Rebooting Virtualization Research (Again)

Alain Tchana, Renaud Lachaize

Rebooting Virtualization Research (Again)

Alain Tchana
Université Côte d'Azur, CNRS, I3S
06900 Sophia Antipolis, France
alain.tchana@unice.fr

Renaud Lachaize
Univ. Grenoble Alpes, CNRS, Grenoble INP, LIG
38000 Grenoble, France
renaud.lachaize@univ-grenoble-alpes.fr

Abstract

Visible or hidden, virtualization platforms remain the cornerstone of the cloud and the performance overheads of the latest generations have shrunk. Is hypervisor research dead? We argue that the upcoming trends of hardware disaggregation in the data center motivate a new chapter of virtualization research. We explain why the guest virtual machine abstraction is still relevant in such a new hardware environment and we discuss challenges and ideas for hypervisor and guest OS design in this context. Finally, we propose the architecture of a research platform to explore these questions.

ACM Reference Format:
Alain Tchana and Renaud Lachaize. 2019. Rebooting Virtualization Research (Again). In *10th ACM SIGOPS Asia-Pacific Workshop on Systems (APSys '19), August 19–20, 2019, Hangzhou, China.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3343737.3343746

1 Introduction

Hypervisors remain a major building block of cloud computing: they are the cornerstone of Infrastructure-as-a-Service (IaaS) platforms, which are themselves used to implement other cloud models such as Containers-as-a-service (CaaS), Functions-as-a-Service (FaaS), and (some types of) Platform-as-a-Service (PaaS). However, while there are vibrant discussions regarding the best ways to design high-level cloud services [18, 28], it may seem that the main research questions regarding hypervisors have now been settled and that the scientific community should instead focus on other concerns. Indeed, to some extent, the past decade has mostly been spent refining the details of a design vision outlined long ago for more efficient and more secure virtualization [23], which relies on static partitions of the physical resources, hardware assists and direct management of I/O devices and interrupts by guests. These efforts have paid off and are already leveraged in production: for example, AWS claims that its Nitro virtualization platform achieves near bare-metal performance [15, 25].

APSys '19, August 19–20, 2019, Hangzhou, China

While it seems that a significant chapter of virtualization research has reached its end, we argue that a new one must begin, motivated by major, upcoming trends in data center design: the shift from monolithic computers (*computer-centric* architecture) to pools of disaggregated resources (*resource-centric* architecture) will lead the systems community to deeply reconsider the ramifications of low-level virtualization in the cloud.

The paper is organized as follows. First (§2, §3), we review the hardware disaggregation trend and underline two potential visions: the use of a small number (a few tens) of large resource blades vs. many (a few hundreds) smaller resource blades. We present the implications of each vision. Second (§4), we stress the necessity of keeping the virtualization layer in this new architecture. In fact, disaggregation makes a data center or a rack behave like a giant computer, motivating more than ever before the need of building isolated execution units. Third (§5), we describe the main systems challenges raised by disaggregation. We argue that although past approaches can be used for addressing some challenges, others require new ideas. In particular, we consider how the split-kernel approach [30] can inspire the general architecture of a distributed hypervisor (*NewVirt*, discussed in §6). Conversely, we motivate the need for a novel approach, *Splinked OS* in order to facilite the construction of distributed guest OSes for a disaggregated architecture, by simplifying the porting of existing guest OSes and supporting unmodified applications (§7). We conclude the paper in §8.

2 Data center disaggregation

Since the seminal paper on the resource disaggregation approach by Lim et al. [26] in 2009, several research works have investigated it [5, 9, 12, 24, 29, 30]. This section summarizes the state of the art and its implications.

Traditional architecture. The architecture adopted in current data centers (DCs) is *computer-centric* (see Fig. 1 top). It means that the DC is composed of computers, interconnected using classical network equipments (switches, routers, etc.). Each computer includes almost all resource types (CPU, RAM, etc.) and accelerators (GPU, FPGA, etc.). In this architecture, an application (OS process) runs on a single computer at a given time.

Disaggregated architecture. In this architecture, the DC is *resource-centric* (see Fig. 1 bottom): it is composed of resource

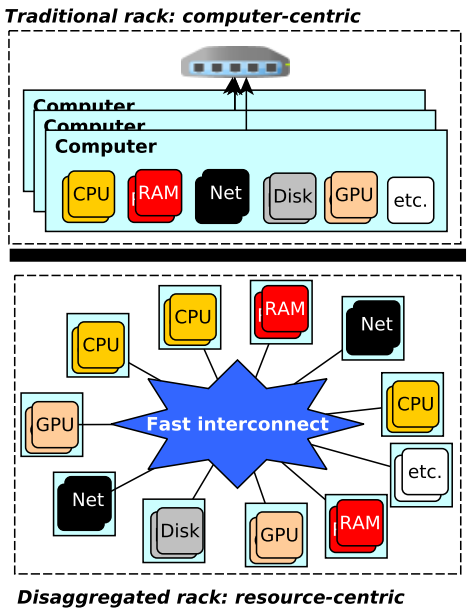

Figure 1. The architecture of a data center rack: computer- (today) vs. resource-centric (the future).

