A Linux-based Virtualized Solution Providing Computing Quality of Service to SDN-NFV Telecommunication Applications[*]

Carlo Vitucci
Huawei Technologies Co., Ltd
Stockholm, Sweden
vitucci.carlo@gmail.com

Juri Lelli
Scuola Superiore Sant'Anna
Pisa, Italy
juri.lelli@sssup.it

Andrea Parri
Scuola Superiore Sant'Anna
Pisa, Italy
andrea.parri@sssup.it

Mauro Marinoni
Scuola Superiore Sant'Anna
Pisa, Italy
mauro.marinoni@sssup.it

Abstract

Currently, Software Defined Networking (SDN) and Network Function Virtualization (NFV) are gaining growing interest due to the increased flexibility they could bring in the design of modern telecommunication systems. The nature of SDN-NFV allows a wide range of applications to run concurrently on the same hardware resources that can be managed by virtualization technologies. However, these applications present heterogeneous computation requirements in terms of resource demand and timing constraints. Thus it is necessary to recognize different classes of virtualized applications and to provide specific execution time guarantees for each of them.

In this paper we define a reference model characterizing the computation requirement of SDN-NFV virtualized applications. We then exploit resource reservation capabilities provided by SCHED_DEADLINE, combined with Direct Interrupt Delivery and Completion, in order to provide specific service guarantees for each of these models. Finally, we conclude with an experimental evaluation which shows the effectiveness of our approach.

Results show that SCHED_DEADLINE is able to guarantee the required level of temporal isolation and higher Quality of Service. Moreover, the overheads introduced by the virtualization infrastructure are drastically reduced by the adoption of Direct Interrupt Delivery and Completion. The results obtained in this work have a big impact in the design of telecommunication systems: they demonstrate the availability of virtualized Linux-based solutions, which are able to fulfill the tight requirements of these applications.

[*]This work has been partially supported by the 7th Framework Programme JUNIPER (FP7-ICT-2011.4.4) project, founded by the European Community under grant agreement n. 318763.

1 Introduction

Over the last years, the telecom world has been strongly changed under the pressure of new technologies like smartphones. Services such as voice and short messages, that were extremely profitable for telecom operators in the last years, fail nowadays to ensure enough profits due to the increased competitiveness among operators and the shift toward data traffic. Moreover, the high-profit traffic volume is continuously decreasing, while the data traffic amount is significantly on the rise, due to the effect of new technologies (e.g., aLTE) and the new connectivity demands (e.g., M2M). Recent studies [1] have presented the hypothesis that the current business models followed by telecom operators will become unprofitable within three years. Moreover, operators are reluctant to make the investments needed to meet the growing demand of data traffic. Summarizing, solutions able to ensure both a response to the growth of data traffic and the possibility of new business models for operators are needed.

The approach that is showing more potentiality and is leading to a new approach in systems design is called Mobile Cloud Computer [3] and is shown in Figure 1. It is based on two concepts: Software Defined Networking (SDN) and Network Functions Virtualization (NFV). SDN [4] is a new network paradigm where network control is decoupled from forwarding and is directly programmable. This shift of controls, previously pinned to single devices, enables the underlying infrastructure to be abstracted for applications services and managed as a virtual entity. NFV [5] is an approach to virtualize network functions beforehand performed by proprietary dedicated appliances. The target is a cost reduction for network devices such as routers, firewalls and security appliances running them all in a common platform that would host the needed environments.

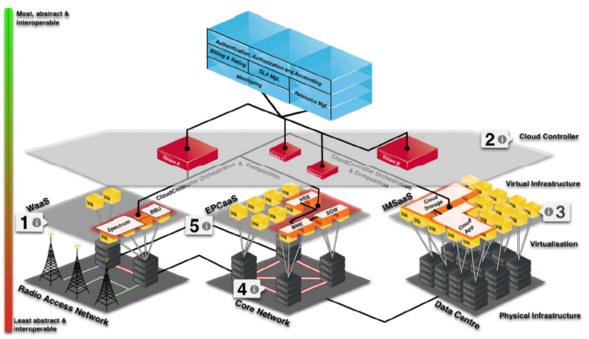

FIGURE 1: *Mobile Cloud Computing Architecture*

The efficiency of this new system approach in terms of services elasticity, infrastructures costs and bandwidth capacity will depend on how close to the user it will be implemented [2]. In the scenario shown in Figure 1, the approach is applied up to the nodes handling the radio access which are called the *edge of the network*.

