The Importance of Being OS-aware

In Performance Aspects of Cloud Computing Research

Tommaso Cucinotta¹, Luca Abeni¹, Mauro Marinoni¹ and Carlo Vitucci²

¹Scuola Superiore Sant’Anna, Pisa, Italy
²Ericsson, Stockholm, Sweden

Keywords: Cloud Computing, Operating Systems, Quality of Service Control, Temporal Isolation, Real-Time Scheduling.

Abstract: This paper highlights inefficiencies in modern cloud infrastructures due to a distance between the research on high-level cloud management/orchestration and the research on low-level kernel and hypervisor mechanisms. Our position about this issue is that more research is needed to make these two worlds talk to each other, providing richer abstractions to describe the low-level mechanisms and automatically map higher-level descriptions and abstractions to configuration and performance tuning options available within operating systems and kernels (both host and guest), as well as hypervisors.

1 INTRODUCTION

Despite the tremendous growth and transformation undergone by Cloud Computing over the last 10 years, this computing paradigm finds its roots back in the ’60s. The paradigm of utility computing was mentioned in the MIT centennial speech by John McCarthy in 1961 (Garfinkel, 2011), in its famous statement: “Computing may someday be organized as a public utility just as the telephone system is a public utility”. Also, machine virtualization, one of the core enablers of cloud computing, had been conceptualized and prototyped already in the ’70s (White, 2015), albeit the hardware available at that time was not ready for a widespread use of the mechanism. Furthermore, since the Internet was born, we have seen web-hosting servers on rental, sometimes along with simple accompanying data-base and back-up services. However, the actual possibility for the general public to be able to rent virtual machines (VMs) on-demand, rapidly and in a self-service and completely automated fashion, dates back to year 2006, when Amazon Web Services (AWS) launched its Elastic Compute (EC2) service (Information Resources Management Association, 2012).

At that time, you could rent simple virtual machines choosing among a few instance types (Barr, 2015), using an Infrastructure-as-a-Service model. Later, other players entered the market, and the available services evolved up to the nowadays native cloud applications making use of a plethora of services available from a Platform-as-a-Service (PaaS) provider, like security, load-balancing, high-availability, data storage and replication, distributed communication middleware and queueing, big-data processing frameworks, etc. Recently, operating-system (OS) level virtualization, a.k.a., containers, are increasingly replacing machine virtualization in various application domains, finding a natural fit within novel distributed processing architectures based on microservices (Brown, 2016).

Academic research has been flanking the emerging industry of cloud computing services, experimenting with novel directions of investigation, mostly inspired to the consolidated areas around distributed computing, fault-tolerance, data-bases, networking protocols, routing, data replication and distributed transactions, security, etc., with many of these research areas overflowing almost naturally from prior, consolidated research lines in the field of grid and distributed computing. Many research groups focused on novel issues coming straight from particular characteristics of the cloud computing paradigm instead, such as the capability to grow and shrink service instantiations by adding or removing VMs at runtime, live-migrate them for re-optimization purposes, as well as the one to create virtual networking overlays on top of the physical infrastructure. This resulted in novel research on scalable resource manage-
ment in massively distributed infrastructures, dealing with issues related to monitoring, resource estimation and workload prediction (Cortez et al., 2017), optimal VM placement (Chang et al., 2010; Mann, 2015) including cases of network-aware placement (Alicherry and Lakshman, 2012; Yao et al., 2013; da Silva and da Fonseca, 2016; Steiner et al., 2012) and allocation of data-intensive VMs (Zhang et al., 2015), energy efficiency (Ghribi et al., 2013), elastic applications management and associated performance models (Xia et al., 2015), architectures of cloud orchestration layers, and others.