blades, interconnected using fast networking technologies (e.g., silicon photonics [12] or infiniband [9]). Each resource blade includes only one resource or accelerator type (e.g., CPU or main memory). For scalability purposes, data center disaggregation is imagined at the rack scale, in the same way as current data centers are organized in racks. Thus, a rack is seen as a "giant" computer. An application (OS process) running on such a disaggregated rack uses several resource blades at a time (a least one blade per resource type).

Two opposed visions of this architecture have emerged in research papers. In the first vision [26] (that we call *heavy blade approach*), the rack is composed of a small number (a few of tens) of "heavy" resource blades, i.e., with each blade including a large pool of resource instances. The second vision [30] (that we call *lightweight blade approach*) promotes the utilization of very small blades, with each blade including a small number of resource instances. The consequence is the multiplication of the number of blades (in the order of hundreds) in the rack. This second vision is more challenging than the first vision. However, it provides all the potential benefits of a disaggregated architecture, which is not the case for the *heavy blade approach*, as detailed below.

Benefits of disaggregation. We summarize below the main benefits expected from resource disaggregation. When necessary, we highlight the ones than can only be achieved with the lightweight blade approach.

1. *It facilitates the scalability of each resource type.* For instance, the memory capacity of the rack can be increased without increasing the number of CPUs.
2. *It facilitates application scalability* since the application can use the entire resource rack. This reduces the development burden for the application owner.
3. *It allows optimal resource utilization* (thus good energy proportionality) by minimizing resource fragmentation, especially when small blades are used.
4. *It limits the failure impact of a given resource type, especially when small blades are used.* In fact, the failure of a blade is limited to that blade, it does not impact other blades neither of the same type nor of a different type.
5. *It facilitates the rapid integration of new technologies, especially for small blades.* For instance, the integration in the rack of a new CPU generation does not require to buy a whole computer, and thus other resource types.

Since it is is not clear yet if one of the two visions (lightweight/heavy blades) will dominate in the future, we believe that it is important to consider software designs that can accomodate one or the other (or a mix of both).

3 Related work

The disaggregation paradigm is inspired by various concepts from the past: for example, in the context of the storage, diskless machines and network block devices. But research regarding more advanced storage rack disaggregation is recent and still ongoing [24]. Similarly, several works have studied memory disaggregation beyond the initial work by Lim et al. [26]. For example, Nitu et al. [29] studied memory disaggregation using commodity servers, and proposed a new ACPI state which allows remote access to the memory of a suspended server. This way, the pool of suspended servers can be used as a supply of memory by powered-on servers. Several manufacturers have also proposed intermediate approaches for rack disaggregation based on micro computers, for example Intel Rack-Scale, AMD SeaMicro and HP Moonshot. All the above-mentioned works correspond to an intermediate step towards full disaggregation. In the context of the dRedBox European project [7], IBM along with several academic institutions are currently trying to build a real disaggregated rack prototype; this work only focuses on hardware aspects.

Shan et al. [30] have introduced an OS blueprint and prototype for disaggregated racks. The authors introduced the concept of *split-kernel*, which consists in organizing the OS of a disaggregated rack as a set of dedicated mini-OSes per resource blade type. While we believe that the general idea of this approach is a sound basis for a fully disaggregated architecture, we question some of their main design choices. First, LegoOS does not provide complete support for legacy applications and some blade configurations. For example, all the threads of a process must be located on the same CPU blade, which may be limiting depending on factors such as the granularity of the blade resources (lightweight/heavy).

Second, we believe that the low-level software layer should hide as much as possible the complexity introduced by disaggregation but nonetheless preserve a hardware-centric interface, akin to IaaS (Infrastructure-as-a-Service) platforms in computer-centric designs; this aspect is detailed in the next section.

4 The need for virtualization

An important question that arises when contemplating the implications of hardware disaggregation is: which interface should be exported by the low-level software layers? Previous works that have considered the disaggregation of a single resource (e.g., memory [26, 29]) have quite naturally chosen a hardware-like interface (thus VMs) in order to facilitate integration with the rest of the system. However, the question becomes more open-ended when considering the case of full disaggregation (i.e., for all resource types); LegoOS [30], the only system to date to tackle this issue, exports a traditional, Linux-inspired process ABI.[1] After all, one can argue that a rack-scale disaggregated hardware infrastructure can be considered as an extreme form of a single large-scale NUMA machine with hot-pluggable components. Furthermore, such an abstraction seems to be a good fit for modern cloud-native workloads, which are mostly based on containers. However, we believe that it is generally more relevant to export such disaggregated resources via a lower-level, hardware-like interface, i.e., a hypervisor offering the abstraction of several independent (virtual) machines hosted in the same rack. Below, we summarize the main arguments that motivate our case.

