Performance Evaluation and Tuning of Virtual Infrastructure Managers for (Micro) Virtual Network Functions

Pier Luigi Ventre*, Claudio Pisa†, Stefano Salsano*†, Giuseppe Siracusano*‡, Florian Schmidt‡, Paolo Lungaroni§, Nicola Blefari-Melazzi*†
*University of Rome Tor Vergata, Italy, †CNIT, Italy, ‡NEC Laboratories Europe, Germany, §GARR, Italy

***Abstract*—Virtualized Network Functions (VNFs) are emerging as the keystone of 5G network architectures: flexibility, agility, fast instantiation times, consolidation, Commercial Off The Shelf (COTS) hardware support and significant cost savings are fundamental for meeting the requirements of the new generation of mobile networks. In this paper we deal with the management of the virtual computing resources for the execution of Micro VNFs. This functionality is performed by the Virtual Infrastructure Manager (VIM) in the NFV MANagement and Orchestration (MANO) reference architecture. We discuss the VIM instantiation process and propose a generic reference model, starting from the analysis of two Open Source VIMs, namely OpenStack Nova and Nomad. We implemented a tuned version of the VIMs with the specific goal of reducing the duration of the instantiation process. We realized a performance comparison of the two VIMs, both considering the plain and the tuned versions. The tuned VIMs and the performance evaluation tools that we have employed are provided openly and can be downloaded from our repository.**

***Index Terms*—Network Function Virtualization, Open Source, Virtual Infrastructure Manager, Virtual Network Function, Performance, Tuning, OpenStack, Nomad**

I. Introduction

Network Function Virtualization (NFV) is a new paradigm [1] able to drastically change the design of current networks. NFV aims at introducing software components in place of specialized network hardware. These software modules, called Virtual Network Functions (VNFs), execute the same functions of network appliances and run using virtual computing resources which are deployed on commodity hardware servers.

NFV can be used to support highly dynamic scenarios, in which the VNFs are instantiated "on the fly" following the service requests. This concept has been described as Superfluid Clouds [2] and is a key concept of the Superfluidity project [3]. In these scenarios, VNFs tend to become small and highly specialized *Micro-VNFs*, i.e., elementary and reusable network elements. Complex services can be build through the "chaining" of these Micro-VNFs.

Different virtualization approaches can be used to support VNFs: Virtual Machines (VMs), Tinified VMs, Unikernels and Containers (see [4] for a description of these concepts and an exhaustive comparison). In this work we focus on Unikernels for their suitability to Micro-VNFs: i) they offer very good performance in terms of low memory footprint and instantiation time; ii) they have very good isolation and security properties.

In particular, ClickOS [5] is a Xen-based Unikernel tailored for NFV appliances and able to provide highly efficient raw packet processing. It has a small footprint (around 5 MB when running), can be instantiated within around 30 milliseconds, processes up to 10Gb/s of traffic and does not need a disk to work. In addition, it benefits from the isolation provided by the Xen Hypervisor and the flexibility offered by the Click modular router.

Existing research efforts ([2] and [5]) demonstrate that it is possible to guarantee low latency instantiation times for Micro-VNFs in a specialized hypervisor. However, the same requirements have to be guaranteed by an NFV framework taken as a whole. In this work, we focus on the instantiation process of Micro-VNFs in an NFV framework. In particular, borrowing the ETSI NFV terminology (see Figure 1) we focus on the Virtual Infrastructure Manager (VIM), which controls and manages the virtual resources in the Network Functions Virtualization Infrastructure (NFVI). Existing Open Source solutions, designed for less specialized Cloud infrastructures, provide a solid base to build VIMs for NFVI. However, a deeper analysis (see Section II and Section VI) reveals that these do not support Unikernels and that there is room for tailoring the instantiation process to the NFV scenario to enhance its performance.

The main contributions of this paper are:

- the description of a general reference model of the VNF instantiation process, based on the analysis of two general-purpose VIMs: OpenStack Nova [6] and Nomad [7] by HashiCorp;
- modifications to these VIMs to instantiate Micro-VNFs based on ClickOS [5];
- realization of performance evaluation tools for the two VIMs to measure VM instantiation times;
- evaluation of ClickOS VMs instantiation times in OpenStack Nova and in Nomad;
- tuning of the VIMs to improve instantiation times;

We believe that the proposed model for VNF instantiation is applicable to other VIMs not considered here. Starting from the implemented components, experimenters/developers can build solutions for NFV frameworks and extend our tools to experiment on other VIMs.