Just recently, further fields of particular success in terms of research productivity have been the ones of cloud federation (Konstanteli et al., 2014) and edge (cloud) computing. The last one, in particular, includes cloudlets (Satyanarayanan et al., 2009) and the cloudification of other IT areas such as those of interest for network operators, with the uprising trend of software defined networking (SDN) and network function virtualization (NFV) (Public, 2016; Networks, 2017) and related technologies (Fiorani et al., 2015). These application scenarios are characterized by tighter constraints concerning low latency and overall Quality of Service (QoS) for mobile applications (Valcarenghi et al., 2017). Moreover, the development of end-to-end SDN-NFV solutions cannot ignore the characteristics and, therefore, the needs of Virtual Functions (VF) that have to be placed in nodes located in different zones of the network, such as the core or the edge. VFs assigned to core nodes present similar needs to the classic cloud infrastructures, while the closer you get to the edge of the network, the more the demand for flexible management of the computing resource becomes critical (Vitucci and Larsson, 2017). These requirements are crucial both concerning the QoS levels and the available resources optimization because, in such contingencies of scarce resources and significant power consumption (SCTE, 2016), a more tailored allocation of energy and resources becomes essential (Gai et al., 2016).

2 THE ROLE OF THE OPERATING SYSTEM

As due to the intrinsic nature of cloud computing, performance of cloud-hosted applications, services, and internal infrastructure management systems has always been of paramount importance, in the overall picture. Similarly to when one buys physical hardware, renting on-line IaaS facilities needs to respect minimum performance specifications that are usually made more or less explicit by the provider. However, plenty of the mentioned research works tackling performance and QoS control aspects of cloud systems, have been focusing on exploiting elasticity to just compensate for the great variability in performance of the rented infrastructure items (unstable VMs processing performance, usually, due to time-sharing of the physical CPUs, but also instability in disk access as well as networking performance). Therefore, one finds an increasing presence of high-level resource management frameworks, orchestration layers and evolved middleware services, dealing with workload monitoring, estimation and prediction, as well as resource management.

One aspect that has not been given sufficient attention, in our opinion, is the bottommost part of the software stack: the operating system (OS) and hypervisor layers. Indeed, a significant amount of research in the field focused on the higher levels of the cloud stack, just relying on well-established basic functionalities provided by traditional OSes, first and foremost, the ability to pin down virtual cores onto physical cores of the underlying hardware platform, and in some cases fine-tuning of the hypervisor memory allocation subsystem by exploiting the NUMA capabilities of the hardware. However, OSes and hypervisors have been evolving over time, exhibiting richer and richer functionality for supporting fine-tuning of performance, with reference to achievable latency and throughput within data centre infrastructures, encompassing multiple areas, such as: CPU scheduling, memory bandwidth allocation and management, interrupt dispatching and virtualized interrupt handling, para-virtualization options and ways to access hardware accelerators including GPUs, GP-GPUs, cryptographic accelerators or generic FPGA-based processing elements (Caulfield et al., 2016; Pellerin, 2017). As exemplified in Figure 1, these capabilities are often pre-tuned according to data centre-wide procedures (or just set as found in OS and other software distributions), and from the high-level perspective of infrastructure orchestration layers they are either not made accessible, or subsumed at a too high of an abstraction level, thus resulting in a mismatch with the plethora of heterogeneous requirements coming from different applications, resulting in a poor effectiveness in infrastructure management.

However, it is our belief that the novel panorama of heavily distributed, elastic, fault-tolerant and often interactive, computing applications and services calls for novel OS services and low-level middleware, and that these need to be better connected to higher-level cloud management and orchestration layers. Using more advanced functionalities provided by mod-
modern OSes can significantly improve the performance, scalability, predictability thus usability of cloud services, making them even more attractive.

3 LOW LEVEL SUPPORT

This section briefly introduces some of the advanced features that can be provided at the lower-levels of the cloud stack, and that could be exploited by higher level cloud control infrastructures to improve performance.

3.1 CPU Scheduling

Management of the CPU resources in cloud infrastructures follows basically two main patterns: the best VM instance types are mapped 1-to-1 so that virtual CPUs run onto dedicated physical CPUs. On the other hand, the cheapest VM instance types are mapped onto clusters of physical machines where there is an over-allocation of virtual CPUs, compared to the available physical CPUs. Finally, the most expensive and best isolated instance types are those relying on entire physical machines dedicated to a single tenant’s instance, either in virtualized or also in bare-metal form.

