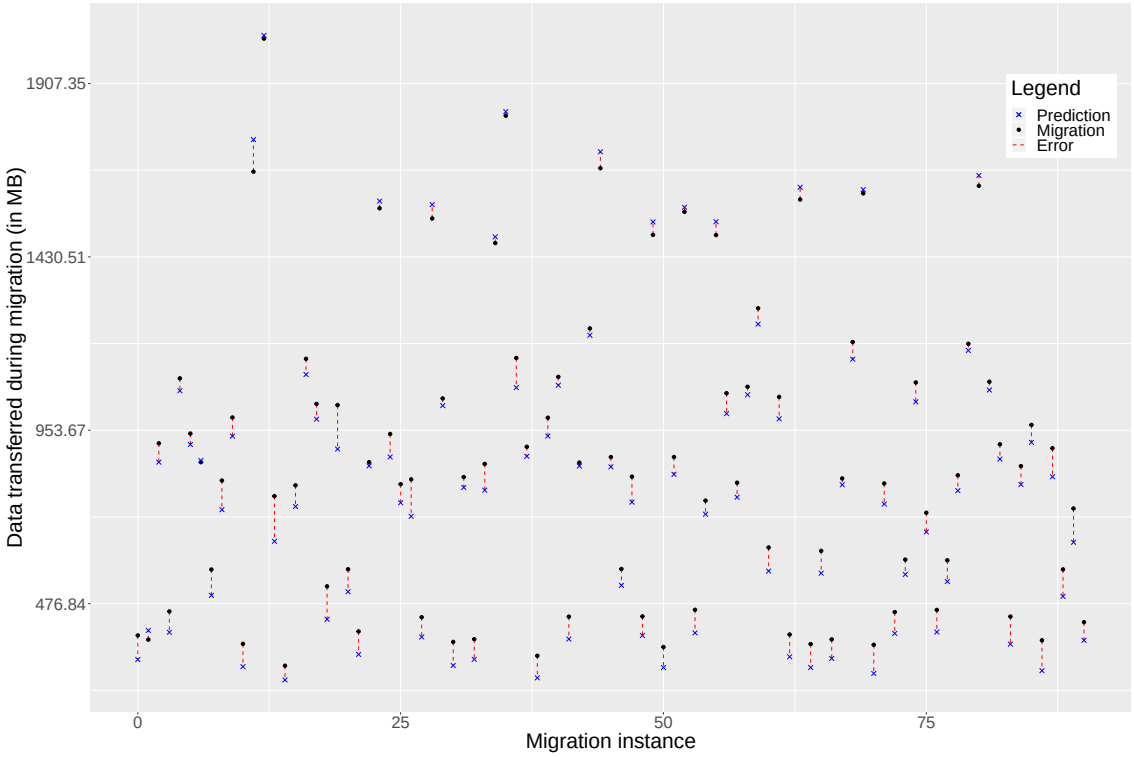
Source: Author

In both figures, it is possible to see the predicted time or data transferred cost for each triggered migration and the subsequent real measurement for its completion. The dashed red lines in the graph indicate the observed error between predicted cost and actual measurement.

5.3 Discussion of Results

The result for time costs shown in Figure 5.4 indicates that 59.34% of the measurements were under-predicted, while 40.66% were over-predicted. The Mean Absolute Error (MAE) for all observations was of 3.08 seconds, with MAE for over-predictions being of 3.15 seconds, and of 3.03 seconds for under-predictions. Considering that the average time cost was of 30.40 seconds, the mean expected error is roughly in 10% range of the predicted value. 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

Figure 5.5: Result for data transferred prediction and migration in the use case.



Source: Author

Figure 5.6: Console output showing the results for a triggered migration.

```
22 clients connected!
Migration triggered...
Cost Prediction:
Time: 27.93 seconds
Data transferred: 727.60 MB
Now migrating...
Migration finished. Time elapsed = 23.483392987 seconds; Data Transferred = 757.60 MB
```

Source: Author

may be required. For example, the analysis of the load in both source and destination host systems, and not only on the container, may provide additional resources to fine-tune the prediction model.

If we consider the residual standard error of 2.56 seconds obtained in the regression model, as presented in Section 4.2, and add a $\pm 5\%$ range in the upper and lower prediction limits, we obtain a prediction range that correctly predicts 74.73% of the observed results. As presented thoroughly in the previous Chapter, the variance in time cost is observed even in our controlled variable experiments. Therefore, the accuracy obtained within reasonable prediction range can be trustfully considered by orchestration decisions when trying to forecast expected costs.

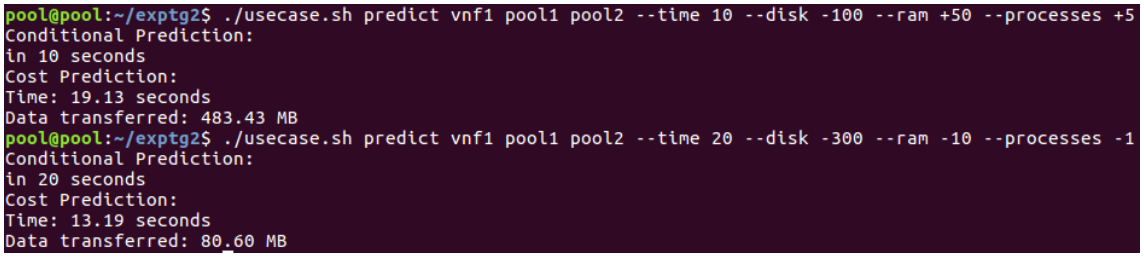
While the results in time predictions showed some slight bias towards underpredicting the costs, Figure 5.5 shows that the prediction regarding data transferred holds

a much stronger bias in the same direction. Considering the exact predicted value, 83.52% of measurements were over the estimated value, and only 16.48% were under the same value. This consistent tendency to underestimate data transferring cost indicates to us that the regression for data transferred obtained in Section 4.2 did not address all relevant parameters correctly. For example, some interaction of parameters in the use case container, not forecasted in the prediction model, result in the migration transferring consistently more than what was expected. Moreover, MAE for over-predictions was of 29.15 MB, and 56.97 MB for over-predictions, with an overall of 52.39 MB. The average data transferred in migration was of 911.54 MB, indicating that the mean error was considerably under 10% from the predicted value.

Additionally, if we shift all predictions up by 50MB to account for the observed bias throughout the use case observations, and use a $\pm 5\%$ variance to calculate the range for predictions, a total of 87.91% of the measurements would fall into the predicted range. This high accuracy obtained in such range indicates that, although the calculated y-intercept can be affected by the experimental scenario, and should be adjusted accordingly, the slope coefficient for the variance in data transferred during migration is precise in estimating the expected variance in a series of migrations. 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. If the orchestrator is knowledgeable of the VNF patterns for resources usage, it can ask the predictor for a future estimation considering the expected increase or decrease in VNF parameters. If even further, the orchestrator can alter the service behavior, it can , for example, decide that the best course of action is to cache new requests for 10 seconds, perform a rather inexpensive migration, and then to resume the requests from the new host. This conditional prediction is performed when the query to the predictor informs which parameters it expects to change, and by how much; current values are used for parameters omitted in the query. Figure 5.7 shows an example of this functionality.