In the description of a use case on Virtualization of Mobile base stations [6] it is said that "Virtualization of mobile base station leveraged IT virtualization technology to realize at least a part of RAN (Radio Access Network) nodes onto standard IT servers, storages and switches". SDN-NFV is heavily dependent on virtualization technologies allowing any application or service to be actually implemented as Virtual Machine (VM) under the supervision of a centralized common controller. While the deployment of SDN-NFV solutions for data centers and core networks is a straightforward application of established virtualization technologies, the usage of such technique at the edge of the network is more challenging. In fact, providing telecom functions as service in edge nodes brings an important issue: the functions of edge telecom networks are characterized by stringent real-time constraints, intolerant with virtualization overheads regarding response times.

To improve the predictability of activities executing in such an heterogeneous environment it is necessary to reserve resource bandwidth and isolate the execution of different applications. In particular, for robustness, security and safety issues, it is necessary to isolate and protect the temporal behaviour of different tasks. Resource Reservations [7] resulted being a suited technique to achieve temporal isolation and time-constrained execution in heterogeneous systems. Resource reservation techniques had originally been proposed for the execution of independent tasks on single processor systems. Recently, they were extended to deal with hierarchical scheduling systems [8, 9] and multi-core platforms [10, 11].

This paper presents an approach based on the integration of Resource Reservations and interrupts management techniques to improve the deterministic behaviour of virtualization environments in order to support SDN-NVF applications deployment. The focus is on the presentation of the general framework, while the determination of the optimal allocations is beyond the scope of this work.

The rest of the paper is organized as follows: Section 2 describes the different techniques, in particular: Section 2.1 presents how to reduce interrupts latency and Section 2.2 shows the use of resource reservation to provide temporal isolation together with higher utilization of computational resources. Section 3 presents some experimental results on resource reservation and finally Section 4 concludes the work, providing ideas about future steps of the research.

2 Proposed Approach

In this section are described the techniques and mechanisms used to provide the level of services needed by SDN-NFV applications. We start categorizing the virtual machines by their requirement in terms of computational resources utilization. To provide a simple classification of such platforms, we divide them by means of computing demand C over a periodic interval P, with the constraint of terminating the execution of each request before a deadline D expires. The model is shown in Figure 2.

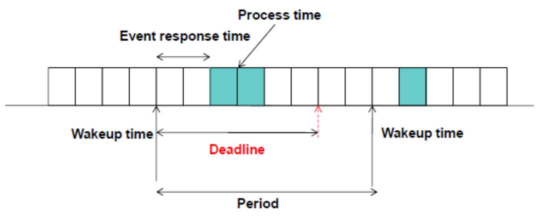

FIGURE 2: *Time parameters of a task*

To simplify the approach, from now on we consider deadline and period to be the same ($D = P$), thus performing the characterization using only computation time and period. This leads to obtain three classes of virtual machines.

The first class describes virtual machines for applications that require deterministic behaviour. Deterministic means that the reaction time to external stimuli should only depend on the status of the guest virtual machine. So, such class of virtual machines cannot share its resources with other VMs. Moreover, those virtual machines usually need the shortest possible period P to guarantee an interrupt latency time that is small enough. An example of this class is given by radio applications, such as UMTS and LTE. In this case, the application expects to receive a significantly high number of packets and it needs to react to each incoming packet within microseconds. For that reason, a C/P ratio equal to one is the only viable allocation, since such a value allocates a dedicated and isolated computing node to those virtual machines.