First, the machine abstraction is required to support legacy systems and, indirectly, platforms and applications that rely on the features of these systems. In addition, a low-level interface provides a substrate that supports diverse kinds of unmodified guest software while allowing guest-specific and hardware-specific optimizations through paravirtualization.

Second, a low-level interface remains a more generic way (and hence more pragmatic, due to the large diversity of workloads) to deal with common administrative tasks such as snapshots and live migration (across racks).

Third, it is generally admitted that enforcing strong security isolation requires multiple layers of defense mechanisms, including at low levels of abstraction. This is typically why most public cloud providers use virtual machines to encapsulate the containers of different tenants in distinct hardware-enforced security boundaries [4, 33]. Besides, hardened sandboxing solutions located at higher levels in the software stack like gVisor [13] have additional shortcomings such as limited syscall compatibility and significant performance overheads for some workloads [31].

Finally, we can also borrow insights from past episodes of infrastructure research. For example, twenty years ago, the Cellular Disco project [14] introduced a hypervisor-based approach to address performance scalability and hardware fault containment issues in a large-scale SMP server, turning it into a virtual cluster with flexible resource management. Subsequent work has shown that monolithic, general-purpose operating systems can be modified to address these challenges but still at the cost of significant engineering efforts. Given the ever increasing scale, density and heterogeneity of resources hosted in a cloud rack, we believe that addressing the above-mentioned challenges at the hypervisor level remains a more relevant strategy today.

Therefore, in the remainder of this paper, we discuss the implications of this hypervisor-based approach on the hardware/software co-design of a disaggregated infrastructure.

5 Challenges

In this section, we try to identify and discuss the main system challenges raised by disaggregated virtualization. We begin with key architectural considerations (§5.1) and then touch on more general and open-ended questions (§5.2).

5.1 Which software infrastructure?

Distributed hypervisor. Obviously, the hypervisor of a disaggregated rack should be distributed, meaning that each resource blade runs a piece of the hypervisor. Building on existing OS paradigms, two main approaches can be envisioned for the construction of such a distributed hypervisor, namely multi-hypervisor (analog to multi-kernel [2, 3]) and split-hypervisor (analog to split-kernel [30]). In the former, the hypervisor is a collection of several complete hypervisors. All hypervisors have the same code base. The latter approach promotes a dedicated hypervisor per resource category (or even per resource model within a category), with possibly significant design differences between them. In both approaches, components communicate by message passing. We think that the split-hypervisor approach is the appropriate one, in particular because it facilitates support for hardware heterogeneity and local optimizations within a blade, which are key factors in a disaggregated environment.

Distributed guest OS. A guest OS running on a disaggregated architecture should be distributable, i.e., be able to span multiple blades of the same resource category, and especially CPU blades. For example, in such a design, the forked children of a main process could be transparently run on other CPU blades. This distribution is necessary (i) to support the provisionning of resource-heavy VMs on top of lightweight blades and (ii) to achieve high hardware resource usage (by overcoming resource fragmentation issues

[1] More precisely, LegoOS provides the concept of *vNodes* (virtual nodes). Each vNode has it own IP address and file-system mount point. This notion is actually more akin to a process group abstraction than to a low-level, hypervisor-based abstraction like the one we advocate.

that cannot be solely addressed via initial VM placement or full-VM migration decisions, e.g. to support "vCPU bursting" towards other CPU blades at small time scales). One may think that building a distributed OS in this context raises the same challenges as with the computer-centric architecture, as studied in the previous decades [1, 32], but a disaggregated context brings different characteristics. First, blade resources are heterogeneous (specialized blades and different technologies even for the same resource type). Second, some of the past approaches required the modification of user applications to integrate some OS tasks such as thread scheduling [3, 32]. Third, to the best of knowledge, none of them were able to support complete legacy/standard APIs like POSIX in addition to fine-grained resource allocation and strong isolation, limiting these OSes to the research area or to single-application or single-tenant contexts. To make disaggregation exploitable in production DC, the ideal goal is the execution of unmodified applications and commodity OSes in the distributed guest — we intend to achieve this via a specific form of paravirtualization within a Linux guest.

The next sections present in more details the above architectural principles, respectively for the hypervisor (§6) and the guest OS (§7). These ideas will be used to build a research prototype in order to explore the solution space for the long-term challenges described next (§5.2).