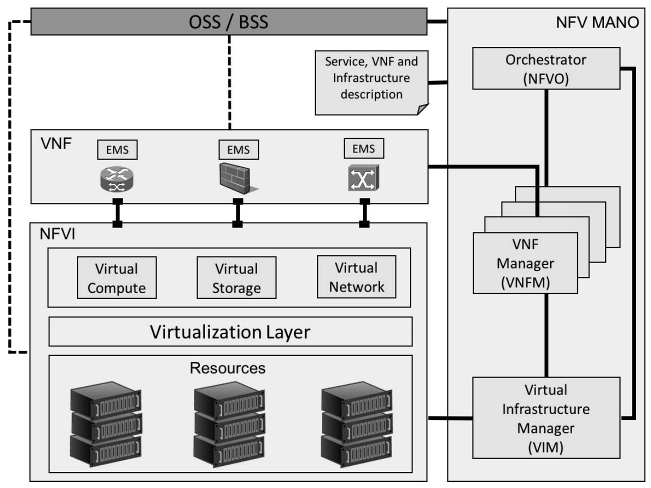


Fig. 1. NFV architecture

The rest of the paper is structured as follows: First, we will give some background of NFVs and VIMs in Section II. Section III describes the general model of the Micro-VNF instantiation process, and, starting from this, the corresponding models for the Nova and Nomad VIMs are derived. In Section IV, the modifications needed to boot Micro VNFs with Nova and Nomad are illustrated. Section V provides a performance evaluation of the Micro VNF instantiation process. Finally, Section VI reports on related work, while in Section VII we draw some conclusions and highlight the next steps.

II. Background on NFV and VIMs

The main building blocks of the ETSI NFV [1] and NFV MANO (MANagement and Orchestration) [8] architecture are represented in Figure 1. The Virtual Network Functions (VNFs) leverage on resources provided by the *NFV Infrastructure* (NFVI) for the execution of the network services. The NFVI layer is composed by COTS (Commercial Off The Shelf) hardware and by the Hypervisor software which implements the *Virtualization layer* abstracting the underlying hardware resources. The NFV MANO components represented in the right part of Figure 1 are responsible for the management of the physical infrastructure and the management and orchestration of the VNFs.

In particular, the NFVO (NFV Orchestrator) has the overall view of the services and the resources, it interacts with the VNF Managers (VNFMs) that manage the VNF life-cycle and deals with global resources management in NFVIs interacting with the VIMs. The VIMs interact with the infrastructure layer in order to manage and orchestrate the virtual resources which are required for the execution of the VNFs. In this work, we consider two general purpose VIMs: OpenStack Nova and HashiCorp's Nomad. Note that we refer to Nomad as a *VIM* while it is also commonly referred to as an *orchestrator*. We do it to be consistent with the ETSI NFV architectural model that separates the Orchestrator (NFVO) from the VIM (Figure 1).

OpenStack is a Cloud platform designed to manage large-scale computing resources in a single administrative domain. OpenStack is composed of different sub-projects. Among them, Nova provides the functionality for orchestrating and managing the computing resources. Its architecture envisages a single Nova node and a number of compute nodes. The Nova node schedules the computing tasks and manages the life-cycle of the virtual computing resources, while the compute nodes run locally in the Infrastructure nodes and interact with the local virtualization managers (Hypervisors). Among the other subprojects in OpenStack, the most important are: Keystone, the identity management service; Glance, the image store; Cinder, the storage service; Neutron, the *networking as a service* component; Horizon, the official dashboard (GUI) of the project.

Nomad by HashiCorp is a minimalistic cluster manager and job scheduler, which has been designed for micro services and batch processing workloads. Once compiled, it is a self-contained executable which provides in a small binary all the functionality of a resource manager and of a scheduler. Nomad can work both in Multi-Datacenter and Multi-Region scenarios. In the Nomad architecture, there are two types of nodes: Servers and Clients. The Server takes care of scheduling and all the Server nodes participate in scheduling decisions. Nomad Clients are the resource managers and locally run the jobs submitted by the Servers. Jobs are the unit of work in Nomad; they are composed of one or more task groups, which are themselves collections of tasks.

There are important differences between OpenStack Nova and Nomad. OpenStack is a complete Cloud suite composed of 9 core projects and a number of side projects; it provides a large set of functions that are essential for managing a Cloud infrastructure but which could add unnecessary overhead when employed in the NFV context. On the other hand, Nomad is packed in a single executable with a small footprint (about 30 MB). It is focused on a minimal complete functional set and thus only contains what is strictly necessary to schedule the virtual computing resources and to instantiate them in an Infrastructure node.

A common characteristic of the Nova and Nomad VIMs is that they do not focus on a specific virtualization technology, but provide an extensible framework to support different types of virtual computing resources. Other projects, such as Kubernetes [9], are designed for a specific virtualization technology (in this case *containers*), and modifying them to support the instantiation of Unikernels or VMs would require massive changes.

