Virtual Network Functions as Real-Time Containers in Private Clouds*

T. Cucinotta, L. Abeni, M. Marinoni, A. Balsini
Scuola Superiore Sant'Anna
Pisa, Italy
Email: firstname.lastname@santannapisa.it

C. Vitucci
Ericsson AB
Stockholm, Sweden
Email: carlo.vitucci@ericsson.com

***Abstract*—This paper presents preliminary results from our on-going research for ensuring stable performance of co-located distributed cloud services in a resource-efficient way. It is based on using a real-time CPU scheduling policy to achieve a fine-grain control of the temporal interferences among real-time services running in co-located containers. We present results obtained applying the method to a synthetic application running within LXC containers on Linux, where a modified kernel has been used that includes our real-time scheduling policy.**

***Index Terms*—cloud computing, real-time scheduling, temporal isolation, performance modeling, network function virtualization**

I. Introduction

Information and communication technologies have undergone a relentless evolution in recent years, with a tremendous push towards distributed computing. The uprise of cloud computing, coupled with the widespread diffusion of broadband Internet connections, caused a paradigm shift towards more and more services provisioned through Cloud Computing infrastructures, in an on-demand, elastic, fashion, with needs of consumers evolving fast towards high-reliability, high-availability, and high-performance.

On the side of infrastructure providers, be it public cloud providers or private ones, there is an increasing interest in the efficient management of the hosted services. This brought to a number of innovations over the last years, ranging from features made available by hardware and CPU manufacturers in form of hardware-assisted virtualization mechanisms to reduce overheads due to machine virtualization, to software solutions based on tweaking internals of hosted operating systems and software stacks, pushing towards para-virtualization or very different solutions based on library operating systems or unikernels [1], [2].

Among others, network operators are looking with a growing interest in adopting (and adapting) the flexibility of (private) cloud computing to the provisioning of network functions as needed throughout their infrastructure. This is witnessed by the growing interest by, e.g., access network operators, in Network Function Virtualization (NFV) [3], a paradigm shift from traditional physical network appliances to network functions deployed as software components throughout a set of data centers. Thanks to the convergence towards IP-based networking, these functions can be managed with the flexibility and dynamicity of a private cloud, becoming effectively Virtual Network Functions (VNFs). These have a strong demand for locality, as these functions stay in the critical per-packet processing path and possess critical latency requirements, calling for solutions that have great efficiency in processing, among others. This is bringing an increasing interest in operating system (OS)-level virtualization techniques [4], i.e. Linux containers, as provided for example by LXC[1], LXD[2] or Docker[3], which exhibit basically no performance loss when compared to bare-metal deployments, still retaining the advantages in isolation, software stack organization and dependencies management typical of deployments using machine virtualization.

Deploying virtual machines (VMs) or containers in a shared infrastructure raises well-known problems of temporal interference among co-located components, particularly in presence of over-subscription of the CPUs, i.e., multiple virtual CPUs (vCPUs) mapped to the same physical CPU (pCPU) of a host, but also in presence of bottlenecks due to other resources, such as disks or networks, for data-intensive workloads. Focusing on the processing performance, controlling the interferences among co-located containers is typically done in nowadays clouds via a 1-to-1 vCPU-to-pCPU allocation, a.k.a., no CPU over-subscription, or dedicating entire physical machines to individual services.

However, this implies a minimum granularity in service allocation equal to the one of a single CPU computing power, with the consequence of a potentially high under-utilization of the infrastructure, in presence of deployed services with relatively small workloads. Therefore, an infrastructure provider is left with the choice between: a) deployments with some degree of over-subscription with potentially highly unstable performance of the individual service as due to the time-varying workload of others sharing the same pCPU(s), because there is little to no control of the interferences among them; b) deployments with no over-subscription, leading to stable performance of individual services at the cost of high under-utilization of the underlying physical infrastructure. Both options have additional drawbacks for a variety of application

*This work has been partially funded by Ericsson.

[1]More information at: https://linuxcontainers.org/lxc/introduction/.
[2]More information at: https://linuxcontainers.org/lxd/introduction/.
[3]More information at: https://docs.docker.com/.

scenarios, including the considered NFV one (i.e., for the nodes at the edge of the network): a) over-subscription impacts on the QoS during traffic peak hours; b) 1-to-1 allocation unacceptably increases power consumption, which is already playing a dominant role in the overall energy budget of access networks.