Focusing on physical machines shared among multiple tenants, the dedicated 1-to-1 virtual to physical core mapping is the set-up that tries to achieve the best temporal isolation among different VMs. However, this is not sufficient to isolate data-intensive applications, such as high-definition multimedia (video) processing workloads, big-data analytics and others. These, even when deployed on dedicated cores, cause quite a lot of interference to processing tasks deployed on other physical cores, due to interferences at the memory access bandwidth and cache-level interferences. Works aimed at limiting said phenomenon can be found (Yun et al., 2013), specifically confining a precise budget in memory access for individual physical cores, that, once exhausted, would cause the VM to be throttled till replenishment of the budget at the next period. Mechanisms have been studied also to limit and keep under control interferences at the cache level on big multi-core machines, where there is a sufficiently big last-level cache (LLC). In such systems, it is both possible to employ software-only cache partitioning schemes based e.g., on cache colouring (Kim and Rajkumar, 2016), or to leverage hardware technologies, such as the recent Intel Cache Allocation Technology (CAT (Corbet, 2016).

Whenever virtual CPUs are over-allocated to physical ones, the way multiple VMs compete among each other for running on the physical CPUs when active at the same time is left to heuristics of the hypervisor scheduler, which unfortunately falls short in its attempt to satisfy requirements of a multitude of application domains, without knowing much about what kind of service/application is running within what VMs. Normally, the temporal interferences a VM gets from other co-scheduled VMs are quite high, making the performance of each hosted VM quite unstable. In this area, prior research works focused on the Xen (Barham et al., 2003) hypervisor, comparing various scheduler types (Cherkasova et al., 2007) and its impact on the performance of the hosted workload.

Other research works focused on co-scheduling virtual CPUs of VMs on the same physical CPUs by employing a special scheduler within the hypervisor, that allows for a reservation-based paradigm. Namely a virtual core may be allotted a budget (a.k.a., runtime) of $Q$ time units every period of $P$ time units, so that the underlying hypervisor (or host OS) scheduler guarantees $Q$ time units every period of duration $P$, and optionally it also constrains the VM to be throttled once said budget is exhausted. This is the example of the IRMOS real-time scheduler for Linux (Cucinotta et al., 2010; Checconi et al., 2009), or the similar recent hierarchical extension of the SCHED_DEADLINE scheduler of Linux (Balsini et al., 2017). Similar efforts have been attempted on Linux/KVM also by exploiting existing in-kernel
features, so that EDF-based scheduling capabilities could be injected into the kernel at run-time, without any need for static code modifications to the kernel code (Aesberg et al., 2011).

3.2 Networking

The service performance are affected not only by the configuration of the used VM, but also by the OS and kernel mechanisms used to relay data among VMs, between host and guest applications, etc... For example, it has been shown (Abeni et al., 2015) that, when using Linux as host OS and KVM as hypervisor, simply changing the packets forwarding mechanism from the default bridge + tap to macvtap can greatly improve the performance of a virtual router. In particular, the rate of small packets the virtual router can forward has been shown to increase of nearly 50%. Other configuration details (such as the queue length of the virtual interfaces, the scheduling priority and interrupt affinity, etc...) that are often ignored in high-level descriptions also affect the performance and the ability to sustain a high forwarding rate in a stable way.

Of course, many of the low-level details are OS or VM specific (for example, the possibility to use macvtap or vhost-net are Linux-only features; Xen provides a completely different way to forward packets - with different configuration options - etc...). This is why many cloud providers often rely to default settings, that do not require any knowledge of the OS, kernel, and VM details, resulting, in much worse performance and usability. As an additional example, high-performance networking in cloud services can often be obtained by using some form of user-space networking (even arriving to user-space network drivers). In this regard FreeBSD provides netmap (Rizzo, 2012a; Rizzo, 2012b), that marked a significant milestone in the area of packet processing in general-purpose operating systems, showing how well-engineered optimizations of existing drivers, coupled with proper novel user-space/kernel-space interfaces for networking, can lead to a great increase in processing throughput of small packets. On different operating systems, similar results can be obtained by using different mechanisms; for example, on Linux-based systems it is possible to use DPDK\(^1\), that has been shown to greatly improve the performance (especially in software defined networks) (Pongrcz et al., 2013). Virtual bridges between different VMs also can be setup in different ways, using different technologies that depend on the used network drivers; for example, if netmap is used then the VALE virtual switch (Rizzo and Lettieri, 2012) is the best choice for high performance communications among VMs on the same host, reaching up to 2 million packets per second without hardware assistance.