5.2 Long-term challenges

Heterogeneity. One of the main advantages of disaggregation is the fact that it allows the rapid integration of new technologies in the data center. Thus, a given rack is likely to be heterogeneous, even for the same resource type. For example, different CPU types (x86, ARM, etc.) can coexist. Also, different CPU models within the same type may support different feature sets, for instance, they may or may not support cache partitioning (e.g., Intel CAT [21]). The challenge here is twofold. First, how to make the hypervisor take into account this heterogeneity? We see above that this can be addressed by the split-hypervisor model. For example, the hypervisor piece which runs on a CAT-aware CPU will enable hardware cache partitioning at start time, while the hypervisor piece running on a non-CAT-aware CPU will use a software-based alternative such as page coloring [34]. The second issue is the provisioning of a homogeneous VM in this context. How to support the execution of a VM's vCPUs over different physical processor types? Current hypervisors are not able to support even this minimal heterogeneity. Also, the challenge of supporting efficient migration between (heterogeneous) blades should be considered.

Blade hot plug. Fine-grained and unbounded hot plugging of resources is another conceptual advantage of disaggregation. Therefore, the hypervisor should be able to dynamically manage the integration and removal of resources. This challenge is much more difficult to address than in current computer-centric designs because several constraints are significantly different. First, the infrastructure must efficiently support a very large number of resource modules that are fully interconnected. Second, it must support a large diversity of resources (which typically become more heterogeneous over the lifespan of a rack, and this lifespan is greater than the one of a single server). In contrast, current approaches are based on different assumptions: low-level solutions (BIOSes/OSes) are efficient but only support a fixed capacity and a limited set of resources that are known a priori, whereas higher-level solutions employed in distributed systems are more flexible but have more overheads and are not designed for the same time scales. Some of these issues might nonetheless be alleviated by leveraging recent approaches such as live hypervisor upgrade solutions [35].

Performance. A disaggregated architecture relies on a very low latency and very high bandwidth interconnect between blades. However, a large number of blades increases traffic, which could rapidly saturate the interconnect, thus resulting in low performance. This leads to an important design principle for the split-hypervisor: supporting virtualization features should not introduce a tight coupling between the different types of blades. Besides, the communication between CPU and memory blades is the most critical one. The question is how to guarantee certain performance levels for this path. Several solutions can be adopted including: (i) building CPU blades with high memory cache sizes (gigabytes) and optimizing the hypervisor to efficiently exploit this local cache [29, 30]; (2) building dedicated network links between CPU and memory blades; (3) allowing bandwidth reservations on a shared interconnect.

Reliability. Compared to the failure of an OS in a native system, the failure of a hypervisor in a virtualized data center has generally bigger consequences because more applications are impacted. When shifting to a disaggregated infrastructure, the consequences worsen, because the entire rack is impacted. In order to mitigate the impact of such failures, replication techniques can (and should) be used at various levels of the software stack. However, only relying on such techniques for a disaggregated infrastructure may raise cost-effectiveness issues. Therefore, it becomes important to investigate the design of techniques allowing the containment of (hardware and software) faults within a rack without negating the potential benefits of disaggregation.

Security. Security remains the top concern for cloud providers and tenants. A disaggregated infrastructure exacerbates several security-related challenges. First, compared to a traditional architecture, it is more complex, and thus has a greater attack surface. We believe that this problem

can be mitigated using techniques from dependable systems, such as microkernels and modular hypervisors [6]. Second, a disaggregated architecture contains many independent components embedding a mini-OS/firmware that must be upgradable but protected from hostile modifications. This can be achieved by generalizing (at the scale of a distributed architecture) the approach used in the AWS Nitro platform: using specific hardware to prevent guest code from having physical access to the firmware memory [25]. Third, (hardware and software) side channel attacks [11, 16] are currently very serious threats without any comprehensive countermeasure to date [10, 17], and are exacerbated in the context of a disaggregated architecture. In particular, the fast interconnect between blades introduces a very sensitive weakness. A possible (yet imperfect) mitigation strategy may consist in preventing the tenant VMs from easily making correlations between low-level architectural events; this could be achieved by (i) abstracting the view of the topology of disaggregated resources exposed to the tenant VMs and (ii) enforcing specific co-location policies (for example, preventing VMs that run on the same CPU blades to also share other resources, such as memory blades). Fourth, disaggregation also complicates the design and implementation of facilities aimed at protecting the integrity and/or confidentiality of the code and data of the tenants (e.g., attestation and secure enclave mechanisms), as they must enforce security and achieve efficient execution at the scale of a distributed and heterogeneous architecture.

Bare metal as a service. Current IaaS cloud providers offer the possibility to rent *bare-metal* instances for customers having very strong performance and/or security requirements. The transposition of this concept to a disaggregated rack is not trivial and raises a number of questions. For example, how can a provider flexibly sell a subset of the rack's disaggregated resources to a customer with performance/security isolation guarantees (for the tenants and the provider) that are at least similar to those of a computer-centric model? And what is the interface that should be used to make these resources accessible to a tenant (e.g., through a pre-configured OS image – either with a high-level interface like LegoOS [30] or a more exokernel-like interface)?