Figure 5.7: Conditional prediction example: migration costs predicted considering application expectations, provided as parameters.



Source: Author

6 CONCLUSION AND FUTURE WORK

In this work, we investigated the costs associated with the orchestration of VNFs 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 move VNFs over time. Through a systematic bibliographic review, we were able to identify what costs can are present when migrating a function between hosts. Using a well-known container virtualization platform, we performed a series of experiments to better understand the association between different parameters, and the specified migration costs. The theoretical background for the migration implementation is used to separate the analysis in two major steps: stateless synchronization (*i.e.*, regarding the filesystem), and run-time synchronization (*i.e.*, regarding the memory pages, processes, etc.).

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 migration. Variables observable in non-specific function container, namely the disk and memory usage, and the number of running processes, are used to estimate migration costs at run-time, whenever an orchestrator intends to migrate a VF, and queries the predictor accordingly. The model is evaluated in a use case, in which a Cloud Computing setup produces the migration of a virtualized service between available hosts. Results for the predictions indicate that, under certain conditions, high accuracy is possible for the predictions of both costs. The predictor also offers 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 taken.

Because several parameters can impact the costs for migrations, 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 remained constant throughout our analysis, and were therefore left out of the proposed models; although the cost for data transferred in a migration should not be significantly affected by this variable, the cost in time for performing the migration is obviously affected by the available bandwidth, for instance. Therefore, 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.

Another important aspect for the future work is to make the predictor available for FUTEBOL experimenters is to integrate it with COPA. Developed in the FUTEBOL

project as a tool to assist monitoring and orchestration of services virtualized in containers, COPA allows experimenters to easily deploy and migrate containers along their experimental network. There is thus great synergy to be explored between COPA and the predictor, as COPA monitoring can be used by the predictor to improve its accuracy further, while the migration costs predictions can be fed back to COPA, improving its orchestration capabilities. A more audacious setup could include some machine-learning technique to use the migration results from users of the testbed to further enhance the predictor accuracy.

REFERENCES

Apache Software Foundation. **Apache HTTP Server**. 2018. Available from Internet: <https://wwww.apache.org>.

BRADFORD, R. et al. Live wide-area migration of virtual machines including local persistent state. In: ACM. **Proceedings of the 3rd international conference on Virtual execution environments**. [S.l.], 2007. p. 169–179.

BUBLITZ, C. F. et al. Unsupervised segmentation and classification of snoring events for mobile health. In: IEEE. **GLOBECOM 2017-2017 IEEE Global Communications Conference**. [S.l.], 2017. p. 1–6.

CELESTI, A. et al. Exploring container virtualization in iot clouds. In: **2016 IEEE International Conference on Smart Computing (SMARTCOMP)**. [S.l.: s.n.], 2016. p. 1–6.

CERRONI, W.; CALLEGATI, F. Live migration of virtual network functions in cloud-based edge networks. In: **2014 IEEE International Conference on Communications (ICC)**. [S.l.: s.n.], 2014. p. 2963–2968. ISSN 1550-3607.

COHEN, R. et al. Near optimal placement of virtual network functions. In: **2015 IEEE Conference on Computer Communications (INFOCOM)**. [S.l.: s.n.], 2015. p. 1346–1354. ISSN 0743-166X.

DALLA-COSTA, A. G. et al. Maestro: An nfv orchestrator for wireless environments aware of vnf internal compositions. In: **2017 IEEE 31st International Conference on Advanced Information Networking and Applications (AINA)**. [S.l.: s.n.], 2017. p. 484–491. ISSN 1550-445X.

DROSTE, H. et al. The metis 5g architecture: A summary of metis work on 5g architectures. In: IEEE. **Vehicular Technology Conference (VTC Spring), 2015 IEEE 81st**. [S.l.], 2015. p. 1–5.

FELTER, W. et al. An updated performance comparison of virtual machines and linux containers. In: **2015 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)**. [S.l.: s.n.], 2015. p. 171–172.

FUTEBOL. **Federated Union of Telecommunications Research Facilities for an EU-Brazil Open Laboratory**. 2018. Available from Internet: <http://www.ict-futebol.org.br/>.

HERBAUT, N. et al. Dynamic deployment and optimization of virtual content delivery networks. **IEEE MultiMedia**, IEEE, v. 24, n. 3, p. 28–37, 2017.

HERRERA, J. G.; BOTERO, J. F. Resource allocation in nfv: A comprehensive survey. **IEEE Transactions on Network and Service Management**, v. 13, n. 3, p. 518–532, Sept 2016. ISSN 1932-4537.

IBN-KHEDHER, H. et al. Opac: An optimal placement algorithm for virtual cdn. **Computer Networks**, v. 120, p. 12 – 27, 2017. ISSN 1389-1286. Available from Internet: <http://www.sciencedirect.com/science/article/pii/S1389128617301391>.

IBN-KHEDHER, H. et al. Scalable and cost efficient algorithms for virtual cdn migration. In: **2016 IEEE 41st Conference on Local Computer Networks (LCN)**. [S.l.: s.n.], 2016. p. 112–120.

JAIN, R. **The art of computer systems performance analysis: techniques for experimental design, measurement, simulation, and modeling**. [S.l.]: John Wiley & Sons, 1990.

JOY, A. M. Performance comparison between linux containers and virtual machines. In: IEEE. **Computer Engineering and Applications (ICACEA), 2015 International Conference on Advances in**. [S.l.], 2015. p. 342–346.

KHEDHER, H. et al. Optimal and cost efficient algorithm for virtual cdn orchestration. In: **2017 IEEE 42nd Conference on Local Computer Networks (LCN)**. [S.l.: s.n.], 2017. p. 61–69. ISSN 0742-1303.

KITCHENHAM, B. Procedures for performing systematic reviews. **Keele, UK, Keele University**, v. 33, n. 2004, p. 1–26, 2004.

KSENTINI, A.; BAGAA, M.; TALEB, T. On using sdn in 5g: The controller placement problem. In: **2016 IEEE Global Communications Conference (GLOBECOM)**. [S.l.: s.n.], 2016. p. 1–6.

LI, X.; QIAN, C. An nfv orchestration framework for interference-free policy enforcement. In: IEEE. **Distributed Computing Systems (ICDCS), 2016 IEEE 36th International Conference on**. [S.l.], 2016. p. 649–658.

LIN, C. C. et al. A practical model for analyzing push-based virtual machine live migration. In: **2016 7th International Conference on Cloud Computing and Big Data (CCBD)**. [S.l.: s.n.], 2016. p. 347–352.

LIU, H. et al. Performance and energy modeling for live migration of virtual machines. In: ACM. **Proceedings of the 20th international symposium on High performance distributed computing**. [S.l.], 2011. p. 171–182.

LIU, J. et al. Migration-based dynamic and practical virtual streaming agent placement for mobile adaptive live streaming. **IEEE Transactions on Network and Service Management**, 2017. ISSN 1932-4537.

LUIZELLI, M. C. et al. A fix-and-optimize approach for efficient and large scale virtual network function placement and chaining. **Computer Communications**, Elsevier, v. 102, p. 67–77, 2017.