The second class is composed by the *real-time applications*. The definition of real-time application in the SDN-NVF domain could be misleading, since it doesn't map to a traditional hard or soft real-time behaviour. There is the demand to respect deadlines, but with a higher tolerance in terms of latency respect to the first class (i.e., the deterministic one). An example of this class is given by video or audio applications, where the constraint is to guarantee enough computation time for each frame.

The third class includes those virtual machines with no specific constraints in terms of interrupt latency time and deadlines that only require enough computation bandwidth to handle the assigned workload. An example of this class could be the control subsystem of a service chain.

The virtualization infrastructures managing the servers at the edge of the network should be able to handle all the three classes of virtual machines. Depending on the characterizations of the assigned VMs the requirements that must be guaranteed are:

- *Low interrupt latency time and deterministic behaviour.* Such a requirement is applicable to virtual machines in the first class, i.e. for those virtual machine where the service delay due to the virtualization must be negligible. The only way to obtain this goal is to assign the computing resource in an exclusive way to the VM;
- *Improved computing utilization.* Improving the management of appliances is one of the core motivations behind the SDN-NFV paradigm. A wise allocation of VMs allows to reduce power consumption and improves the overall system performances while reducing the drawbacks of vitualization. However, this metric is important for VMs in the second and third class, but cannot be applicable to the first class of VMs that are subdued to more stringent temporal constraints.
- *Strong temporal isolation.* Temporal isolation is fundamental for the success of the SDN-NVF model, because it guarantees that the available computational resources fulfill the agreement with the platform. For the first class of applications this requirement is an automatic byproduct of the exclusive allocation of a computation node. For the other two classes, combining this and the previous requirement needs enhanced virtualization infrastructures.

A possible allocation policy providing the needed resources to a set of applications is composed by the following sequence of phases:

- Assign each VMs to a class depending from its requirements;

- Statically allocate VMs with deterministic constraints (i.e., first class) to dedicated nodes to guarantee their strong temporal constraints;
- Deploy the others VMs to maximize the nodes utilizations.
- Select which enhancements to activate in each node to fulfill the constraints of the assigned VMs.

In the following paragraphs are described the different techniques that need to be included in the virtualization platforms in order to successfully execute all the assigned applications.

2.1 Lowering interrupt latency time

One of the main drawback of virtualization technologies is the increase of latency times in particular when dealing with interrupts. This is particularly important when serving applications characterized by stringent temporal constraints. In some cases, like those within the first class, it is not possible to share the core among different applications thus it is statically assigned. This reduces the flexibility provided by the virtualization, but allows to run strongly constrained applications together with less critical ones. However, other techniques are required to reduce the latency in order to guarantee the timing constraints imposed by the first class of applications. Some mechanisms can be used depending on the hardware platform and the application requirements:

- Use *Hardware-Assisted Virtualization* functionalities to reduce overheads. Processors manufacturers have been introducing hardware mechanisms to reduce the cost of some activities offloading the specific hardware [12, 13, 14];
- Prevent latencies induced by delays in the management of specific interrupts related to workload from less critical VMs and due to long interrupt-off or preemption-off region. A solution for symmetric multi-processor (SMP) systems is the *CPU shielding* [15] and *CPU affinity* [16], as they assign tasks and interrupt executions to predefined cores;
- Another issue that degrades performances of virtualized environments is the *Cache Pollution* that represents eviction of needed data by lower-priority activities. Reducing this effect is possible with the hardware management of Huge Pages [17] and nested page tables using Second Level Address Translation (SLAT) [18];
- Prevent unneeded preemptions of the VM removing the system tick in the CPU that are shielded [19];
- Drastically reduce the overhead of interrupt management using the support of Direct Interrupt Delivery and Completion.

The last technology is particularly important for virtual machines with notably restrictive timing constraints like those characterizing the first class of applications, thus we present it in details.

The Direct Interrupt Delivery and Completion has been investigated since 2012 by different hardware manufacturers. Two approaches have been proposed independently by Hitachi^{TM} [20] and IBM^{TM} [21]. The Direct Interrupt Delivery and Completion mechanism is also called ExitLess Interrupt (ELI) and IBM called its implementation ELVIS (Efficient and Scalable Virtio), that is currently used by IBM server solutions. Siemens^{TM}designed a new isolation manager, called Jailhouse, where direct interrupt delivery and completion functionality is available, but with the limitation that Jailhouse is not applicable for VM, at the moment.