6 The NewVirt platform

This section describes a blueprint for the general architecture of the new virtualization platform that we envision to build as a research testbed, depicted in Fig. 2. It is expected to provide the following properties: (i) *efficient execution* of distributed guest VMs, leveraging hardware acceleration when possible, (ii) *flexibility* of configuration, allowing freedom of designs exploration for research purposes and customization for specific use cases, and (iii) *strong isolation* (for performance and security concerns) between guest and hypervisor domains. In particular, this architecture is compatible with the research directions suggested in §5.2.

In each rack of a virtualized disaggregated DC, we distinguish three blade categories: (i) *Resource Blades (RB)* provide resources to VMs; Examples of RBs are CPU, GPU, memory, network, disk blades. (ii) *Resource Blade Managers (RBM)*: an RBM manages RBs from the same resource type, e.g., CPU. (iii) The *Global Manager (GM)* controls RBMs, and thus manages the entire rack.

An RB includes *Resource Components (RComp)*, which represent resources that will be shared among VMs. For instance, the CPU blade includes a CPU package that contains cores, caches, etc. Notice that RComps may implement virtualization features, but such features should be related only to the component on the blade, not to a remote component. For instance, the implementation of a feature like Intel Page Modification Logging [20], currently within the CPU, should be implemented on the memory blade in this new architecture. In addition to RComps, an RB includes a programmable chip (FPGA in the figure), a controller (*Ctrl*), and optionally a set of accelerators (*Acl*, related to virtualization or not). The Ctrl is a dedicated processor that runs a piece of the software part of the hypervisor, called *Local Blade Manager (LBM)*. This way, the hypervisor does not share resources with the VMs, thus avoiding hypervisor interferences and security risks. Besides, the FPGA is at the heart of the RB architecture (all the other components are linked to it). It implements the hypervisor parts that require very high efficiency but also some flexibility, and that have moderate complexity amenable to FPGA synthesis. The code run by the FPGA monitors, configures, and drives the other components on the RB except the Ctrl. This way, the FPGA can implement, for instance, a vCPU overprovisioning scheduler on a CPU RB. It can also implement a working set size estimation algorithm, necessary for memory oversubscription, on a memory RB. All RBs communicate with each other using a fast interconnect. Components inside an RB can directly talk to another RB or can go through the FPGA before. The latter option allows the execution of a custom operation (e.g., setting a tag) by the FPGA before accessing the remote RB.

The *Local Blade Manager* (LBM), *Resource Blade Manager* (RBM) and *Global Manger* (GM) run the software part of the hypervisor. The GM is the entry point for managing the rack. The LBM has several roles. First, it runs the FPGA synthesizer. Second, it runs all the hypervisor's algorithms that are too complex to run in a FPGA (e.g., live migration to another rack, checkpointing). Finally it executes the code that configures RComps (e.g., partition sizing). The RBM plays two roles: configuring LBMs of a specific type (e.g., CPU) and hosting hypervisor algorithms that require a global view of a VM resource utilization. Concerning the latter, the RBM should run the top level of the CPU or memory overprovisioning

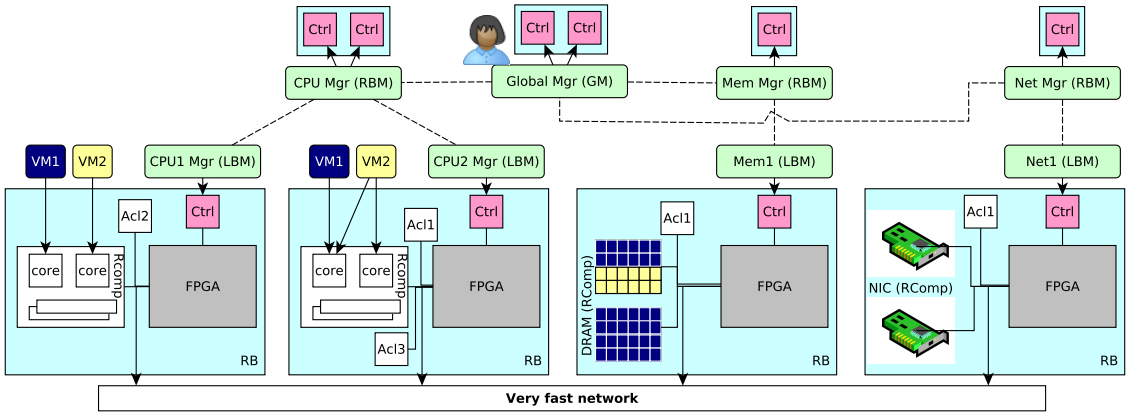

Figure 2. The NewVirt platform for rack-scale disaggregated virtualization.

algorithms because a VM's vCPUs or memory can be hosted by several blades (VMs 1 & 2 in the figure).