R Core Team. **R: A Language and Environment for Statistical Computing**. Vienna, Austria, 2015. Available from Internet: <https://www.R-project.org/>.

RIGGIO, R.; RASHEED, T.; NARAYANAN, R. Virtual network functions orchestration in enterprise wlans. In: IEEE. **Integrated Network Management (IM), 2015 IFIP/IEEE International Symposium on**. [S.l.], 2015. p. 1220–1225.

ROSEN, R. Resource management: Linux kernel namespaces and cgroups. **Haifux, May**, v. 186, 2013.

SEZER, S. et al. Are we ready for sdn? implementation challenges for software-defined networks. **IEEE Communications Magazine**, v. 51, n. 7, p. 36–43, July 2013. ISSN 0163-6804.

SHARIATMADARI, H. et al. Machine-type communications: current status and future perspectives toward 5g systems. **IEEE Communications Magazine**, IEEE, v. 53, n. 9, p. 10–17, 2015.

SOLTESZ, S. et al. Container-based operating system virtualization: a scalable, high-performance alternative to hypervisors. In: ACM. **ACM SIGOPS Operating Systems Review**. [S.l.], 2007. v. 41, n. 3, p. 275–287.

SUN, Q. et al. Forecast-assisted nfv service chain deployment based on affiliation-aware vnf placement. In: **2016 IEEE Global Communications Conference (GLOBECOM)**. [S.l.: s.n.], 2016. p. 1–6.

TAO, F. et al. Bgm-bla: A new algorithm for dynamic migration of virtual machines in cloud computing. **IEEE Transactions on Services Computing**, v. 9, n. 6, p. 910–925, Nov 2016. ISSN 1939-1374.

TEETOR, P. **R cookbook: Proven recipes for data analysis, statistics, and graphics**. [S.l.]: " O'Reilly Media, Inc.", 2011.

VAKALI, A.; PALLIS, G. Content delivery networks: status and trends. **IEEE Internet Computing**, v. 7, n. 6, p. 68–74, Nov 2003. ISSN 1089-7801.

WANG, C. et al. A switch migration-based decision-making scheme for balancing load in sdn. **IEEE Access**, v. 5, p. 4537–4544, 2017.

WICKHAM, H. **ggplot2: Elegant Graphics for Data Analysis**. Springer-Verlag New York, 2016. ISBN 978-3-319-24277-4. Available from Internet: <http://ggplot2.org>.

XAVIER, M. G. et al. Performance evaluation of container-based virtualization for high performance computing environments. In: **2013 21st Euromicro International Conference on Parallel, Distributed, and Network-Based Processing**. [S.l.: s.n.], 2013. p. 233–240. ISSN 1066-6192.

XIA, J.; CAI, Z.; XU, M. Optimized virtual network functions migration for nfv. In: **2016 IEEE 22nd International Conference on Parallel and Distributed Systems (ICPADS)**. [S.l.: s.n.], 2016. p. 340–346. ISSN 1521-9097.

XIA, J. et al. Reasonably migrating virtual machine in nfv-featured networks. In: **2016 IEEE International Conference on Computer and Information Technology (CIT)**. [S.l.: s.n.], 2016. p. 361–366.

Appendices

AppendixA TG1

TG1 - Predição de Custos para Orquestração de Funções Virtualizadas por Containers

Rafael de Jesus Martins[1]
Orientador: Lisandro Zambenedetti Granville[1]
Co-orientadores: Juliano Wickboldt[1]**, Cristiano Bonato Both**[2]

[1]Instituto de Informática – Universidade Federal do Rio Grande do Sul (UFRGS)
Caixa Postal 15.064 – 91.501-970 – Porto Alegre – RS – Brazil

[2]DECESA - Universidade Federal de Ciências da Saúde de Porto Alegre (UFCSPA)
CEP 90.050-170 – Porto Alegre - RS - Brazil

Abstract. *The advent of function virtualization concept, especially that of network functions (NFV), presents critical benefits for networks of the future. Offering scalable solutions for dynamic network requirements, virtualized functions can be expanded or withdraw along the network infrastructure accordingly instantaneous demands. Although this functions orchestration present gains for network operators and clients alike, the overhead for moving functions has been little explored so far. In this paper, we present the most relevant works found in this subject, and the plans of developing a cost predictor for the migration of virtualized functions, to be done in the sequent work (TG2).*

Resumo. *O surgimento do conceito de virtualização de funções, em especial as de rede (NFV, do inglês Network Function Virtualization), traz benefícios críticos para as redes do futuro. Oferecendo soluções escaláveis para requisitos dinâmicos da rede, as funções virtualizadas podem ser expandidas e retraídas ao longo da estrutura de rede de acordo com necessidades instantâneas. Embora esta orquestração de funções represente ganhos para os operadores e os clientes da rede, o próprio custo da movimentação das funções foi pouco explorado até o momento. Neste artigo, são apresentados os principais trabalhos encontrados sobre o tema, além dos planos para o desenvolvimento de um preditor de custos para migrações de funções virtualizadas, a ser realizado no trabalho sequente (TG2).*

1. Introdução

Redes do futuro, como por exemplo a nova geração LTE 5G, serão construidas com alicerces em tecnologias como redes definidas por software (SDN, do inglês *Software Defined Network*), e virtualização de funções de rede (NFV) [6]. No caso específico de NFV, aplicações que tradicionalmente seriam realizadas apenas por equipamento especializado, conhecidos como *middleboxes*, poderão também ser oferecidas por *software* executando em *hardware* genérico. As possibilidades portanto oferecidas à gerência da rede são massivamente ampliadas, uma vez que a implantação de uma função de rede qualquer (*e.g.* um *firewall*) independe da aquisição e instalação de equipamento específico; ao contrário, para a função virtualizada podem ser oferecidos mais ou menos recursos de processamento, por exemplo, de acordo com a carga percebida na rede em cada instante.

A orquestração de funções virtualizadas de rede (VNFs, do inglês *virtuaized network functions*), *i.e.*, a tomada de decisão para inicializar, parar, ou migrar uma VNF de um *host* para outro, desempenha um papel importante na otimização de uso dos recursos de rede. A decisão de qual função orquestrar, e de que forma, têm sido alvo de múltiplos estudos, com abordagens por ângulos diversos. Contudo, nem sempre são considerados no cômputo do algoritmo de orquestração o custo induzido pelas próprias mudanças executadas na rede, *e.g.* a transferência sobre a rede para migrar uma função entre *hosts* ao executar um balanceamento de carga. O foco deste trabalho é justamente esclarecer quais são estes custos, e de que forma eles podem implicar na decisão de orquestração. Em especial, este trabalho considera o cenário das funções virtualizadas em uma rede LTE 5G, onde funções bem definidas do núcleo da rede podem ser virtualizadas, e orquestradas conforme as necessidades do operador.

O restante deste artigo está estruturado da seguinte forma: na Seção 2, os principais trabalhos relacionados são discutidos; na Seção 3, a proposta para o TG2 é descrita; resultados preliminares são apresentados na Seção 4; e na Seção 5, os comentários de conclusão são apresentados.