When focusing on interrupt management in virtualized environments, the main source of overhead is directly connected to the high number of VM-exits, which mark the points at which transitions are made between the currently executing virtual machine and the hypervisor that has to perform system control activities. The sequence of phases is depicted in Figure 3. It shows that this procedure introduces extra execution time, thus consuming computational resources, reducing the performance, and increasing the interrupt latency time.

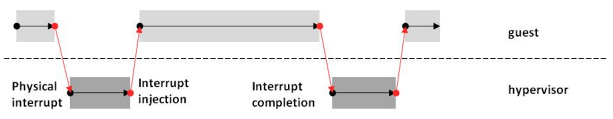

FIGURE 3: *Transitions during the execution of a VM-exit.*

The basic concept about the ELI hardware functionality is to use the latest Hardware-Assisted Virtualization features that provide two interrupt controllers in modern microprocessors: a global controller managing the whole chipset and a local for each core. The hardware allows to set which interrupts should be propagated into local controller. Using the support for virtual data paths, it is possible to have different interrupts per different virtual paths, thus allowing to connect one virtual path to a virtual machine with its dedicated interrupts. When the virtual machine is running, it already knows how to handle those interrupts and doesn't need to involve the hypervisor for the end of interrupt (EOI). The benefit of using Direct Interrupt Delivery and Completion is evident when a virtual machine is running alone in a dedicated core, since the VM will run 100% of time, without the need of hypervisor actions at all (under the proper isolation conditions), then presenting the same performances and overhead of the non-virtualized execution. Note that, this comparison doesn't consider some coding shortcuts that can be used while directly accessing low level hardware and are not possible in a virtualized environment.

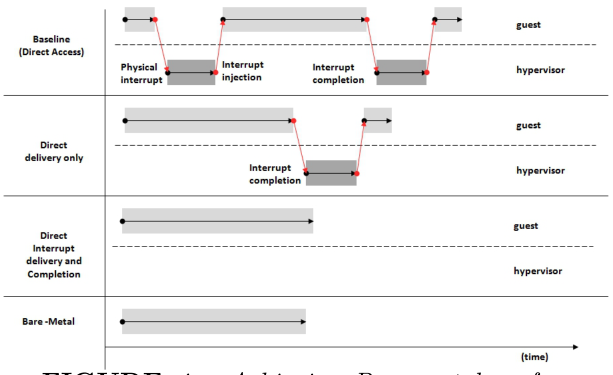

FIGURE 4: *Achieving Bare-metal performance for the VM through Direct Interrupt Delivery and Completion and full isolation.*

Moreover, Direct Interrupt Delivery and Completion brings benefits even for a core that is shared between multiple VMs, because the VM-exit won't happen at any packet, but only for packets from a virtual path that belongs to a no-running virtual machine. This reduces the number of VM-exit executions and increases performances, but not up to a level able to manage VMs belonging to the first class.

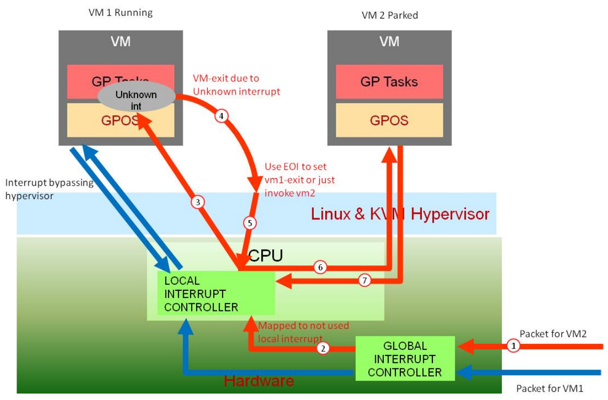
VM 1 Running
VM 2 Parked
VM
GP Tasks
Unknown int
GPOS
VM
GP Tasks
GPOS
VM-exit due to Unknown interrupt
Use EOI to set vm1-exit or just invoke vm2
Interrupt bypassing hypervisor
Linux & KVM Hypervisor
CPU
LOCAL INTERRUPT CONTROLLER
Mapped to not used local interrupt
GLOBAL INTERRUPT CONTROLLER
Packet for VM2
Packet for VM1
Hardware
1
2
3
4
5
6
7