III. Modelling Approach

The proposed general model of the VM instantiation process is shown in Figure 2. We decompose the operations among the *VIM core components*, the *VIM local components* and the *Compute resource/hypervisor*. The VIM core components are responsible for receiving the VNF instantiation requests (i.e., the *initial request* in Figure 2) and for choosing the resources to use, i.e., the scheduling. This decision is translated into a set

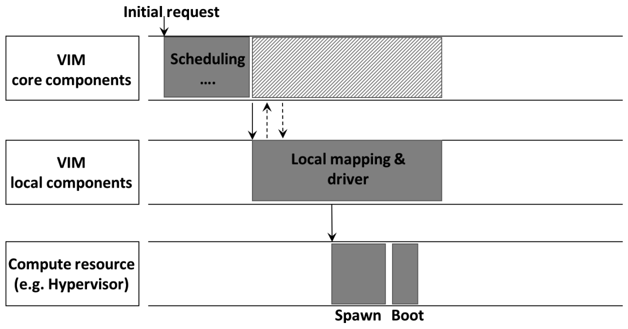
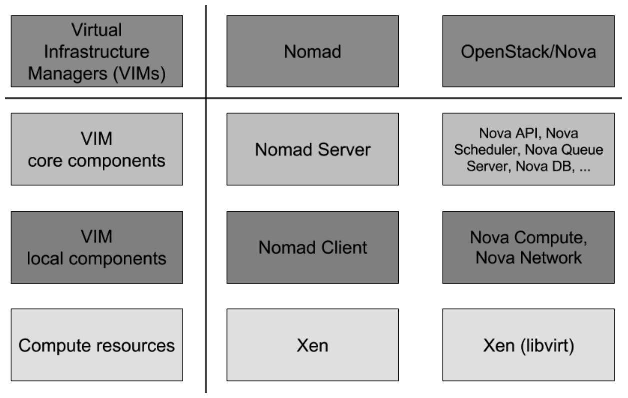


Fig. 2. VIM instantiation general model



Virtual Infrastructure Managers (VIMs)
Nomad
OpenStack/Nova
VIM core components
Nomad Server
Nova API, Nova Scheduler, Nova Queue Server, Nova DB, ...
VIM local components
Nomad Client
Nova Compute, Nova Network
Compute resources
Xen
Xen (libvirt)

Fig. 3. Mapping of the reference model to the considered VIMs

of requests which are sent to the VIM local components. These are located near the resources and are responsible for enforcing the decisions of the VIM core components by mapping the received requests to the corresponding hypervisor technology API calls. These APIs are typically wrapped by a *driver*, which is responsible for instructing the Compute resource to instantiate and boot the requested VNFs.

Figure 3 shows how the Nova and Nomad components can be mapped to the proposed reference model. In the next subsections, we provide a detailed analysis of the operations of the two VIMs. As for the Compute resource/hypervisor, in this work we focus on Xen [10], an open-source hypervisor commonly used as resource manager in production clouds. Xen can be configured to use different *toolstacks* (tools to manage guests creation, destruction and configuration).

A. Modelling OpenStack Nova

Figure 4 shows the scheduling and instantiation process for OpenStack Nova. The requests are submitted to the Nova API using the HTTP protocol (REST API). The Nova API component manages the *initial requests* and stores them in the Queue Server. At this point, an authentication phase towards *Keystone* is required. The next step is the retrieval of the image from *Glance*, which is required for the creation of virtual resources. At the completion of this step, the Nova Scheduler is involved: this component performs scheduling tasks by taking the requests from the Queue Server, deciding the Compute nodes where the guests should be deployed and sending back its decision to the Nova API (passing through the Queue Server). The components described so far are mapped to the *VIM core components*, in our model. After receiving the scheduling decision from the Nova Scheduler, the Nova API contacts the Nova Compute node. This component manages the interaction with the specific hypervisors using the proper *toolstack* and can be mapped (along with Nova Network) to the *VIM local components*. The VM instantiation phase can be divided in two sub-steps: Network creation and Spawning. Once this task is finished, Nova Compute sends all the necessary information to *libvirt*, which manages the spawning process instructing the Xen hypervisor to boot the virtual machine. When the completion of the boot process is confirmed, Nova Compute sends a notification and the Nova API confirms the availability of the new VM. At this point the