2. Trabalhos Relacionados

Para iniciar este trabalho, uma revisão sistemática da literatura foi realizada. Utilizando a ferramenta de busca Scopus[1], cuja principal vantagem é apresentar curadoria dos trabalhos indexados, e portanto oferecer maior relevância nos resultados, a seguinte busca foi feita:

TITLE-ABS-KEY (((nfv OR vnf OR container OR "function virtualization"OR "virtualized function") AND ((migration OR orchestration OR deployment OR placement) W/3 (cost OR penalty OR tradeoff OR risks)) ANDNOT (ship OR cargo OR sea OR disease OR patient OR food))) AND (LIMIT-TO (PUBYEAR , 2018) OR LIMIT-TO (PUBYEAR , 2017) OR LIMIT-TO (PUBYEAR , 2016) OR LIMIT-TO (PUBYEAR , 2015) OR LIMIT-TO (PUBYEAR , 2014) OR LIMIT-TO (PUBYEAR , 2013))

No dia 26/01/2018, tal pesquisa resultou em 94 documentos. Todos estes documentos foram transportos para uma planilha[2], onde foram sistematicamente categorizados, e excluídos de considerações futuras quando considerados fora do escopo. Para tal, três filtros foram aplicados sequêncialmente, com os resultados considerados irrelevantes sendo excluidos para os filtros seguintes. No primeiro filtro, o título e o resumo de cada resultado encontrado foram considerados. No segundo filtro, adicionou-se na consideração a introdução e a conclusão de um artigo relevante para o primeiro filtro. No terceiro e último filtro, o artigo inteiro é considerado. As observações mais relevantes para cada artigo, e em cada filtro, são copiadas para a planilha de resultados.

Dentre os cenários mais importantes para a orquestração de funções virtualizadas, temos o de computação na Nuvem. Na computação na Nuvem há grande interesse por parte do operador da Nuvem de obedecer os requisitos mínimos previstos em contrato com seus clientes, enquanto minimiza os custos de operação da Nuvem. Para tal, é importante, por exemplo, poder migrar as funções ao longo da Nuvem de acordo com a carga

[1] `https://www.scopus.com`

[2] `https://docs.google.com/spreadsheets/d/1zy8nuEDackDtBhioGpzQt96bPDg4d_dltYCBIXR1cAI/edit#gid=710132590`

computacional percebida em cada instante.

Quatro trabalhos de destaque foram categorizados no cenário de computação na Nuvem. O trabalho de Cerroni *et al.* [1] foca na análise de desempenho quando da migração ativa (*i.e.*, quando o estado em execução da função é movido para um outro *host*) de funções de rede virtualizadas. O desempenho é considerado pela composição do tempo total que um conjunto de migrações demora a ser realizado, e do *downtime* (intervalo mínimo em que a função é parada na origem, e retomada no destino) do serviço. Vale apena notar que, a exemplo da maior parte dos artigos verificados, o trabalho usa máquinas virtuais (VMs, do inglês *virtual machines*) para encapsular as funções virtualizadas, o que pode incutir em uma eficiência menor ao serviço, quando comparado ao uso de *containers* [2]; este ponto torna-se relevante quanto à proposta para o trabalho sequente, onde serão explorados os impactos de orquestração utilizando-se virtualização por *containers*. Na mesma linha de trabalho, Tao *et al.* [10] trataram do problema de migração dinâmica de VMs na computação na Nuvem. Os autores introduzem uma proposta para a otimização nos planos de migração, que são comparados quanto ao desempenho. Mais uma vez, o tempo total para a execução das migrações é levado em consideração para o cômputo do desempenho, com a adição do custo de transferência das VMs pela rede neste cálculo.

Ainda na análise de desempenho de migrações de VMs na computação na Nuvem, Lin *et al.* [7] utilizaram uma abordagem um pouco distinta dos anteriores. Enquanto também consideraram o custo da migração como o tempo total para esta ser realizada, e o *downtime* observado para o serviço, os autores focaram nas métricas quanto às páginas de memória a serem transferidas. Como a função a ser migrada precisa ter a memória no destino e na origem sincronizadas para finalizar o processo de migração, os autores compararam diferentes taxas de *dirtying* das páginas de memória (*i.e.* páginas que são modificadas na origem depois de já terem sido transferidas ao destino). Taxas de *dirtying* mais altas provocam uma necessidade de mais iterações no laço para sincronia de memória entre origem e destino, o que pode ser fundamental quando são consideradas as migrações de aplicações com tais padrões. Os resultados apresentam diferenças bastante significantes no custo de migração quando do aumento na taxa de *dirtying* das páginas de memória, algo que provavelmente deve ser considerado quando da predição de custos para se executar um plano de migrações. Por fim, Liu *et al.* [8] focam no cenário para transmissão de vídeo ao vivo. Em tal cenário, as migrações ao longo da rede são utilizadas para maximizar a qualidade de experiência dos usuários, enquanto minimiza os custos ao operador da rede. Para tais fins, dependendo do consumo de vídeo requisitado pelos usuários, o serviço de vídeo pode ser migrado ao longo da rede. Devido aos requisitos temporais e de banda presentes no cenário, isto é, os limites de atraso e de taxas mínimas de transferência aceitáveis, por se tratar de uma transmissão ao vivo de vídeo, os custos induzidos por uma migração devem ser bem pesados contra os beneficios esperados de uma migração. Neste caso, mais uma vez o tempo total para migrar, o *downtime* causado pela migração, e o tráfego introduzido na rede, são considerados custos da operação de migração.

Outros três trabalhos em cenários mais genéricos foram considerados neste levantamento. Nestes trabalhos, a rede onde realiza-se as migrações de FV apenas é considerada como uma com suporte a tal, sem outras características ou requisitos especiais que a diferencie. Dentre estes resultados, o estudo de Sun *et al.* [9] foca no problema de encade-

amento de serviços (SC, do inglês *Service Chain*) de VNFs. O encadeamento de serviços é necessário quando duas ou mais funções virtualizadas precisam estar conectadas pela rede (mesmo quando encontram-se em um mesmo *host* físico); o posicionamento ótimo então de uma VNF pertencente a um *chainning* depende também das VNFs as quais ela se conecta. Para reduzir a necessidade de transferência das funções pela rede, o que é eleito como principal custo de orquestração neste trabalho, os autores buscam predizer as demandas futuras das VNFs em um encadeamento, reduzindo a necessidade de posteriores migrações em resposta a variações na demanda. Resultados de simulação são apresentados, corroborando com a argumentação teórica. Ainda em cenários de redes genéricas com suporte a NFV, dois trabalhos [12, 13] por Jing Xia *et al.* aparecem com a proposta clara de otimizar as migrações de funções virtualizadas. Como em trabalhos apresentados anteriormente, os principais custos considerados para uma migração são os de transferência total, e do tempo total para a operação. Os autores propõem soluções com heurísticas para resolver o problema de reposicionamento das funções, e apresentam resultados de simulação para comparar os desempenhos das soluções.