7 Distributed guest OSes

This section presents *Sprinkled OS*, a new design model for supporting distributed VMs on a disaggregated architecture, with the illusion of a single system image per VM. Note that this model is agnostic to the two disaggregation visions presented in §2 (heavy and lightweight blades). The main motivation of this model is to facilitate the porting of legacy guest code (initially designed for the computer-centric model) to a disaggregated architecture. More precisely, the goal is to support unmodified guest applications via a limited set of changes made to the code base of an existing guest OS (e.g., Linux). With this model, the distributed VM is a collection of several *sprinkled VMs*, each running on a CPU blade (see Fig. 3). The number of *sprinkled VMs* that compose a distributed guest OS equals the number of CPU blades hosting at least one vCPU of the distributed guest VM. A user who connects to a distributed guest VM is redirected to a *sprinkled VM* instance. The latter transparently sees all the hardware and software resources (e.g., CPU cores and processes, respectively) assigned to the entire distributed guest VM. The data structures of a *sprinkled OS* are organized in two types: real and stub. Real data structures directly reference virtual CPUs and processes that are currently mapped to the local CPU blade. Stub data structures represent remote entities; accessing them (e.g., for sending a signal to a process running on a remote blade) triggers a trap in the hypervisor (*Mgr* elements in Fig. 3), which then forwards the requests to the target remote *sprinkled OS* instance. We believe that this design facilitates the utilization of existing OSes to provide *sprinkled VMs*, with less modifications which can be automated using tools like Coccinelle [27].

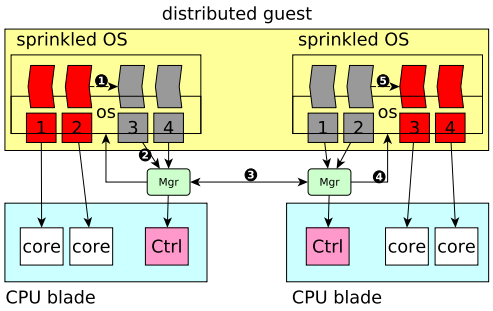

Figure 3. A distributed guest VM provided by two sprinkled OS instances. The figure only depicts the CPU blades.

8 Conclusion

As the vision of full hardware disaggregation in the data center is gaining traction, the implications on the software stack are in need of clarification. We have made the case for the persisting interest of virtualization in this context, and we have suggested design directions for the hypervisor and guest OSes. The exploration of hardware/software co-designs to address these challenges will likely be difficult but the blooming ecosystem of related tools also provides opportunities to succeed [8, 19, 22].

Acknowledgments

This work was funded by the "ScaleVisor" project of Agence Nationale de la Recherche, number ANR-18-CE25-0016, and the "Studio virtuel" project of BPI and ERDF/FEDER, grant agreement number 16.010402.01.

References

[1] A. Barak and R. Wheeler. 1988. *MOSIX: An integrated Unix for multiprocessor workstations.* TR-88-004. Univ. of Berkeley, California.

[2] Antonio Barbalace, Marina Sadini, Saif Ansary, Christopher Jelesnianski, Akshay Ravichandran, Cagil Kendir, Alastair Murray, and

Binoy Ravindran. 2015. Popcorn: Bridging the Programmability Gap in heterogeneous-ISA Platforms. In *Proceedings of the Tenth European Conference on Computer Systems* (Bordeaux, France) *(EuroSys '15)*. ACM, New York, NY, USA, Article 29, 16 pages. https://doi.org/10.1145/2741948.2741962

[3] Andrew Baumann, Paul Barham, Pierre-Evariste Dagand, Tim Harris, Rebecca Isaacs, Simon Peter, Timothy Roscoe, Adrian Schüpbach, and Akhilesh Singhania. 2009. The Multikernel: A New OS Architecture for Scalable Multicore Systems. In *Proceedings of the ACM SIGOPS 22Nd Symposium on Operating Systems Principles* (Big Sky, Montana, USA) *(SOSP '09)*. ACM, New York, NY, USA, 29–44. https://doi.org/10.1145/1629575.1629579

[4] Marc Brooker and Holly Mesrobian. 2018. A Serverless Journey: Under the Hood of AWS Lambda. https://www.youtube.com/watch?v=QdzV04T_kec. Online; accessed 2019-07-07.

[5] Amanda Carbonari and Ivan Beschasnikh. 2017. Tolerating Faults in Disaggregated Datacenters. In *Proceedings of the 16th ACM Workshop on Hot Topics in Networks* (Palo Alto, CA, USA) *(HotNets-XVI)*. ACM, New York, NY, USA, 164–170. https://doi.org/10.1145/3152434.3152447

[6] Patrick Colp, Mihir Nanavati, Jun Zhu, William Aiello, George Coker, Tim Deegan, Peter Loscocco, and Andrew Warfield. 2011. Breaking Up is Hard to Do: Security and Functionality in a Commodity Hypervisor. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles (SOSP '11)*. ACM, Cascais, Portugal, 189–202.