This work, after a short review of related work in the research literature in Section II, presents in Section III our vision for tackling the problem mentioned above, based on a fine-grain allocation of pCPU(s): a real-time CPU scheduler in the OS kernel guarantees allocation of precise shares of a pCPU time to individual containers with a per-container time granularity, resulting in a stable and predictable performance of the hosted services.

The approach is validated in Section IV by using a patched Linux kernel extending the mainline SCHED_DEADLINE CPU scheduler [5] with hierarchical scheduling capabilities, applied to scheduling containers hosting our synthetic application. Finally, conclusions are drawn in Section V, along with a sketch of directions for future extensions of the work.

II. Related Work

Recent years have seen the growing success of the cloud paradigm outside of its original employment scenarios due to all its well-known advantages. However, those new application fields are characterized by additional constraints that cannot be handled without improving the modeling and the management of the underlying infrastructure, to exploit it more significantly [6].

Several works exist on controlling performance of distributed cloud services via elasticity and auto-scaling mechanisms [7], [8], intelligent placement strategies [9], [10], possibly including network-awareness [11] and SDN-based approaches [12]. To mitigate interferences of co-located services, real-time scheduling applied to hypervisors has been proposed, e.g., for the KVM [13] and Xen [14] hypervisors. In these works the hypervisor is extended to apply hierarchical real-time scheduling theory [15] and to allocate precise slices of the physical CPU execution time to each VM running on the platform. Some extensions to OpenStack [16] have also been proposed to deal with the experimental features introduced at the hypervisor level. The same Xen extensions have also been used for realizing a real-time NFV solution [17].

Some works addressed the problem of accelerating cloud infrastructures with support for heterogeneous hardware platforms, including GP-GPUs [18] and FPGAs [19]. This is orthogonal to the problem of limiting temporal interferences among co-located services, dealt with herein. To this end, a wide range of performance-related features is nowadays available in operating systems and hypervisors, that need to be exposed to higher cloud orchestration layers, something we are also actively working on [6].

III. Proposed approach

Our general reference system, illustrated in Figure 1, is composed of a number of clients submitting requests to a service topology including a number of servers either in charge of picking up a fraction of the incoming service traffic as due to the action of a load-balancer (shown in the top half of the figure), or being part of a group of replicated services, or to be traversed sequentially after one another (service chain, shown in the bottom half of the figure) or a combination of these elements.

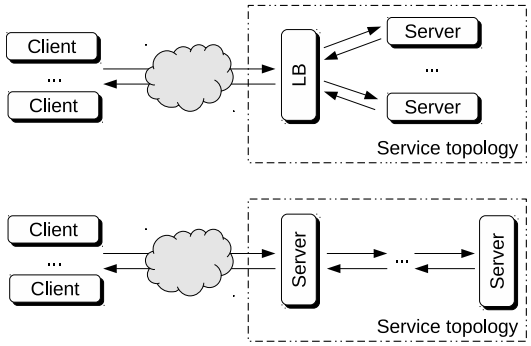

Fig. 1. Reference service topologies.

In these cases, traditional performance control approaches in cloud computing prescribe the use of elastic scaling (typically horizontal, or, less frequently, vertical) coupled with load-balancing within cloud orchestration layers for controlling the overall service QoS. This has two major drawbacks: 1) the approach is viable only with services spanning across multiple instances; 2) if services are co-located on the same physical servers and physical CPUs, to make an efficient use of the infrastructure, then CPU contention causes the performance of each individual server to be highly unstable as due to changes in the workload of co-located services. Furthermore, elastic control loops tend to recover possible performance shortcomings a-posteriori, once the problem becomes evident through monitoring at the orchestrator sensing level, likely once clients have already been impacted by the performance degradation.

In our proposed approach, it is thus of paramount importance to have low-level mechanisms to keep the performance of individually hosted servers as stable as possible, even in cases of co-located functions on the same CPUs/cores. As it will become clear later, this is possible by recurring to special real-time scheduling of the CPU at the OS/kernel (or hypervisor) layer, allowing for temporal isolation among co-located containers. This mechanism can be nicely integrated with standard QoS control mechanisms for cloud services, making it easier to perform said control actions, thanks to the better stability of the performance of individual elements deployed within an elastically provisioned service. Furthermore, in the proposed approach it is possible to dynamically change the scheduling parameters, introducing an additional knob that can be used by an orchestration layer to fine-tune the CPU allocation to individual containers (vertical scalability), while achieving its control goals.