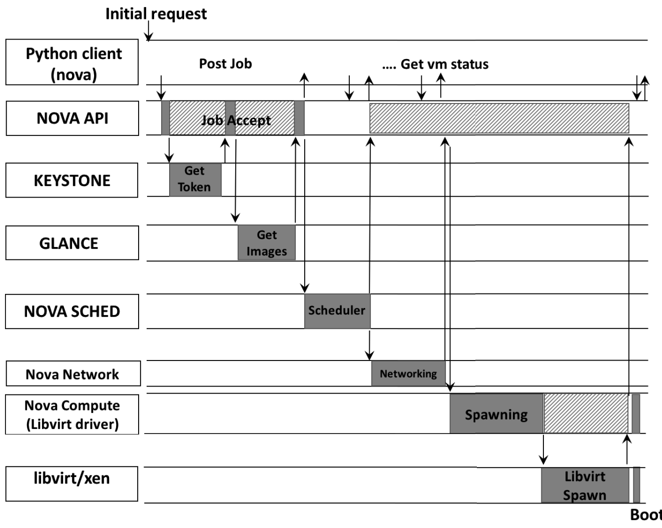


Fig. 4. VIM instantiation model for OpenStack Nova

machine is ready and started. The above description, reflected in Figure 4, is a simplified view of the actual process: for sake of clarity many details have been omitted. For example, the messages exchanged between the components traverse the messaging system (Nova Queue Server, which is not shown), and at each step the system state is serialized in the Nova DB (not shown).

B. Modelling Nomad

The scheduling and instantiation process for Nomad is shown in Figure 5. According to our model (cf. Figure 3), the *Nomad Server* is mapped to the *VIM core components*. It receives the requests for the instantiation of VMs (jobs) through the REST API. Once the job has been accepted and validated, the Server takes the scheduling decision and selects which Nomad Client node to run the VM on. The Server contacts the Client sending an array of job IDs. As response the Client provides a subset of IDs which are the ones that will likely be executed in this transaction. The Server acknowledges the IDs and the Client executes these jobs. The Nomad Client is mapped to the *VIM local components* and interacts with *compute resources/hypervisors*. In Xen, this

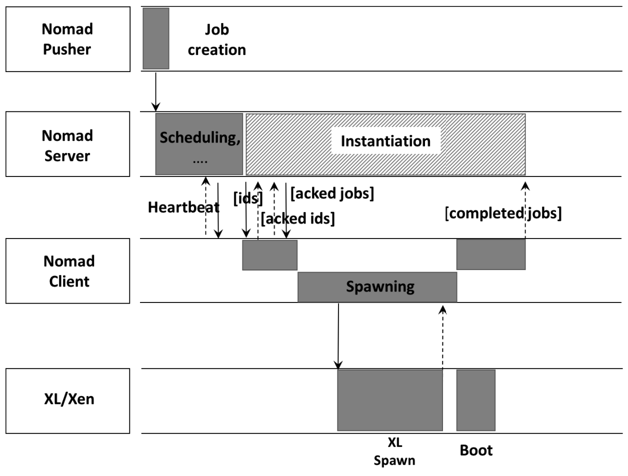


Fig. 5. VIM instantiation model for Nomad

interaction is done via a user-level *toolstack*, of which there are several choices. We used *XL* (see section IV), the default Xen toolstack, to interface with the local Xen hypervisor. *XL* provides a command line interface for guest creation and management. The Client executes these jobs loading the Nomad Xen driver, which takes care of the job execution interacting with the *XL* toolstack. The instantiation process takes place and once completed the Client notifies the Server about its conclusion. Meanwhile the boot process of the VM starts and continues asynchronously with respect to the Nomad Client.

IV. VIM Modifications to boot Micro-VNFs

We designed and implemented some modifications in the two VIMs in order to be able to instantiate Micro-VNFs based on ClickOS using Nova and Nomad.

The main reasons for these adaptations lie in the peculiarities of the ClickOS Unikernel Virtual Machines compared to regular VMs. A regular VM can boot its OS from an image or a disk snapshot that can be read from an associated *block device* (disk). The host hypervisor instructs the VM to run the boot loader that reads the kernel image from the block device. On the other hand, we are interested in ClickOS based Micro-VNFs, provided as a tiny self-contained kernel, without a block device. These VMs need to boot from a so-called *diskless image*. When instantiating such VMs, the host hypervisor reads the kernel image from a file or a repository and directly injects it in the VM memory.

For OpenStack, we enabled the boot of diskless images targeting only one component (Nova Compute) and a specific toolstack, i.e. *Libvirt*. We selected this toolstack because it supports *libxl*, which is the default Xen toolstack API. We first implemented the minimal set of changes to the *Libvirt* driver as needed to instantiate the ClickOS VMs. In particular, we modified the XML description of the guest domain provided by the driver, changing the XML description on the fly before the creation of the domain. We did not patch the OpenStack image store (*Glance*) because a mainstream patch was out of the scope of our work. We resorted to a simple hack to trigger the execution of our code needed to boot ClickOS VMs, adding a new image to the store with a specific image name. When this specific name is selected for instantiation, our patches to the Nova compute Libvirt driver are executed.