[7] dRedBox EU H2020 project. 2016. dRedBox: Disaggregated Data Center in a Box. http://www.dredbox.eu/. Online; accessed 2019-07-07.

[8] ETH Zürich Systems Group. 2019. The Enzian Research Computer. http://www.enzian.systems/. Online; accessed 2019-07-07.

[9] Peter X. Gao, Akshay Narayan, Sagar Karandikar, Joao Carreira, Sangjin Han, Rachit Agarwal, Sylvia Ratnasamy, and Scott Shenker. 2016. Network Requirements for Resource Disaggregation. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation* (Savannah, GA, USA) *(OSDI'16)*. USENIX Association, Berkeley, CA, USA, 249–264. http://dl.acm.org/citation.cfm?id=3026877.3026897

[10] Qian Ge, Yuval Yarom, Tom Chothia, and Gernot Heiser. 2019. Time Protection: the Missing OS Abstraction. In *EuroSys Conference* (2019-3-25). ACM, Dresden, Germany.

[11] Qian Ge, Yuval Yarom, David Cock, and Gernot Heiser. 2018. A Survey of Microarchitectural Timing Attacks and Countermeasures on Contemporary Hardware. *Journal of Cryptographic Engineering* 8 (April 2018), 1–27. Issue 1. https://doi.org/10.1007/s13389-016-0141-6

[12] Madeleine Glick, Sebastien Rumley, and Keren Bergman. 2018. Silicon Photonics Enabling the Disaggregated Data Center, In Advanced Photonics 2018 (BGPP, IPR, NP, NOMA, Sensors, Networks, SPPCom, SOF). *Advanced Photonics 2018 (BGPP, IPR, NP, NOMA, Sensors, Networks, SPPCom, SOF)*, NeM3F.4. https://doi.org/10.1364/NETWORKS.2018.NeM3F.4

[13] Google, Inc. 2019. gVisor. A container sandbox runtime focused on security, efficiency, and ease of use. https://gvisor.dev. Online; accessed 2019-07-07.

[14] Kinshuk Govil, Dan Teodosiu, Yongqiang Huang, and Mendel Rosenblum. 2000. Cellular Disco: Resource Management Using Virtual Clusters on Shared-memory Multiprocessors. *ACM Transactions on Computer Systems* 18, 3 (Aug. 2000), 229–262.

[15] Brendan Gregg. 2017. AWS EC2 Virtualization 2017: Introducing Nitro. http://www.brendangregg.com/blog/2017-11-29/aws-ec2-virtualization-2017.html. Online; accessed 2019-07-07.

[16] Daniel Gruss, Erik Kraft, Trishita Tiwari, Michael Schwarz, Ari Trachtenberg, Jason Hennessey, Alex Ionescu, and Anders Fogh. 2019. Page Cache Attacks. https://arxiv.org/abs/1901.01161. *arXiv preprint arXiv:1901.01161* (Jan. 2019).

[17] Gernot Heiser. 2018. For Safety's Sake: We Need a New Hardware-Software Contract! *IEEE Design and Test* 35 (March 2018), 27–30. Issue 2. https://doi.org/10.1109/MDAT.2017.2766559

[18] Joseph M. Hellerstein, Jose M. Faleiro, Joseph E. Gonzalez, Johann Schleier-Smith, Vikram Sreekanti, Alexey Tumanov, and Chenggang Wu. 2019. Serverless Computing: One Step Forward, Two Steps Back. In *Proceedings of the 2019 biennial Conference on Innovative Data Systems Research (CIDR '19)*. Asilomar, CA, USA.

[19] John Hennessy and David Patterson. 2018. A New Golden Age for Computer Architecture: Domain-Specific Hardware/Software Co-Design, Enhanced Security, Open Instruction Sets, and Agile Chip Development. Turing lecture given at ISCA '18. https://www.youtube.com/watch?time_continue=125&v=3LVeEjsn8Ts. Online; accessed 2019-07-07.

[20] Intel corporation. 2015. Page-Modification Logging for Virtual-Machine Monitor. https://www.intel.com/content/www/us/en/processors/page-modification-logging-vmm-white-paper.html. Online; accessed 2019-07-07.

[21] Intel Corporation. 2016. Introduction to Cache Allocation Technology in the Intel Xeon Processor E5 v4 Family. https://software.intel.com/en-us/articles/introduction-to-cache-allocation-technology. Online; accessed 2019-07-07.