Therefore, in the following, the focus of this paper narrows down to the problem of isolating the performance of individual

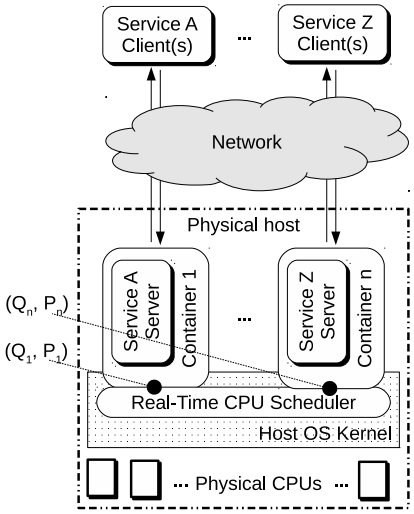

Fig. 2. Proposed approach: n services are deployed as containers over a host with multiple physical CPUs. Container i has runtime Q_i and period P_i.

co-located containers within a cloud platform, with particular reference to CPU scheduling, thus CPU-intensive services[4], as illustrated in Figure 2. The meaning of the per-container scheduling parameters (Q_i, P_i) will be clarified just below.

In the general context just highlighted, our on-going research is looking, among others, at the specifics of the NFV use-case, where a set of Virtual Network Functions (VNFs) are deployed as containers hosting packet processing servers (characterized by heterogeneous timing requirements, as due to different classes of handled traffic) across a number of possibly heterogeneous computing nodes.

A. Hierarchical real-time scheduling of Linux containers

In what follows, the proposed modifications to the Linux real-time scheduler are described, with reference to how the technique has been applied to isolate execution of Linux containers. This way, it is possible to choose the configuration parameters for the server (Q, P) as a function of the desired QoS. That allows a resources controller to figure out the feasibility of a requested throughput based on the underlying resources and the already allocated services.

Linux containers, as created via the `lxc` tool, are associated with a *control group* (`cgroup`) allowing for the specification of *limits* on the amount of resources each container can use, including memory, physical CPUs, as well as limit to the amount of time real-time tasks (`SCHED_FIFO` and `SCHED_RR`) within a container can be scheduled for. We modified the Linux scheduler to allow building theoretically-sound scheduling hierarchies through `cgroups`[5].

The mainline Linux kernel has been recently added the `SCHED_DEADLINE` CPU scheduling class [5], a variant of the well-known EDF-based Constant Bandwidth Server (CBS) [20], allowing for attaching each task with a CPU reservation, expressed in terms of a `runtime` Q and a `period` P, with the meaning that Q time units are granted to the task on the CPU(s) every P time units.

Our "Hierarchical CBS" (HCBS) scheduler[6] extends this mechanism with hierarchical scheduling concepts [15], realizing a mechanism where a CPU real-time reservation can be assigned to a control group as a whole, controlling the amount of time real-time tasks in each group are allowed to run on each CPU/core. This results in a 2-levels hierarchy of schedulers, where `SCHED_DEADLINE` selects the control group to be scheduled on each CPU, and the fixed priority real-time scheduler in the Linux kernel selects one of the tasks from the scheduled control group.

The resulting mechanism, conceptually similar to [21], supports partitioned scheduling in the host (each `SCHED_DEADLINE` entity used to schedule a control group is bound to a CPU/core) and generic affinities in the guest (fixed priority tasks in the control group can have generic affinities; hence both partitioned and global scheduling of real-time tasks are supported).

IV. Experimental results

This section presents some experiments with the approach presented in Section III, using a Linux kernel v4.16.0-rc1, modified with our HCBS patch, and Linux containers through `lxc`, where we have set per-container HCBS parameters (Q, P) as needed for each experiment.

Although the presented HCBS scheduler can support the stochastic analysis based on queueing theory (extending, for example, the analysis already performed for single tasks served by `SCHED_DEADLINE` [22]), it can also support hard real-time scheduling with guarantees provided through the Compositional Scheduling Framework (CSF) [14], [15].