Since it is clear that an administrator in charge of setting up a cloud cannot have all of the information needed to properly select the most appropriate technologies and configurations, it becomes clear that it is of vital importance to automate this process by properly mapping high-level descriptions understandable by the administrator to low-level features. To do this, it is fundamental to develop a way to make the low-level virtualization infrastructure attributes directly visible to the higher levels of the cloud management stack, or exploit richer and more complex abstract interfaces, that allow for properly tuning / configuring / using all the available options (even if they are not standard).

3.3 Computing and Networking

Some authors tried to build mechanisms to improve the responsiveness of networking applications, in presence of concurrently running CPU-bound activities. This is the case of the work in (Kim et al., 2009), where a heuristic identifies I/O tasks within a VM, and boosts their scheduling priority (tuning parameters of the Xen credit-based scheduler) whenever there are pending incoming packets to be received by that task in the VM. A similar approach has been attempted on KVM-based systems (Cucinotta et al., 2011), where a VM being scheduled with a reservation-based scheduler has been made able to dynamically switch to a finer-grain scheduling deadline whenever there were packets pending to be processed by particular services running in the VM at higher priority.

4 EXPLOITING THE LOW-LEVEL FEATURES

When the performance of the cloud services becomes relevant, it is important to properly map high-level performance requirements of the cloud application or service to low-level virtualization infrastructure configuration and tuning parameters.

---

\(^1\)http://www.dpdk.org
4.1 Adoption of Reservation-based Approaches

As an example, consider a specific area of system support (CPU scheduling): one of the obstacles into getting reservation-based scheduling widely adopted, is that software design needs a shift from tuning priorities, to tuning budgets and periods, a change that is not trivial at all. While a reservation period is easily found considering activation and latency constraints of an application/service, the budget/runtime is more difficult to find out, also due to its inherent dependency on the specific platform the software runs atop.

An interesting line of work has been proposed in the RT-Xen project (Xi et al., 2011), investigating on the use of various server-based scheduling algorithms for Xen domains, with scheduling parameters set according to sound arguments from hierarchical real-time theory literature (Feng and Mok, 2002). RT-Xen, like many of the research works mentioned above, only focused on low-level kernel or hypervisor mechanisms, implementing features that do not result to be available to higher levels of the cloud management stack. The only work that tries to address this concern is RT-OpenStack (Xi et al., 2015), that tries to integrate the analysis within the OpenStack Nova scheduling algorithm. While a lot of work still needs to be done in this area (for example, RT-OpenStack is strictly dependent on RT-Xen and cannot take advantage of the SCHED_DEADLINE (Lelli et al., 2016) policy that is currently available on vanilla Linux kernels), the RT-Xen / RT-OpenStack work shows a possible strategy for exploiting advanced features provided by the lower levels of the cloud stack.

The next subsection shows how a similar approach can be extended to better integrate with higher level abstractions and to consider other lower level features.

4.2 Bridging Gaps in Abstraction Levels

One of the interesting results shown by RT-OpenStack is that to make a real and practical use of all the OS-level improvements described in the previous section, it is crucial to export all the available features and tuning points to have them available during the placement.

To reach such a result, the model of the underlying infrastructures has to be enhanced, and the corresponding descriptors must be enriched to describe such an enhancement. As an example, consider OpenStack and the Heat Orchestration Template (HOT) used by the OpenStack orchestration engine (Heat) to describe the system resources. The level of details currently described by the template has been defined focusing on homogenous cloud infrastructures and is unable to exploit hybrids clouds and modern features provided at the OS level in each physical node. This lack concerning representativeness leads to a suboptimal use of resources also at higher levels of the stack.