Em uma série de trabalhos [3, 4, 5], os autores investigaram o problema de otimização na orquestração em Redes de Distribuição de Conteúdo (CDN, do inglês *Contend Delivery Network*). Em CDNs, o conteúdo (*e.g.* vídeo) disponibilizado em uma rede pode ser distribuido ou movido ao longo dos servidores desta rede, de acordo com o consumo apresentados pelos usuários. O uso de NFV e SDN por tais redes pode impactar significativamente tanto a qualidade do serviço entregue ao usuário, quanto o custo de operação da rede, implicando em grande interesse na orquestração para a área. Utilizando VMs, novamente como custos de migração são considerados o *downtime* e o tempo para concluir a operação.

Também foram encontradas abordagens diferentes para soluções usando SDN, por exemplo. No cenário de 5G, Ksentini *et al.* [6] investigaram os custos quanto as realocações necessárias do SGW. Neste caso, múltiplas instâncias da função virtualizada são utilizadas, e o controlador SDN é quem exerce o trabalho de ”migrar”os usuários de uma instância da função para outra. O custo considerado para uma migração é, portanto, o da troca de mensagens no controlador para a sinalização da migração. Contudo, esta solução não é viável para todos os cenários, *e.g.* redes sem suporte SDN, ou funções com fortes requisitos de contexto. Um resultado parecido foi encontrado para cenários mais genéricos por Wang *et al.* [11], também utilizando SDN para a migração, buscando-se balancear a carga na rede.

Como apresentados até então, a maior parte dos trabalhos focou no aspecto teórico do problema, geralmente demonstrando resultados de simulação. Ademais, os trabalhos com componente experimental utilizaram máquinas virtuais para encapsular as VNFs, tendo pouco explorado meios de virtualização mais atuais, como a virtualização por *containers*. Quanto à composição dos fatores que devem ser considerados como custo de migração, aspectos como o tempo total para se realizar uma migração, e a transferência necessária entre origem e destino, foram recorrentes na maior parte dos trabalhos; ainda assim, apesar da aparente convergência, meios para prever com acurácia tais custos, dado um plano de migrações arbitrário, foram pouco desenvolvidos até então.

3. Planejamento para o TG2

O trabalho proposto para o TG2 focará no aspecto prático do problema de estimativa de custos na orquestração de funções virtualizadas, utilizando um ambiente experimental para a obtenção de resultados. A virtualização por *containers* será utilizada para o encapsulamento das VNFs; este ponto é relevante, como citado anteriormente, dada a maior eficiência deste tipo de virtualização quando comparada a virtualização por máquinas virtuais, principal utilizada pelos trabalhos relacionados. Espera-se, pois, que os custos de migração observados sejam atenuados no caso médio utilizando-se *containers*, mas diferenças entre as implementações de virtualização, *e.g.*, nos processos de sincronização de memória entre origem e destino, podem influenciar estes resultados. É parte do trabalho proposto, pois, esclarecer de que forma impacta nos fatores de custo de migração a utilização de uma técnica de virtualização distinta daquela utilizada na maior parte dos trabalhos relacionados.

O ambiente para execução dos experimentos será aquele fornecido pelo projeto FUTEBOL[3]. Como colaboradores do projeto, os recursos disponibilizados pelos ambientes experimentais do FUTEBOL são bastante conhecidos, facilitando a execução do trabalho proposto. Em especial, será utilizada uma ferramenta desenvolvida no projeto especificamente com o intuito de facilitar o manejo de *containers* por um experimentador, conhecida como COPA (*Container Orchestrator and Provisioning Architecture*, ou Arquitetura de Orquestração e Provisionamento de Container, em tradução livre). O COPA oferece uma interface amigável para o usuário gerenciar e monitorar um ou mais *containers* em múltiplos servidores, os quais são alocados nos ambientes de experimentação disponibilizados pelo projeto. Além disso, o COPA também oferece seus recursos através de uma API, que será utilizada para a realização do experimento de forma automática, incluindo a coleta de resultados (*e.g.* tempo para migrar, tráfego entre os servidores).

O cenário avaliado será um de extensão ou sub-conjunto de um experimento do FUTEBOL. No contexto do experimento do projeto, um serviço de *streaming* de vídeo virtualizado é consumido por múltiplos clientes; a conexão destes clientes dá-se através de uma rede LTE 5G, onde elementos do núcleo são virtualizados. Neste cenário, um orquestrador manipula uma ou mais das funções virtualizadas nada rede de forma a melhorar a qualidade do serviço de usuários, enquanto mantém o consumo dos recursos de rede a um mínimo. O objetivo aqui será, então, dado um plano de modificações planejadas pelo orquestrador, criar um método para predizer os custos esperados para a execução deste plano. As estimativas dadas por este preditor serão, então, comparadas ao resultado efetivo observado quando da execução do plano de mudanças. Espera-se que, com um preditor suficientemente confiável, suas estimativas possam ser utilizadas como nova entrada para o orquestrador, para que o custo-benefício de um dado plano possa melhor refletir o observado na prática.

A Tabela 3 apresenta a sequência de passos a serem seguidos para a realização do trabalho, incluindo um plano de cronograma. O plano indica a sequência esperada de execução do trabalho, considerando o prazo para a finalização do trabalho no próximo semestre. A escrita da monografia, bem como a montagem da apresentação final, requisitos para a diplomação, dar-se-ão em paralelo com as atividades práticas, conforme estas avançarem.

[3] `http://www.ict-futebol.org.br`

Ainda quanto ao cronograma, caso os resultados esperados sejam obtidos em tempo, espera-se explorar outras opções para a solução. Como exemplo, a adição de aprendizado de máquina (do inglês, *machine learning*) no cômputo da predição de custo pode permitir ao preditor maior precisão e adaptabilidade em cenários não considerados neste trabalho. Neste caso, se o trabalho for considerado suficientemente maduro para tal, planeja-se uma submissão para uma (ainda a definir-se) revista ou conferência científica.

Agenda de Atividades	Cronograma
Execução do experimento utilizando o orquestrador	Setembro
Análise de custos experimentais	Setembro
Comparação estatística de predição com observado	Outubro
Refinamento do preditor	Outubro
Finalização dos resultados	Novembro

Tabela 1. Atividades previstas e prazos esperados para a realização do TG2.

4. Resultados Preliminares

De forma a melhor compreender o trabalho proposto para o TG2, um cenário de experimentação simplificado foi criado. Como pode-se ver na Figura 4, utilizamos dois *hosts* físicos, conectados por um *switch* em uma rede local. Em cada um dos *hosts* são instanciadas duas máquinas virtuais. A função virualizada, ou mais precisamente, o *container* que a encapsula, pode então estar executando em qualquer uma das quatro VMs. Além disso, a migração desta VF pode ser feita de qualquer VM origem, para qualquer VM destino (12 cenários possíveis, pois). Por exemplo, na Figura 4 considera-se o caso em que a VF inicialmente está rodando na VM 1. Por razões quaisquer, *e.g.* outras funções executando na mesma VM saturam o poder computacional nesta VM, deseja-se migrar a VF para outra VM. As opções de migração são, pois, para a VM 2, que encontra-se no mesmo *host* físico da VM 1, ou para uma das duas VMs que encontram-se no outro *host* (VM 3 e VM 4). Nota-se que as razões que levam a uma migração, bem como o estado instantâneo do cenário proposto, compõe a escolha do melhor destino para a VF. Mais importantemente, o interesse é comparar o impacto de migrações entre os *hosts* A e B, e de migrações entre VMs de um mesmo *host*. Uma diferença considerável deve ser observada, uma vez que a migração *inter-hosts* necessita de transferência de rede, mais lenta quando comparada a transferência ocorrida na migração *intra-host*.