To show this, the task set $\Gamma = \{(4879, 30000), (561, 36000), (10427, 104000), (4408, 109000), (20271, 250000)\}$ (where (C, T) indicates a periodic real-time task with Worst Case Execution Time C and period T; times are expressed in μs - microseconds) has been scheduled by `SCHED_FIFO` and priorities assigned according to Rate Monotonic in an `lxc` container. The container has been assigned various combinations (Q, P) of runtime and period (according to CSF analysis, the task set is schedulable with $Q = 8ms$ and $P = 18ms$), and the resulting *normalized lateness* of the real-time tasks have been measured. The normalized lateness $l = \frac{r-T}{T}$ is defined as the difference between the *response time* r of a task and the task period T, divided by the task period (positive values indicate a missed deadline).

Figure 3 presents the Cumulative Distribution Function (CDF) of the normalized lateness measured for 3 combinations of scheduling parameters:

- $(Q = 8ms, P = 18ms)$: schedulable task set (no missed deadlines) according to CSF analysis. The CDF reaches

[4]For data-intensive services, the technique can be enriched by integrating additional QoS control mechanisms at the networking, disk or I/O layers.

[5]The Linux kernel already provides hierarchical scheduling for real-time tasks, but its design aims only at acting as a limitation, not as a guarantee.

[6]The patch is available at: https://github.com/lucabe72/LinuxPatches/tree/Hierarchical_CBS-patches.

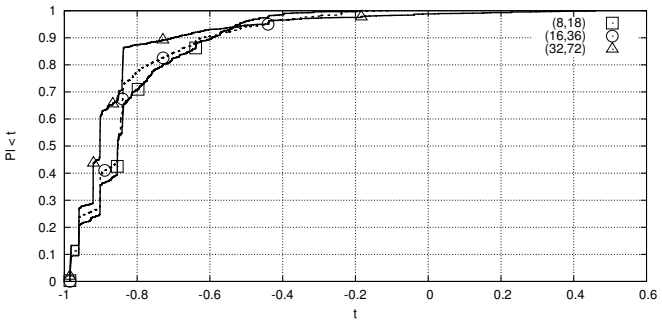

Fig. 3. CDF for the normalized lateness of a task set scheduled in an `lxc` container with various values of runtime and period.

1 for values of the normalized lateness smaller than 0, so the theoretical results are confirmed

- $(Q = 16ms, P = 36ms)$: CSF analysis does not guarantee the schedulability of the task set, however the CDF reaches 1 for values of the normalized lateness smaller than 0 (no missed deadlines). This result does not contradict CSF analysis, that only provides a sufficient schedulability condition, and is quite pessimistic
- $(Q = 32ms, P = 72ms)$: the scheduling system is stable according to queueing theory (the utilization of the task set $U = \sum \frac{C}{T} = 0.4$ is smaller than $32/72 = 0.44444$), but CSF analysis does not guarantee the schedulability of the task set. This is confirmed by the fact that the CDF reaches 1 for a normalized lateness equal to 0.46, so some deadlines are missed.

V. Conclusions

This paper introduced our vision for deploying distributed cloud services with stable performance (with focus on NFV), based on containers and lightweight OS virtualization functionalities. Thanks to the used hierarchical real-time scheduler (which leverages existing theory in real-time literature), the proposed approach provides predictable QoS and can be used, for example, for QoS control in components in the context of 5G network function split. The mechanism ensures stable performance of deployed services, enabling the possibility to apply sound performance modeling, analysis and control techniques.

As future work, we plan to apply our architecture to real software components. For example, we plan to prototype the mechanism within the OpenStack cloud management and orchestration framework, and use the OpenAirInterface[7] software as a case-study related to access network.

References

[1] A. Madhavapeddy, R. Mortier, C. Rotsos, D. Scott, B. Singh, T. Gazagnaire, S. Smith, S. Hand, and J. Crowcroft, "Unikernels: Library operating systems for the cloud," *SIGARCH Comput. Archit. News*, vol. 41, no. 1, pp. 461–472, Mar. 2013.

[2] A. Kantee, "The rise and fall of the operating system," *USENIX login*, vol. 40, no. 5, October 2015.

[7] http://www.openairinterface.org/

[3] NFV Industry Specif. Group, "Network Functions Virtualisation," Introductory White Paper, 2012.

[4] G. Banga, P. Druschel, and J. C. Mogul, "Resource containers: A new facility for resource management in server systems," in *OSDI*, vol. 99, 1999, pp. 45–58.