For what concerns Nomad, this VIM supports different types of workload through the driver framework. In particular the following workload types have built-in support: Docker, Java VMs and QEMU/KVM. Starting from the QEMU driver, we developed a new Nomad driver for Xen, called *XenDriver*. The new driver communicates with the *XL* Xen toolstack and it is also able to instantiate a ClickOS VM.

By extending the driver framework we automatically added the support in Nomad for Xen-type jobs without the need to also modify the Server. The drivers are loaded at boot time and the supported drivers are communicated to the Servers in the context of the Client's registration. Therefore, using the standard Nomad CLI or a HTTP client we can create Xen jobs and submit them to the Server.

We define as *stock* the VIM versions that only include the modifications required to boot the ClickOS VMs. We analysed the time spent by the *stock* VIMs during the different phases of the driver operations. Then, taking into account the results, we proceeded with an optimization trying to remove those functions/operations not directly needed to instantiate Micro-VNFs. We define as *tuned* these optimized versions.

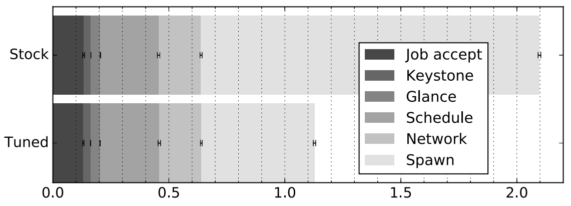
In Nomad it was relatively easy to introduce the support for booting from diskless images thanks to its simple architecture and to the flexibility of the nomad driver framework. It was more difficult to understand how to modify OpenStack, because of its larger codebase, supporting several types of hypervisors, each of these with more than one toolstack.

Our patches to OpenStack Nova and Nomad Xen driver are available at [11].

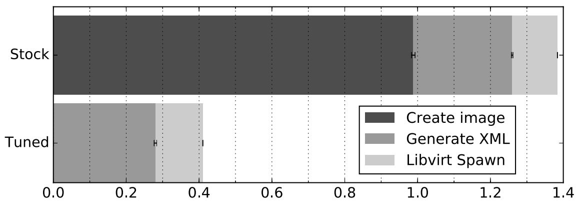
V. Experimental Results and Performance Tuning

In order to evaluate the VIM performances in the VM scheduling and instantiation phase, we combined different sources of information. As a first step, we analyzed the message exchanges in order to obtain coarse information about the beginning and the end of the different phases of the VM instantiation process. The analysis of messages is a convenient approach as it does not require modification of the source code. For this analysis, we developed a *VIM Message Analyzer* tool with a Python script and the Scapy [12] library. The VIM Message Analyzer (available at [11]) is capable of analyzing Nova and Nomad message exchanges.

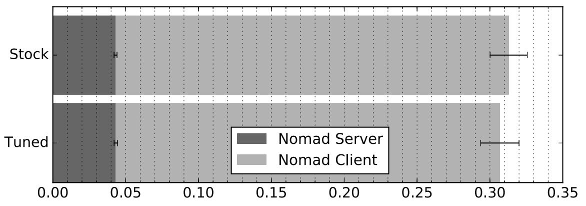
For a more detailed breakdown of the timings for specific components or phases, we inserted timestamp logging instructions into the code of the Nomad Client and Nova Compute nodes. We generated the workload for OpenStack using Rally [13], a well known benchmarking tool. For the generation of the the Nomad workload, instead, we developed the *Nomad Pusher* tool. It is an utility written in the GO language, which can be employed to programmatically submit jobs to the Nomad Server.



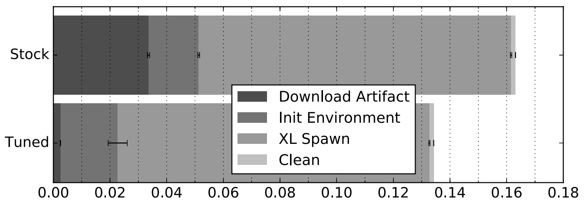
(a) ClickOS instantiation time breakdown on OpenStack