Para este primeiro teste, foi criado um *container* de teste com uma imagem disponível da distribuição Linux Alpine [4]. Esta distribuição objetiva oferecer uma solução com foco em segurança e simplicidade, ideais para cenários de virtualizações de funções, pois. A imagem base é copiada para o sistema de cada uma das quatro VMs; desta forma, a migração do *container* entre as VMs é feita à partir da diferença entre o *container* e a imagem base; de outra forma, o *container* em sua totalidade teria que ser migrado quando direcionado aos servidores que não possuem a imagem, induzindo a um grande aumento na transferência (e por coseguinte, no tempo) necessária para a migração. Para este primeiro teste, não foram executadas quaisquer funções adicionais, para que o resultado seja utilizado como base de referência para *containers* futuros, executando de fato funções

[4]https://alpinelinux.org/

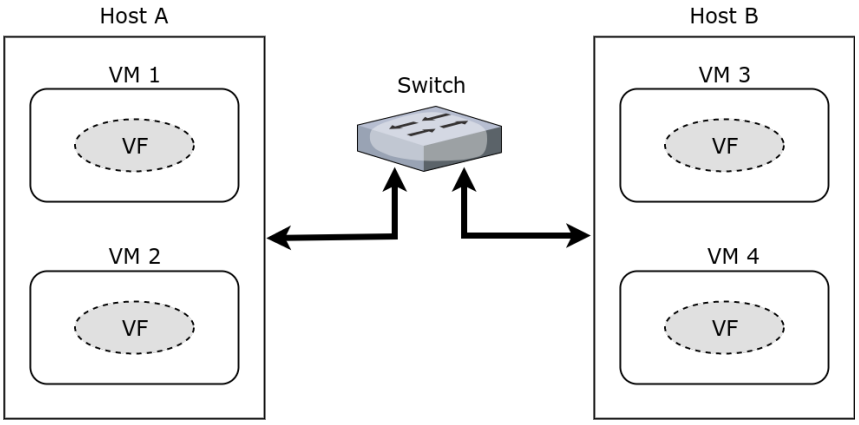


Figura 1. Cenário simplificado representando as alternativas de posicionamento de uma VF.

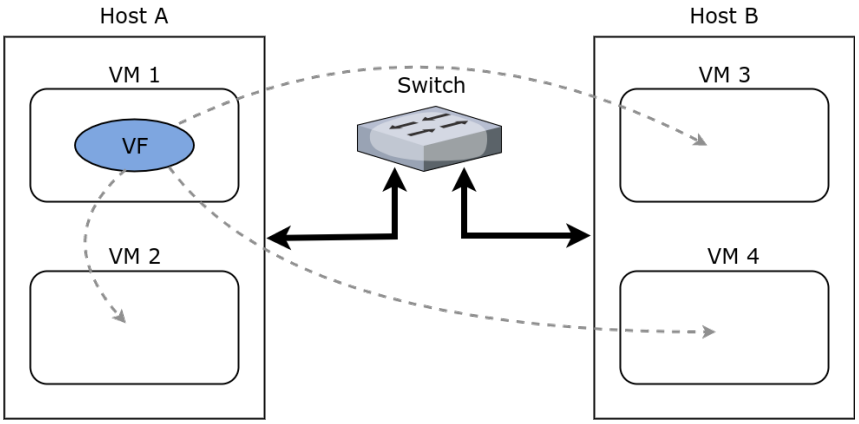


Figura 2. Migrações possíveis para a VF em VM 1.

virtualizadas. Como resultado, temos um *container* mínimo, com apenas os processos de sistema vitais ao seu funcionamento em execução, como pode ser visto na Listagem 1. Como já mencionado anteriormente, a carga computacional pode interferir nos custos de uma migração, como é o caso com diferentes padrões de alteração de páginas da memória. Ademais, o tráfego introduzido por uma função pode competir com o próprio tráfego para a migração, tornando-a menos eficiente. Desta forma, espera-se que os resultados para as migrações aqui sejam os melhores obtíveis neste cenário.

Listing 1. Listagem de todos os processos executando no *container* mínimo de teste

```
pool@pool:~$ lxc exec -- alp ps -A
PID   USER     TIME  COMMAND
    1 root      0:00 /sbin/init
  211 root      0:00 /sbin/syslogd -Z
  238 root      0:01 /usr/sbin/crond -c /etc/crontabs
  282 root      0:00 udhcpc -b -p /var/run/udhcpc.eth0.pid
    -i eth0 -x hostname:
  293 root      0:00 /sbin/getty 38400 console
  658 root      0:00 ps -A
```

Para este cenário de testes, realizou-se a migração entre servidores 60 vezes para o caso *intra-host* (30 em cada direção), e 60 vezes para o caso *inter-host* (novamente, 30 em cada direção), de forma a obter-se um resultado com maior significância estatística. Observou-se como custos o tempo médio para a migração, o tráfego médio gerado por uma migração. Deve-se notar que os valores de transferências na rede são obtidos diretamente da interface de rede; assim sendo, alguma variância é esperada, devido ao tráfego de pacotes de ARP, por exemplo. Para a sequência deste trabalho, a ferramenta COPA deve facilitar na coleta e visualização de medidas mais precisas. Os resultados obtidos são apresentados na Figura 4.

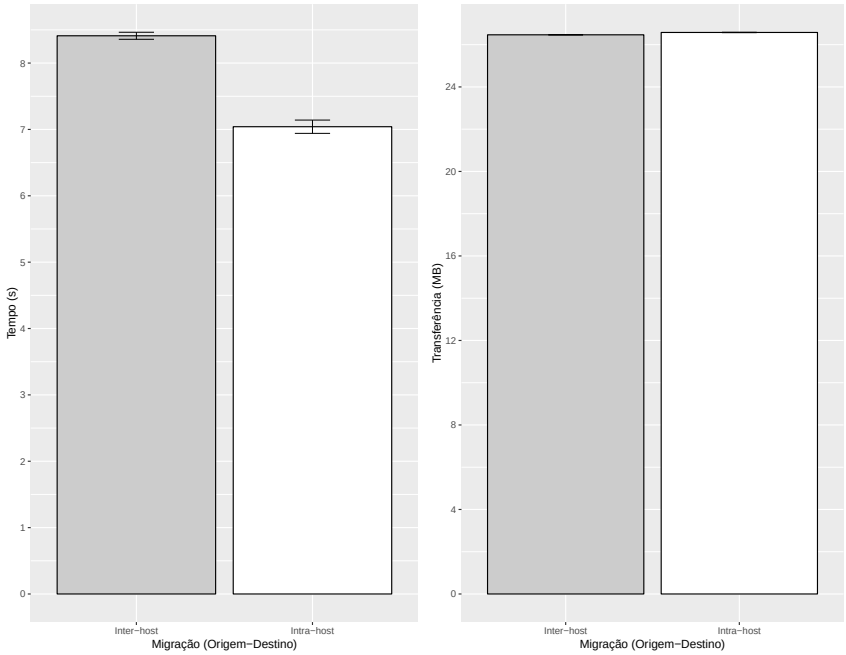


Figura 3. Resultados para custos de migração em tempo e transferência, para um *container* mínimo.