FIGURE 5: *Direct Interrupt Delivery and completion when multiple VMs share the same core.*

The Direct Interrupt Delivery and Completion is an approach that requires support from the hardware platform, in particular are required hierarchical interrupt handling and the capability to distribute different interrupt per different virtual data path. In Intel processors they are available using x2APIC, Virtual Machine Device Queues (VMDq) and Single Root I/O Virtualization (SR-IOV) technologies [14]. Hierarchical interrupt handling is also available in modern AMD and ARM processors, while SR-IOV is present in most of the recent architectures based on the PCI-express bus.

2.2 Temporal isolation and increased utilization

This section presents the techniques used to provide temporal isolation among virtual machines. The isolation for the first class is implicit, since this class is based on the encapsulation and per-core pinning of the virtual machine. For the second and the third classes of VMs is needed a scheduling algorithm which is able to provide a level of isolation compatible with the applications requirements.

The chosen solution is a resource reservation mechanism based on the *Constant Bandwidth Server* (CBS) [25] that implements a budget aware extension of *Earliest Deadline First* (EDF) scheduler [23] to provide a bandwidth reservation strategy. The concept of the CBS is to ensure temporal isolation of tasks in terms of meeting deadlines as if each task is running on an independent processor. Using this scheduling policy, the quantity of computing resource assigned and used by a virtual machine is not going to change due to other VMs actions even in case of malicious behaviour.

Using the mechanisms provided by EDF, the CBS approach assigns to each task a server that is characterized by a period P and a budget Q of computation time on the core. The bandwidth utilization U is computed as Q/P and represents the fraction of CPU time that is reserved by the scheduler for each period. The advantages of using a dynamic scheduling algorithm like EDF is the possibility to provide temporal guarantees for task set having an higher total utilization with the same QoS for all tasks [22, 24], that can simplify the tasks allocation to cores. Also, the knowledge of task parameters allows to perform offline schedulability analysis and develop acceptance test, while providing online mechanisms against tasks trying to execute more than the declared computation time.

Recently, a new scheduling class has been made available for the Linux kernel, called SCHED_DEADLINE [26]. It implements partitioned, clustered and global EDF scheduling with hard and soft reservations using a variant of the CBS algorithm to obtain temporal isolation among tasks. It does not need any particular hypothesis on tasks characteristics, allowing it to serve periodic, sporadic and aperiodic tasks.

Summarizing, the described solution fulfills all the requirements for the scheduling policy, that are:

- maximize the utilization of the cores without jeopardizing the temporal constraints;
- provide strong temporal isolation against task overrun and malfunctioning code;
- present a bounded service delay for each application.

3 Benchmark results

In this section are presented some results showing the behaviour of two technologies previously described and the compatibility between their performances and the applications constraints. The impacts of the SCHED_DEADLINE scheduling policy and the effects of Direct Interrupt Delivery and Completion on the interrupt latency time.

All the experiments have been executed on a Intel Core i5-4300M processor with 4 cores running at a frequency up to 2.60GHz, equipped with 4Gb of RAM. The selected distribution has been a Ubuntu 12.04 LTS with a standard configuration except for the kernel, whose configurations will be described in the specific tests.

3.1 Evaluation of SCHED_DEADLINE

In this section are presented some tests aimed to show the improvement in terms of temporal isolation provided by the SCHED_DEADLINE scheduling class. All the experiments are performed using the Linux kernel version 3.14.4 that already includes SCHED_DEADLINE.