(b) ClickOS spawning time breakdown on Nova-compute



(c) ClickOS instantiation time breakdown on Nomad



(d) ClickOS spawning time breakdown on Nomad

Fig. 6. VIMs micro-NFV (ClickOS) instantiation and spawn time[s] breakdown

We executed experiments to evaluate the performance of the considered VIMs. We present here two main results: i) the total time needed to instantiate a ClickOS VM (representing a Micro-VNF); ii) the timing breakdown of the Spawning process in Nova and of the Driver execution in Nomad. The first result is based on the VIM Message Analyzer we developed. The timing breakdown was obtained with the approach of inserting timestamp loggers into the source code of the VIMs. All results were obtained by executing a test of 100 replicated runs, in unloaded conditions. Error bars in the figures denote the 95% confidence intervals of the results. In each run we requested the VIM to instantiate a new ClickOS VM. The VM was deleted before the start of the next run. Our experimental set-up comprised two hosts with an Intel Xeon 3.40GHz quad-core CPU and 16GB of RAM. One host (hereafter referred to as HostC) was used for the VIM core components, the other host (HostL) for the VIM Local components and the Compute resource. We used Debian 8.3 operating systems with Xen-enabled v3.16.7 Linux kernels. Both hosts were equipped with two network interfaces at 10 Gb/s: one interface was used as the management interface and the other one for the direct interconnection of the host (data plane network). In order to emulate a third host running OpenStack Rally and Nomad Pusher, we created a separated network namespace using the *iproute2* suite in the HostC, then we interconnected this namespace to the data plane network.

A. OpenStack Nova Experimental Results

For the OpenStack setup, we run Keystone, Glance, Nova orchestrator, Horizon, and the other components in HostC, while we run Nova Compute and Nova Network in HostL. We use the modifications presented in Section IV and the *Libvirt/libxl* support in OpenStack in order to boot ClickOS Xen machines.

With reference to Figure 4, we report in Figure 6a the measurements of the instantiation process in OpenStack, separated for each component. The upper horizontal bar (*Stock*) refers to the OpenStack version that only includes the modifications to boot the ClickOS VMs. The experiment reports a total time exceeding two seconds which is not adequate for highly dynamic NFV scenarios. Analyzing the timing of the execution of the single components, we note that most of the time is spent during the spawning phase while the other components account for around 0.5 seconds.

In Figure 6b (upper horizontal bar), we show the details of the spawning phase which is split in tree phases: 1) *Create image*, 2) *Generate XML* and 3) *Libvirt spawn*. The first two are executed by the Nova Libvirt driver and the last one is executed by the underlying Libvirt layer. The Nova Libvirt driver also executes some network configuration steps which are not shown, as they happen in parallel with the *Create image* phase, which always terminates after the network configuration has finished. By analyzing the timing of the *stock* version, we can see that the *Create image* is the slowest step with a duration of about 1 second. This step includes operations such as the creation of log files and the creation of the folders to store Glance images. The original OS image (the one retrieved from Glance) is re-sized in order to meet user requirements (the so called *flavors* in OpenStack's jargon). Moreover, if it is required, the swap memory or the ephemeral storage are created and finally some configuration parameters are injected into the image (e.g. SSH key-pair, network interface configuration). Considering these results, we focused on this step in our tuning of the OpenStack performance (see V-C).

The *Generate XML* and *Libvirt spawn* steps introduce a total delay of 0.4 seconds, but optimizing these steps is non-trivial as none of the performed operations can be skipped or significantly reduced. During the *Generate XML* step, the configuration options of the guest domain are retrieved and then used to build the guest domain description (the XML file given as input to *Libvirt*). For instance, options such as the number of CPUs and CPU pinning are inserted in the XML file. Once this step is over, the libxl API is invoked in order to boot the VM. When the API returns, the VM is considered spawned, terminating the instantiation process. In this test we are not considering the whole boot time of the VM, as this is independent from the VIM operations, and thus out of the scope of this work. The considered spawning time measures only the time needed to create the Xen guest domain.

B. Nomad Experimental Results

According to the Nomad architecture, the minimal deployment comprises two nodes. Therefore, we deployed a Nomad Server, which performs the scheduling tasks, on HostC and a Nomad Client, which is responsible for the instantiation of virtual machines, on HostL. In Section III, we identified two major steps in the breakdown of the instantiation process: Scheduling and Instantiation. The upper horizontal bar in Figure 6c reports the results of the performance evaluation for the stock Nomad. The total instantiation time is much lower than the one obtained for OpenStack. This result is not surprising: as discussed in Section II, Nomad is a minimalistic VIM providing only what is strictly needed to schedule the execution of virtual resources and to instantiate them. Looking at the details, the scheduling process is very light-weight, with a total run time of about 50 ms. The biggest component in the instantiation time is the spawning process which is executed by the XenDriver. Diving in the driver operations, we identified 4 major steps: *Download artifact*, *Init Environment*, *Spawn*, and *Clean*, as reported in in Figure 6d. In the first step, Nomad tries to download the artifacts specified by the job. For a Xen job, the Client is required to download the configuration file describing the guest and the image to load. This part adds a delay of about 40 ms and can be optimized or entirely skipped. *Init Environment* and *Clean* introduce a low delay (around 20 ms) and are interrelated: the former initializes data structures, creates log files and folders for the *Command executor*, the latter cleans up the data structures once the command execution is finished. The *XL spawn* is the step which takes longer but by studying the source code we found no room to implement further optimizations: indeed the total spawning measured time is around 100 ms. Considering a light overhead introduced by the *Command executor* the time is very similar to the what we obtain by running directly the *XL* toolstack. The overall duration of the spawning phase is 160 ms, lower that the 280 ms for the instantiation phase reported in Figure 6c. This is due to the notification mechanism from the client towards the server. It uses a lazy approach for communicating the end of the scheduled jobs: when the message is ready to be sent, the client waits for a timer expiration to attempt to aggregate more notifications in a single message. This means that the instantiation time with Nomad is actually shorter than the one shown in Figure 6c.