Os resultados mostram que a migração inter-*host* em média foi 20% mais lenta do que a migração intra-*host*, mesmo com medidas de transferência praticamente idênticas. Além disso, nota-se que o desvio padrão da transferência é praticamente nulo, e de valor aproximado de 26MB, o que confirma o pequeno tamanho de imagem da distribuição utilizada, bem como da facilidade de sincronizar-se o *container* mínimo por conta da estaticidade de seu estado interno.

Em seguida, sabendo-se que a carga computacional deve impactar nos custos de migração, um novo caso de teste foi feito. Para este teste, uma função para introduzir carga de processamento foi virtualizada no *container*. O *script* executado por tal função pode ser visto na Listagem 2 (baseado em resposta fornecida *online*[5]).

Listing 2. Script para introduzir carga de processamento no *container*.

```
#!/bin/bash
```

[5] https://stackoverflow.com/questions/2925606/how-to-create-a-cpu-spike-with-a-bash-

```
while [ 1 ] ; do
    echo $((13**99)) 1>/dev/null 2>&1
    sleep 0.001
done
```

Deve notar-se aqui que um primeiro teste foi realizado para uma carga de aproximadamente 100% na CPU (removendo a linha 4 do *script*), para o qual a migração falhou. Estima-se que, por ser impossível para o processo da migração sincronizar origem e destino em tempo hábil, tal função não poderia ser migrada nestes termos. Sendo assim, alternativas devem ser buscadas para tais funções, como por exemplo a suspensão da função antes de migrá-la, sendo resumida posteriormente no destino; obviamente, tal solução incutiria em outros custos, como o *downtime* da função entre ser suspensa na origem, e retomada no destino.

Para o caso presente, foi reduzida a carga aplicada à CPU para aproximadamente 20% (novamente, devido a linha 4 do *script*), permitindo a correta migração da VF entre as VMs. Novamente, os custos de migração foram obtidos em função do tempo e do tráfego médios, e são apresentado na Figura 4.

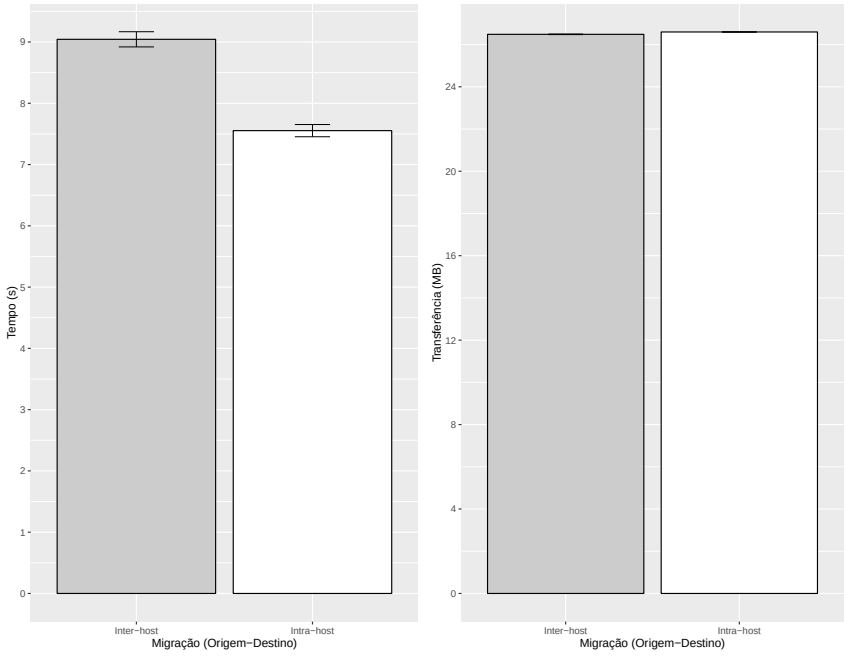


Figura 4. Resultados para custos de migração em tempo e transferência, para o *container* rodando uma VF para carga constante de CPU de aproximadamente 20%.

Comparando os resultados com aqueles obtidos no primeiro teste (Figura 4), vemos que, apesar de os resultados para transferência terem permanecidos praticamente os mesmos, houve um acréscimo de aproximadamente 0.5 segudos no tempo total, tanto para o caso *intra*-host como para o caso *inter*-host. Mais precisamente, para o primeiro caso, o tempo representou um acréscimo de aproximadamente 7.3% no tempo total; para o segundo caso, o acréscimo foi de 7.5%. Vê-se que, embora a diferença possa ser considerada pequena aqui, casos mais extremos podem tanto provocar um maior impacto nos

custos de migração (*e.g.* a migração falha quando a VF consome 100% da CPU), quanto podem ser mais sensíveis a estes custos.

Por fim, um novo teste foi feito, adicionando um fluxo de dados entre o *container* e a VM destino, de forma a melhor avaliar o impacto no tempo da migração. Para tal, foi utilizado a ferramenta *iperf* [6], que possibilita diversos testes referentes a vazão em redes IP. Com a ferramenta, primeiro constatou-se que a largura da banda entre os *hosts físicos* deve ser de 1 Gbit/s (mediu-se 830 Mbit/s com a ferramenta). Aqui, é importante salientar que a vazão de rede apresentada no cenário (mesmo para o caso *inter-host*) é muito superior ao que espera-se em cenários mais complexos, em especial quando considerarmos tráfego sobre redes com menos garantias, como na Internet. Nota-se ainda que, como mencionado anteriormente, os resultados de tráfego para os experimentos foram obtidos através dos contadores da interface de rede; como aqui a própria função virtualizada introduz a maior parte do tráfego medido, tal resultado diz menos sobre o custo de migração do que sobre a execução da própria VF, e portanto foram aqui desprezados. Vale salientar que os recursos de monitoramento fornecidos pelo COPA devem permitir análises mais profundas em casos assim, a serem tratados nos experimentos futuros. Os resultados obtidos são apresentados na Figura 4