The first two experiments show the performances of the Netperf[1] tool under different scheduling classes. Netperf is a benchmark that can be used to measure the performance of many different types of networking. It provides tests for both unidirectional throughput, and end-to-end latency. In particular, Figure 6 shows that SCHED_DEADLINE provides a throughput that is almost stable. Instead, the other scheduling policies (i.e., SCHED_FIFO and SCHED_OTHER) are characterized by a throughput with lower average value and greater variability. Thus, SCHED_DEADLINE provides better Quality of Service (QoS).

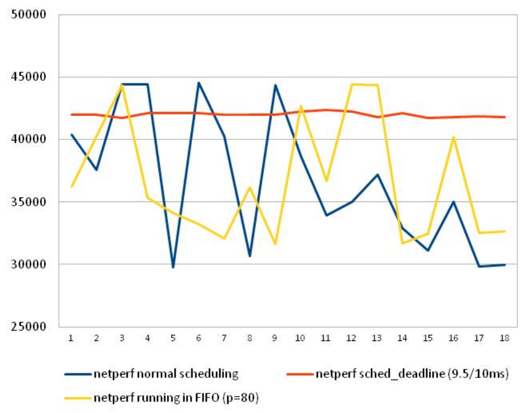

FIGURE 6: *The effect of different scheduling policing on Netperf results*

In the next experiment is shown the throughput of Netperf as a Function of the budget assigned by the SCHED_DEADLINE scheduling class to the application. In Figure 7 are plotted two different assignments: the first with budget $Q = 9.5ms$ and period $P = 10ms$, while the second has budget $Q = 2ms$ and the same period. It is trivial to see that the QoS is almost constant, independent from the application bandwidth request and proportional to the resource allocation fixed by the scheduler.

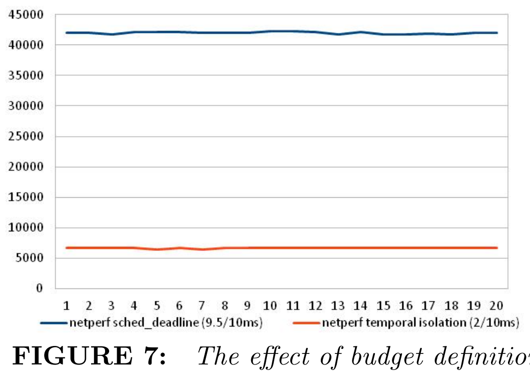

FIGURE 7: *The effect of budget definition on Netperf results*

An experiment has been performed to show the effectiveness of SCHED_DEADLINE in terms of temporal isolation among application. Note that KVM creates one thread per VCPU and one thread per VM doing I/O, thus the idea is to encapsulate VCPU threads inside reservations to provide QoS guarantees. In this last experiment two virtual machines are used. The former one (VM1) runs an instance of

[1]http://www.netperf.org/netperf/

the Iperf[2] server, that is a tool to measure network performance, while the latter virtual machine (VM2) executes the Sysbench[3], that is a benchmark tool for evaluating OS parameters.

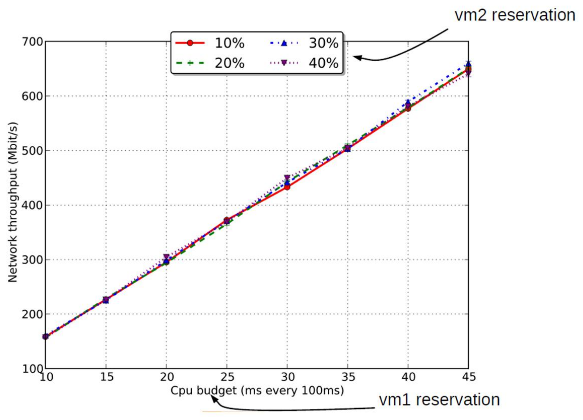

FIGURE 8: *Performances of VM1 as a function of the assigned bandwidth*

The results are shown in Figure 8 and clearly point out how the performance of VM1 is almost linear with the budget allocated to its reservation and remains the same for each value of the budget for the second virtual machine.