[22] S. Karandikar, H. Mao, D. Kim, D. Biancolin, A. Amid, D. Lee, N. Pemberton, E. Amaro, C. Schmidt, A. Chopra, Q. Huang, K. Kovacs, B. Nikolic, R. Katz, J. Bachrach, and K. Asanovic. 2018. FireSim: FPGA-Accelerated Cycle-Exact Scale-Out System Simulation in the Public Cloud. In *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA '18)*. 29–42.

[23] Eric Keller, Jakub Szefer, Jennifer Rexford, and Ruby B. Lee. 2010. NoHype: Virtualized Cloud Infrastructure Without the Virtualization. In *Proceedings of the 37th Annual International Symposium on Computer Architecture* (Saint-Malo, France) *(ISCA '10)*. ACM, New York, NY, USA, 350–361. https://doi.org/10.1145/1815961.1816010

[24] Sergey Legtchenko, Hugh Williams, Kaveh Razavi, Austin Donnelly, Richard Black, Andrew Douglas, Nathanaël Cheriere, Daniel Fryer, Kai Mast, Angela Demke Brown, Ana Klimovic, Andy Slowey, and Antony Rowstron. 2017. Understanding Rack-scale Disaggregated Storage. In *Proceedings of the 9th USENIX Conference on Hot Topics in Storage and File Systems* (Santa Clara, CA) *(HotStorage '17)*. USENIX Association, Berkeley, CA, USA, 2–2. http://dl.acm.org/citation.cfm?id=3154601.3154603

[25] Anthony Liguori. 2018. Powering Next-Gen EC2 Instances – Deep Dive into the Nitro System. https://www.youtube.com/watch?v=e8DVmwj3OEs. Online; accessed 2019-07-07.

[26] Kevin T. Lim, Jichuan Chang, Trevor N. Mudge, Parthasarathy Ranganathan, Steven K. Reinhardt, and Thomas F. Wenisch. 2009. Disaggregated Memory for Expansion and Sharing in Blade Servers. In *Proceedings of the 2009 ACM/IEEE Annual International Symposium on Computer Architecture (ISCA '09)*. ACM, 267–278.

[27] LIP6. 2018. Coccinelle: A Program Matching and Transformation Tool for Systems Code. http://coccinelle.lip6.fr/. Online; accessed 2019-07-07.

[28] Martin Maas, Krste Asanović, and John Kubiatowicz. 2017. Return of the Runtimes: Rethinking the Language Runtime System for the Cloud 3.0 Era. In *Proceedings of the 16th Workshop on Hot Topics in Operating Systems (HotOS '17)*. ACM, Whistler, BC, Canada, 138–143.

[29] Vlad Nitu, Boris Teabe, Alain Tchana, Canturk Isci, and Daniel Hagimont. 2018. Welcome to Zombieland: Practical and Energy-efficient Memory Disaggregation in a Datacenter. In *Proceedings of the Thirteenth EuroSys Conference (Eurosys '18)*. ACM, Porto, Portugal, 16:1–16:12.

[30] Yizhou Shan, Yutong Huang, Yilun Chen, and Yiying Zhang. 2018. LegoOS: A Disseminated, Distributed OS for Hardware Resource Disaggregation. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation (OSDI '18)*. USENIX Association, Carlsbad, CA, USA, 69–87.

[31] Zhiming Shen, Zhen Sun, Gur-Eyal Sela, Eugene Bagdasaryan, Christina Delimitrou, Robbert Van Renesse, and Hakim Weatherspoon. 2019. X-Containers: Breaking Down Barriers to Improve Performance and Isolation of Cloud-Native Containers. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '19)*. ACM, Providence, RI, USA, 121–135.

[32] Andrew S. Tanenbaum, Robbert van Renesse, Hans van Staveren, Gregory J. Sharp, and Sape J. Mullender. 1990. Experiences with the Amoeba Distributed Operating System. *Commun. ACM* 33, 12 (Dec. 1990), 46–63. https://doi.org/10.1145/96267.96281

[33] Liang Wang, Mengyuan Li, Yinqian Zhang, Thomas Ristenpart, and Michael Swift. 2018. Peeking Behind the Curtains of Serverless Platforms. In *2018 USENIX Annual Technical Conference (USENIX ATC '18)*. USENIX Association, Boston, MA, 133–146.

[34] Ying Ye, Richard West, Zhuoqun Cheng, and Ye Li. 2014. COLORIS: A Dynamic Cache Partitioning System Using Page Coloring. In *Proceedings of the 23rd International Conference on Parallel Architectures and Compilation* (Edmonton, AB, Canada) *(PACT '14)*. ACM, New York, NY, USA, 381–392. https://doi.org/10.1145/2628071.2628104

[35] Xiantao Zhang, Xiao Zheng, Zhi Wang, Qi Li, Junkang Fu, Yang Zhang, and Yibin Shen. 2019. Fast and Scalable VMM Live Upgrade in Large Cloud Infrastructure. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '19)*. ACM, Providence, RI, USA, 93–105.