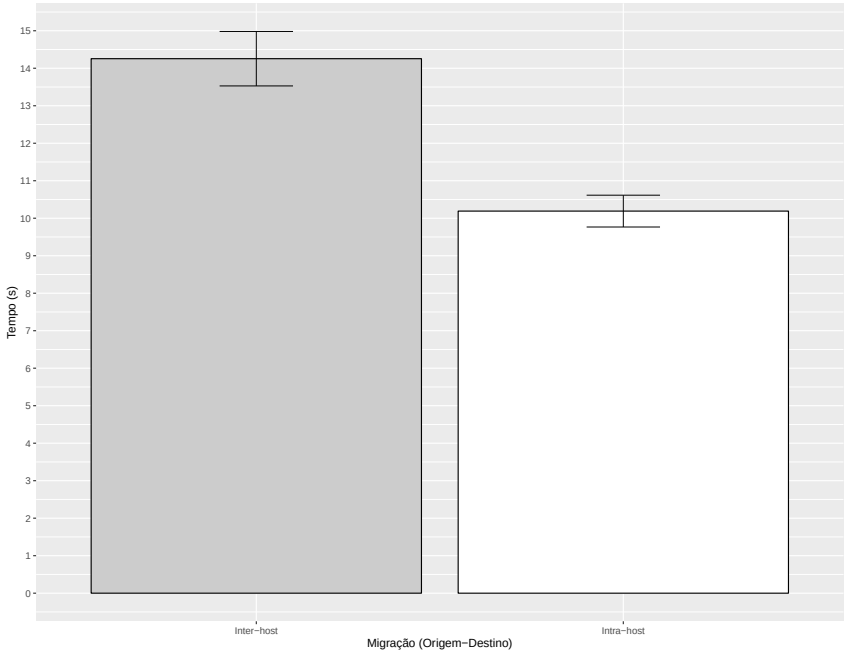


Figura 5. Resultados para custos de migração em tempo, para um *container* executando uma VF introduzindo tráfego entre as VMs origem e destino.

Como esperado, vê-se que o impacto da migração da VF é substancialmente dependente das definições de *host* de origem e destino. O tempo acrescido na migração intra-*host*, quando comparado ao medido no primeiro experimento (Figura 4), foi de aproximadamente 47%; na comparação análoga para a migração inter-*host*, o acréscimo foi de aproximadamente 70%. Além disso, nota-se que o erro padrão foi substancialmente maior do que nos experimentos anteriores, o que denota o gargalo para a sincronização

[6]https://iperf.fr/

entre origem e destino causado pela VF. Conforme a topologia da rede torna-se mais complexa, como é de esperar-se em cenários mais realistas, supõe-se que tais diferenças sejam ainda mais acentuadas, uma vez que a conexão entre dois *hosts* pode dar-se por um número muito maior de saltos, com *links* compartilhados/congestionados, etc. Por outro lado, redes com suporte a engenharia de tráfego (*e.g* SDN), podem fazer uso de sua versatilidade e utilizar *links* diferentes para o fluxo da VF e o fluxo da migração, evitando criar tais gargalos na rede.

5. Conclusão

A virtualização de funções de rede terá um papel importante nas redes do futuro, possibilitando uma maior adaptabilidade por parte da rede a um custo financeiramente viável às operadoras de rede. Para maximizar seu benefício, a orquestração das funções ao longo da rede de acordo com demandas instantâneas se faz presente. Desta forma, a capacidade de predizer os custos incorridos por migrações planejadas pode aprimorar um orquestrador com mais e melhores informações quando da tomada de decisão. O projeto da sequência deste trabalho é investigar a composição desses custos, e oferecer tais predições a um orquestrador de funções virtualizadas. Desta forma, um orquestrador pode comparar predições de custos para planos alternativos, por exemplo, e interferir na rede da maneira mais eficiente possível.

Para além do trabalho de conclusão de curso, o resultado tanto encontra suporte, quanto pretende auxiliar no desenvolvimento do projeto FUTEBOL, tal qual posto nas seções anteriores. Assim sendo, espera-se dar continuação no trabalho como recurso disponibilizado pelo projeto mesmo após a apresentação do trabalho como tema de TCC, seja com acréscimos nas funcionalidades, seja com maior integração com os outros recursos do projeto.

Referências

[1] W. Cerroni e F. Callegati. “Live migration of virtual network functions in cloud-based edge networks”. Em: *2014 IEEE International Conference on Communications (ICC)*. 2014, pp. 2963–2968. DOI: 10.1109/ICC.2014.6883775.

[2] W. Felter et al. “An updated performance comparison of virtual machines and Linux containers”. Em: *2015 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. 2015, pp. 171–172. DOI: 10.1109/ISPASS.2015.7095802.

[3] H. Ibn-Khedher et al. “Scalable and Cost Efficient Algorithms for Virtual CDN Migration”. Em: *2016 IEEE 41st Conference on Local Computer Networks (LCN)*. 2016, pp. 112–120. DOI: 10.1109/LCN.2016.23.

[4] Hatem Ibn-Khedher et al. “OPAC: An optimal placement algorithm for virtual CDN”. Em: *Computer Networks* 120 (2017), pp. 12 –27. ISSN: 1389-1286. DOI: https://doi.org/10.1016/j.comnet.2017.04.009. URL: http://www.sciencedirect.com/science/article/pii/S1389128617301391.

[5] H. Khedher et al. “Optimal and Cost Efficient Algorithm for Virtual CDN Orchestration”. Em: *2017 IEEE 42nd Conference on Local Computer Networks (LCN)*. 2017, pp. 61–69. DOI: 10.1109/LCN.2017.115.

[6] A. Ksentini, M. Bagaa e T. Taleb. “On Using SDN in 5G: The Controller Placement Problem”. Em: *2016 IEEE Global Communications Conference (GLOBECOM)*. 2016, pp. 1–6. DOI: 10.1109/GLOCOM.2016.7842066.

[7] C. C. Lin et al. “A Practical Model for Analyzing Push-Based Virtual Machine Live Migration”. Em: *2016 7th International Conference on Cloud Computing and Big Data (CCBD)*. 2016, pp. 347–352. DOI: 10.1109/CCBD.2016.074.

[8] J. Liu et al. “Migration-based Dynamic and Practical Virtual Streaming Agent Placement for Mobile Adaptive Live Streaming”. Em: *IEEE Transactions on Network and Service Management* (2017). ISSN: 1932-4537. DOI: 10.1109/TNSM.2017.2740432.

[9] Q. Sun et al. “Forecast-Assisted NFV Service Chain Deployment Based on Affiliation-Aware vNF Placement”. Em: *2016 IEEE Global Communications Conference (GLOBECOM)*. 2016, pp. 1–6. DOI: 10.1109/GLOCOM.2016.7841846.

[10] F. Tao et al. “BGM-BLA: A New Algorithm for Dynamic Migration of Virtual Machines in Cloud Computing”. Em: *IEEE Transactions on Services Computing* 9.6 (2016), pp. 910–925. ISSN: 1939-1374. DOI: 10.1109/TSC.2015.2416928.

[11] C. Wang et al. “A Switch Migration-Based Decision-Making Scheme for Balancing Load in SDN”. Em: *IEEE Access* 5 (2017), pp. 4537–4544. DOI: 10.1109/ACCESS.2017.2684188.

[12] J. Xia, Z. Cai e M. Xu. “Optimized Virtual Network Functions Migration for NFV”. Em: *2016 IEEE 22nd International Conference on Parallel and Distributed Systems (ICPADS)*. 2016, pp. 340–346. DOI: 10.1109/ICPADS.2016.0053.

[13] J. Xia et al. “Reasonably Migrating Virtual Machine in NFV-Featured Networks”. Em: *2016 IEEE International Conference on Computer and Information Technology (CIT)*. 2016, pp. 361–366. DOI: 10.1109/CIT.2016.96.