3.2 Latency reduction with Direct Interrupt Delivery and Completion.

This test has the goal of showing the reduction in terms of latency that the use of the Direct Interrupt Delivery and Completion technology could provide. The experimental results are collected using the *cyclictest* tool [27] that acquires timer jitter by measuring accuracy of sleep and wake operations of highly prioritized real-time threads. Note that the kernel version used for the test with Direct Interrupt Delivery and Completion is the 3.5.0-rc6, that is the last one for which the Hitachi Lab provided the patch (i.e., version RFC-v2-XX-21-KVM-x86-CPU-isolation-and-direct-interrupt-delivery-to-guest.patch)

Figure 9 presents the distribution of latency times running cyclictest in a virtual machine without the Direct Interrupt Delivery and Completion mechanism. The obtained results show that latencies of the mainline kernel are too long to serve the applications we are investigating.

[2]https://iperf.fr/

[3]https://launchpad.net/sysbench

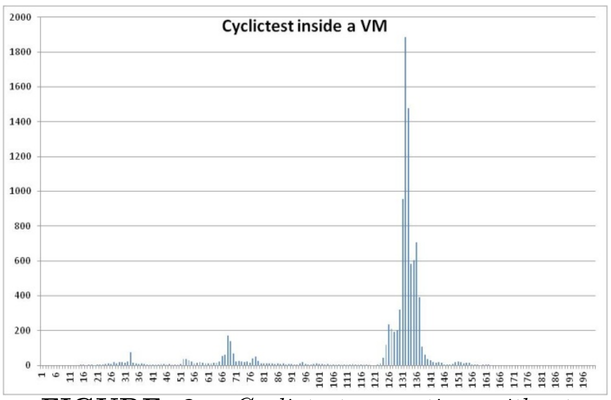

FIGURE 9: *Cyclictest execution without Direct Interrupt Delivery and Completion*

Instead, Direct Interrupt Delivery and Completion is removing most of the virtualization cost, as expected. In this configuration, the latencies of cyclictest running inside the virtual machine are comparable to the results when the test tool is running directly on the host, as they are around $2\mu s$.

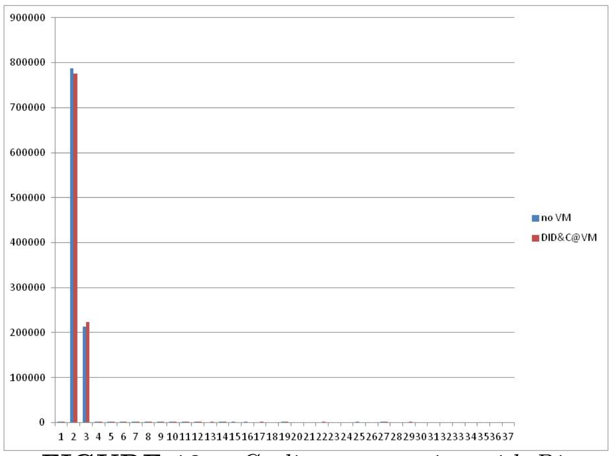

FIGURE 10: *Cyclictest execution with Direct Interrupt Delivery and Completion compared against bare-metal results*

4 Conclusions

In this paper we presented an approach that uses a Linux-based virtualization infrastructure exploiting hardware features provided by modern architectures and kernel extensions to develop a virtualization infrastructure for SDN-NFV applications.

After describing the constraints of the different classes of such applications, the main technology improvements are described. In particular, Direct Interrupt Delivery and Completion allows to shift interrupts handling directly inside the virtual machine, removing the virtualization cost due to the kernel interrupt handling (i.e, VM_EXITs). The reduction of latencies caused by cache pollution from other VMs can be obtained using Huge Page technique.

Bounded latencies and utilization of VMs sharing cores can be obtained using resource reservation techniques like the SCHED_DEADLINE scheduling class that implements the CBS server. Instead, the isolation of a VM holding a dedicated core can be improved with the dynamic tick (dyntick) technique. Using sched_deadline, temporal isolation is always guaranteed. Some experiments are shown to demonstrate that Direct Interrupt Delivery and Completion and SCHED_DEADLINE can provide performances compatible with applications constraints.