For example, Tacker (Orchestration, 2017), the NFV orchestrator, that is in charge of managing NFV component cannot utilize specific OS features in the MANO (ETSI, 2014) descriptors because they are not exported by the Virtualized Infrastructure Manager (VIM) (aka OpenStack). This could jeopardize the possibility to satisfy the strict latency requirements of this kind of applications.

To address this issue, the interfaces among the different layers must be enriched allowing to describe the QoS requirements regarding the computational capacity at all levels, as shown in Figure 2.

Within the MANO descriptor of each NFV component, we can find the definition of the required Virtual Deployment Units (VDU) needed to deploy that service. Each of them is characterized by a specification of the number of required CPUs, which should be extended to include the capability to declare CPU fractions defined by budget and period or by utilization and maximum service delay, in order to exploit reservation-based real-time scheduling features. This enriched VDU descriptors would be processed by the
VIM to produce the corresponding orchestration templates (HOT), equivalently enhanced, required by the Heat orchestration engine. With such an approach, a real-time container or VM could be deployed according with the specified scheduling parameters, after proper admission control done at the orchestration layer, exploiting the kernel extensions and modern scheduling algorithms, such as SCHED_DEADLINE.

4.3 Guest Architecture

Although most of the previous discussion focused on the issues in mapping high-level abstractions to the host OS and VMM configurations, similar opportunities for improvements can be encountered in the configuration of the guest system.

For example, many of the currently used cloud management stacks end up installing a complete general-purpose OS (that can run in containers or in more isolated VMs) for every cloudified service. However, better performance (with smaller resource requirements) can be achieved by properly configuring the guest OS to take advantage of the virtualization features or even using specialized guest OSes. Of course, this requires some visibility of the virtualization infrastructure from the guest.

Although Unikernels (Madhavapeddy et al., 2013; Kantee, 2015) have been proposed as a way to easily specialize the guest OS for specific services and specific virtualization technologies, many for the of the most commonly used cloud management stacks are not able to properly exploit this possibility.

Again, an appropriate mapping of abstract service descriptions to guest system details (and a better description of the host OS / hypervisor capabilities) is needed to allow the automatic configuration and deployment of virtualized services with better performance.

5 CONCLUSIONS

A major drawback of existing research efforts in performance control and isolation of multiple VMs when co-located on the same physical hosts resides in their lack of direct and easy applicability in the context of real industrial cloud providers. For example, with a few exceptions, the above works provide prototype implementations in the form of hacks to the original open-source software ensemble (e.g., Xen, Linux kernel, KVM sources), rather than well-engineered solutions that can readily be leveraged in higher-level infrastructure management stacks.

Indeed, industrialization of prototypes as described in a research paper can be something totally non-trivial. As it was the case for SCHED_DEADLINE, having a prototype non-standard mechanism landing into the mainline Linux kernel can cost several months of development effort, with continuous interactions with core kernel developers and maintainers. Therefore, we have several of these custom approaches, patches and hacks, that are rarely made available and released in the public in a usable state, with the natural side-effect that it is overly difficult to design higher-level cloud and NFV orchestration layers that can rely on such experimental low-level features. On the side of OS vendors and maintainers, it is always difficult to incorporate novel non-standard features, without a clear idea of the need for said features by at least a community of users. This chicken-and-egg situation impairs availability of advanced performance control features in orchestration engines, which, coupled with the lack of exposure of the plethora of capabilities and performance tunables already available in nowadays OS and hypervisors, leads to the consequence that it is quite difficult for real cloud and NFV providers to leverage solid and important research results in these areas, as well as existing advanced performance control mechanisms.

In our opinion, an important step in this regard is to increase exposure of low-level performance tunables to the high-level layers of the infrastructure orchestration suite, in order to allow for a proper matchmaking process that considers also high-level application/service requirements. Therefore, we are working on further research along these lines.

REFERENCES


ETSI. (2014). Network functions virtualisation (NFV); management and orchestration. http://www.etsi.org/deliver/etsi_gs/NFV-MAN/001_099/001/01.01.01_60/gs_nfvm-
man001v010101p.pdf.


Kantee, A. (2015). The rise and fall of the operating system. USENIX login, 40(5).