C. Performance Tuning Results

The performance measurements for Nova results reported in Section V-A show that most of the instantiation time is spent on the *Create image* operation. Thus we focused on optimizing this phase. The “tuned” horizontal bar in Figure 6a reports the obtained result. The reduction of the instantiation time highlights that the advantages of using Unikernels with respect to a full fledged OS are not only the shorter boot times: specific VIM optimizations can be performed due to their characteristics. We are using a tiny diskless VM, which means we can skip most of the actions performed during the image creation step. For example, we do not need to resize the image in order to match the flavor disk size. Moreover, unikernels provide only the functionality needed to implement a specific VNF, so it is unlikely to have an SSH server running on it, hence SSH key-pairs configuration is not needed. Furthermore, if a Micro-VNF does not require a full IP stack, the injection of the network configuration in the image is useless. Indeed, some Micro-VNFs need only to have configured the bridge on which it has to be attached. Implementing these optimizations we are able to reduce the spawning time down to 0.4 s (Figure 6b, bottom bar) obtaining a reduction of about 70% compared to the stock version of OpenStack. Looking at the overall instantiation time, the relative reduction (Figure 6a, bottom bar) is about 45%, down to 1.15 s from 2.1 s for the stock OpenStack.

Regarding Nomad, we already observed that the instantiation time are smaller, in the order of 0.3 s (or 0.2 s if we do not consider the notification delay from Nomad Client to Nomad server). This can be explained by the following: i) the scheduler in the Nomad Server is very lightweight compared to OpenStack scheduler and its functionality is much simpler; ii) in the client we have implemented from scratch a new Nomad driver for Xen (cf. Section IV) and hence included only code in the driver which is strictly necessary to interact with *XL*. Starting from our initial implementation, we introduced further improvements streamlining the *Download Artifact* step of the XenDriver, assuming that the images of the Micro VNFs can be stored locally in the Nomad Client. Pursuing this approach we can reduce the Driver operation of about 30 ms (see Figure 6d).

VI. Related Work

VIM performance evaluation is a topic addressed also by other works. In [14], [15], [16] authors compare the performance of OpenStack versus other projects (CloudStack, OpenNebula, Eucaliptus). However, the performances of VIMs are analyzed in terms of time needed to instantiate fully fledged VMs and they are mainly focused on mere benchmarking without a deep analysis of the tasks performed during the instantiation. On the other hand in [17], [18], authors consider only OpenStack and focus on particular aspects such as networking components rather than the scheduling.

Other works (such as [5], [19], [2]) focus on the performance of ClickOS and of the NFVI. They demonstrate that it is possible to guarantee low latency instantiation times for Micro-VNFs and the suitability of ClickOS for the NFV use case. Instead, our work is focused on the analysis of the performance of VIMs and of their suitability in the NFV frameworks. In [20], [21] the authors describe solutions covering the whole

NFV framework and for the implementation of ETSI MANO specifications. Among these, solutions such as CloudBand, CloudNFV and OpenNFV are proprietary and there are not enough details available to include them in our analysis, while OPNFV defines a general framework for NFV, and, as for the VIM component, it employs OpenStack. However these works do not cover the performances of the practical implementations of the NFV framework.

In this context, OpenMANO is an open source projects providing a practical implementation of the NFV MANO architecture, according to the ETSI standardization. OpenMANO has been recently incorporated by Open Source MANO [22], an ETSI project that aims at developing an Open Source NFV Management and Orchestration (MANO) software stack aligned with ETSI NFV. TeNOR is another open source project which implements the NFV MANO framework and in [23] also reported a performance evaluation. However, their evaluation only covers their own system and at a much higher level, and thus is not comparable with our work.

Openvim is the implementation of an NFV Virtualized Infrastructure Manager initially considered in the context of the OpenMANO project. It is a lightweight Python program which can operate efficiently, based only on *SSH* and *Libvirt* to interface with the NFVI. In the near future, we plan to extend our analysis to consider also Openvim. Note that OpenMANO is now working to support different VIMs, and has started supporting OpenStack in addition to Openvim.