Next step will be the integration of all the technologies to provide a prototype of the full environment and its testing with some applications working as a benchmark for the addressed scenario.

References

[1] T. Kridel, *The End of Profitability*, Tellabs Insight Q2, 2011.

[2] A. Manzalini, *SDN and NFV: in track and disruptive strategies*, Telecom Italia workshop, 2013.

[3] T. Braun, *MobileCloud Future Mobile Telecommunication Networks Using Cloud Technologies*, Open Cloud Day, 2012.

[4] *Software-Defined Networking: The New Norm for Networks*, Open Networking Foundation (ONF), 2012.

[5] *Network Functions Virtualisation: An Introduction, Benefits, Enablers, Challenges & Call for Action*, SDN and OpenFlow World Congress, 2012.

[6] *Network Functions Virtualisation (NFV); Use Cases*, European Telecommunications Standards Institute (ETSI), 2013.

[7] R. Rajkumar, K. Juvva, A. Molano, and S. Oikawa, *Resource Kernels: A Resource-Centric Approach to Real-Time and Multimedia Systems*, Proceedings of the Conference on Multimedia Computing and Networking, 1998.

[8] X. Feng and A.K. Mok, *A model of hierarchical real-time virtual resources*, Proceedings 23rd IEEE Real-Time Systems Symposium, 2002.

[9] G. Lipari and E. Bini, *A methodology for designing hierarchical scheduling systems*, Journal of Embedded Computing, 2004.

[10] A. Easwaran and B. Andersson, *Resource sharing in global fixed-priority preemptive multiprocessor scheduling*, Proceedings of IEEE Real-Time Systems Symposium, 2009.

[11] I. Shin, A. Easwaran, and I. Lee, *Hierarchical scheduling framework for virtual clustering multiprocessors*, Proceedings of the 20th Euromicro Conference on Real-Time Systems, 2008.

[12] J. Rodel, *AMD Next-generation interrupt Virtualization for KVM*, AMD Public technical presentation, 2012.

[13] *ARM Architecture Reference Manual: ARMv8, for ARMv8-A architecture profile*, ARM Limited, 2013.

[14] *Intel 64 Architecture x2APIC Specification*, Intel Corporation, 2010.

[15] *Shielded CPUs: Real-Time Performance in standard Linux*, Linux Journal, 2004.

[16] *CPU affinity*, Linux Journal, 2003.

[17] A. Szlavik, *Cache-Aware Virtual Page Management*, University of Waterloo, 2013.

[18] S. Yang, *Extending KVM with new Intel Virtualization technology*, KVM forum, 2008.

[19] F. Weisbecker, *Full dynticks status*, Linux Plumbers Conference, 2013.

[20] T, Sekiyama, *Improvement of Real-time Performance of KVM*, HITACHI Linux Technology Center, CloudOpen '12, 2012.

[21] *ELI: Bare-Metal Performance for I/O Virtualization*, ASPLOS, 2012.

[22] M.L. Dertouzos, M. L., *Control Robotics: The Procedural Control of Physical Processes*, Proceedings of IFIP Congress, 1974.

[23] C.L. Liu and J.W. Layland, *Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment*, Journal of the ACM, Vol.20, No.1, pp.46-61, 1973.

[24] G.C. Buttazzo, *Rate Monotonic vs. EDF: Judgment Day*, Real-Time System, Vol.29, pp.5-26, 2005.

[25] L. Abeni and G. Buttazzo, *Integrating multimedia applications in hard real-time systems*, Proceedings of the IEEE Real-Time Systems Symposium, Madrid, Spain, 1998.

[26] D. Faggioli, F. Checconi, M. Trimarchi, and C. Scordino, *An EDF scheduling class for the Linux kernel*, Proceedings of the Eleventh Real-Time Linux Workshop, Dresden, Germany, 2009.

[27] T. Gleixner, *Cyclictest tool webpage*, https://rt.wiki.kernel.org/index.php/Cyclictest, 2013.