[5] J. Lelli, C. Scordino, L. Abeni, and D. Faggioli, "Deadline scheduling in the Linux kernel," *Software: Practice and Experience*, vol. 46, no. 6, pp. 821–839, 2016, spe.2335.

[6] T. Cucinotta, L. A. M. Marinoni, and C. Vitucci, "The Importance of Being OS-aware - In Performance Aspects of Cloud Computing Research," in *Proceedings of the 8th International Conference on Cloud Computing and Services Science*, 2018.

[7] A. Ali-Eldin, J. Tordsson, and E. Elmroth, "An adaptive hybrid elasticity controller for cloud infrastructures," in *2012 IEEE Network Operations and Management Symposium*, April 2012, pp. 204–212.

[8] N. Roy, A. Dubey, and A. Gokhale, "Efficient autoscaling in the cloud using predictive models for workload forecasting," in *2011 IEEE 4th International Conference on Cloud Computing*, July 2011, pp. 500–507.

[9] K. Konstanteli, T. Cucinotta, K. Psychas, and T. Varvarigou, "Admission Control for Elastic Cloud Services," in *2012 IEEE Fifth International Conference on Cloud Computing*, June 2012, pp. 41–48.

[10] K. Konstanteli, T. Cucinotta, K. Psychas, and T. A. Varvarigou, "Elastic admission control for federated cloud services," *IEEE Transactions on Cloud Computing*, vol. 2, no. 3, pp. 348–361, July 2014.

[11] M. Alicherry and T. V. Lakshman, "Network aware resource allocation in distributed clouds," in *2012 Proceedings IEEE INFOCOM*, March 2012, pp. 963–971.

[12] T. Cucinotta, D. Lugones, D. Cherubini, and E. Jul, "Data Centre Optimisation Enhanced by Software Defined Networking," in *2014 IEEE 7th International Conference on Cloud Computing*, June 2014, pp. 136–143.

[13] T. Cucinotta, G. Anastasi, and L. Abeni, "Respecting temporal constraints in virtualised services," in *2009 33rd Annual IEEE International Computer Software and Applications Conference*, vol. 2, July 2009.

[14] J. Lee, S. Xi, S. Chen, L. T. X. Phan, C. Gill, I. Lee, C. Lu, and O. Sokolsky, "Realizing compositional scheduling through virtualization," in *2012 IEEE 18th Real Time and Embedded Technology and Applications Symposium*, April 2012, pp. 13–22.

[15] I. Shin and I. Lee, "Compositional real-time scheduling framework," in *25th IEEE International Real-Time Systems Symposium*, Dec 2004, pp. 57–67.

[16] S. Xi, C. Li, C. Lu, C. D. Gill, M. Xu, L. T. X. Phan, I. Lee, and O. Sokolsky, "RT-Open Stack: CPU Resource Management for Real-Time Cloud Computing," in *2015 IEEE 8th International Conference on Cloud Computing*, June 2015, pp. 179–186.

[17] Y. Li, L. T. X. Phan, and B. T. Loo, "Network functions virtualization with soft real-time guarantees," in *IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications*, April 2016, pp. 1–9.

[18] X. Yi, J. Duan, and C. Wu, "GPUNFV: A GPU-Accelerated NFV System," in *Proceedings of the First Asia-Pacific Workshop on Networking*, ser. APNet'17. New York, NY, USA: ACM, 2017, pp. 85–91.

[19] X. Ge, Y. Liu, D. H. Du, L. Zhang, H. Guan, J. Chen, Y. Zhao, and X. Hu, "OpenANFV: Accelerating Network Function Virtualization with a Consolidated Framework in Openstack," in *Proceedings of the 2014 ACM Conference on SIGCOMM*, ser. SIGCOMM '14. New York, NY, USA: ACM, 2014, pp. 353–354.

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

[21] F. Checconi, T. Cucinotta, D. Faggioli, and G. Lipari, "Hierarchical Multiprocessor CPU Reservations for the Linux Kernel," in *Proceedings of the 5th International Workshop on Operating Systems Platforms for Embedded Real-Time Applications (OSPERT 2009)*, June 2009.

[22] T. Cucinotta, M. Marinoni, A. Melani, A. Parri, and C. Vitucci, "Temporal Isolation Among LTE/5G Network Functions by Real-time Scheduling," in *Proceedings of the 7th International Conference on Cloud Computing and Services Science*, Funchal, Madeira, Portugal, March 2017, pp. 368–375.