VII. Conclusions

We described a general reference model of the VNF and Micro-VNF instantiation process, and detailed this model for two Virtual Infrastructure Managers (VIMs), OpenStack Nova and Nomad. We provided measurements on the performance of the instantiation process in an experimental setup, using ad-hoc developed profiling tools. The tools we developed are available as open source. Starting from an analysis of the measurements results, we optimized the instantiation times of Micro-VNFs by modifying the source code of the VIMs.

For the performance measurements we considered the basic case of an unloaded system and of a very simple physical deployment. We measured the instantiation time of a VM considering a single request (no other background requests in parallel), assuming that the requested VM is the first one to be allocated/scheduled (the database of allocated VMs is empty) and that only one target node is available for deploying the VM. An important direction for future work is to extend the analysis considering the impact of system load on the performance. We plan to measure the average instantiation times considering batches of incoming requests with given rates (requests/s) and arrival patterns, as well as the performance impact of the number of already allocated VMs and of the number of target nodes to be deployed.

Another potential future direction of our work would be to further improve the performance of the considered VIMs through the optimization of other components which have not been considered here (e.g., trying to replace the lazy notification mechanism of Nomad with a reactive approach). Finally, we plan to extend the analysis to other VIMs, in particular, as already mentioned in Section VI, considering Openvim from the OpenMANO project.

Acknowledgment

This paper has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 671566 ("Superfluidity"). This paper reflects only the authors views and the European Commission is not responsible for any use that may be made of the information it contains.

References

[1] "ETSI Network Function Virtualization." [Online]. Available: http://www.etsi.org/technologies-clusters/technologies/nfv

[2] F. Manco *et al.*, "The case for the superfluid cloud," in *7th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 15)*, 2015.

[3] "Superfluidity project." [Online]. Available: http://superfluidity.eu

[4] F. Huici *et al.*, "Vms, unikernels and containers: Experiences on the performance of virtualization technologies." [Online]. Available: https://www.ietf.org/proceedings/95/slides/slides-95-nfvrg-2.pdf

[5] J. Martins *et al.*, "Clickos and the art of network function virtualization," in *Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation*. USENIX Association, 2014, pp. 459–473.

[6] "OpenStack." [Online]. Available: https://www.openstack.org

[7] "Nomad project." [Online]. Available: https://www.nomadproject.io

[8] ETSI. Group for NFV, "Network Functions Virtualisation (NFV); Management and Orchestration; Functional requirements specification," 2016.

[9] "Kubernetes." [Online]. Available: http://kubernetes.io

[10] "Xen project." [Online]. Available: http://www.xenproject.org

[11] "VIM tuning and evaluation tools." [Online]. Available: https://github.com/netgroup/vim-tuning-and-eval-tools

[12] "Scapy." [Online]. Available: http://www.secdev.org/projects/scapy/

[13] "Openstack rally." [Online]. Available: https://wiki.openstack.org/wiki/Rally

[14] A. Paradowski *et al.*, "Benchmarking the performance of openstack and cloudstack," in *2014 IEEE 17th International Symposium on Object/Component-Oriented Real-Time Distributed Computing*. IEEE, 2014, pp. 405–412.

[15] D. Steinmetz *et al.*, "Cloud computing performance benchmarking and virtual machine launch time," SIGITE12, pp. 89–90, 2012.

[16] E. Caron *et al.*, "Comparison on openstack and opennebula performance to improve multi-cloud architecture on cosmological simulation use case," in *Research Report RR-8421*. INRIA, 2013, p. 23.

[17] G. Callegati *et al.*, "Performance of network virtualization in cloud computing infrastructures: The openstack case," in *Cloud Networking (CloudNet), 2014 IEEE 3rd International Conference on*. IEEE, 2014, pp. 132–137.

[18] Litvinsk *et al.*, "Experimental evaluation of openstack compute scheduler," *Procedia Computer Science*, vol. 19, pp. 116–123, 2013.

[19] F. Manco *et al.*, "Towards the super fluid cloud," in *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 4. ACM, 2014, pp. 355–356.

[20] R.Mijumbi *et al.*, "Management and orchestration challenges in network function virtualization," IEEE, 2016.

[21] R. Mijumbi *et al.*, "Network function virtualization: State-of-the-art and research challenges," IEEE, 2015.

[22] "ETSI open source MANO." [Online]. Available: https://osm.etsi.org/

[23] J. Riera *et al.*, "Tenor: Steps towards an orchestration platform for multi-pop nfv deployment," in *NetSoft Conference and Workshops (NetSoft), 2016 IEEE*. IEEE, 2016, pp. 243–250.