Optimized Resource Allocation on Virtualized Non-Uniform I/O Architectures

Dinh Ngoc Tu, Boris Teabe, Daniel Hagimont, Georges da Costa

Optimized Resource Allocation on Virtualized Non-Uniform I/O Architectures

Tu Dinh Ngoc, Boris Teabe, Daniel Hagimont, Georges Da Costa
IRIT, Université de Toulouse, CNRS, Toulouse INP, UT3
Toulouse, France

***Abstract*—Nowadays, virtualization is a central element in data centers as it allows sharing server resources among multiple users across virtual machines (VM). These servers often follow a *Non-Uniform Memory Access* (NUMA) architecture, consisting of independent nodes with their own cache hierarchies and I/O controllers. In this work, we investigate the impact of such an architecture on network access.**

As network devices are typically connected to one particular NUMA node, this leads to a situation where device access on one node is faster than another. This phenomenon is called *Non-Uniform I/O Access* (NUIOA). This non-uniformity impacts the performance of I/O applications that are not executed on the correct NUMA node.

In this paper, we are interested in NUIOA effects in virtualized environments. Our contribution in this work is twofold: 1) we thoroughly study the impact of NUIOA on application performance in VMs, and 2) we propose a resource allocation strategy for VMs that reduces the impact of NUIOA. We implemented our allocation strategy on the Xen hypervisor and carried out evaluations with well-known benchmarks to validate our strategy. The obtained results show that with our NUIOA allocation scheme, we can improve the performance of application in VMs by up to 20% compared to common allocation strategies.

***Index Terms*—I/O virtualization, NUMA, NUIOA**

I. Introduction

Virtualization has become the key to increasing hosting density and efficiency in modern datacenters by efficiently and securely sharing the same set of hardware resources among multiple users. Virtualized resources are managed by a software called the *hypervisor* and exposed to users under the form of *virtual machines* (VMs). Modern hypervisors are capable of utilizing hardware features such as the *I/O memory management unit* (IOMMU) [1] to grant VMs direct access to a physical device, in a process called *device passthrough*. When supported, device passthrough provides VMs with the lowest virtualization overhead and the best possible performance [2], and therefore is the focus of our current work. On current server platforms, hardware components are often connected to each other via a *Peripheral Component Interconnect Express* bus, called PCIe for short. Notable components connected using the PCIe bus include network adapters, storage cards/adapters, GPUs and so on (see Fig 1a for an example); PCIe has even been adapted as a cluster interconnect bus, such as the *Compute Express Link* (CXL) standard [3].

The cost of NUMA designs. On architectures where the processor is connected to devices using a single bus link (which is often the case for single-socket systems with uniform memory access, SMP), all CPU cores share the same I/O path to a certain device (blue arrow on Fig 1a). However, with ever-increasing demands for processing power and hosting density, current servers can come with multiple processors, each with its own CPU cores, memory hierarchy and I/O link organized as an independent node (see Fig 1b). These nodes are then linked with a fast *NUMA interconnect*, which can be PCIe itself or a proprietary interconnect (e.g. Intel UPI). This independent node architecture is often called *Non-Uniform Memory Access* (NUMA) [4], but its implications extend beyond memory access, as we will demonstrate below. I/O devices such as network cards and storage drives are most of the time furnished with a single PCIe link, and as a result each device is affiliated with a single NUMA node, which we call the *home node* of that device. It follows that any communications (e.g. register writes, interrupts, DMA operations) between a device and a node that is not its home node (or *remote node* for short) would suffer extra overhead, due to them needing to cross the NUMA interconnect between nodes. This situation is called *Non-Uniform I/O Access* [5], or **NUIOA** for short. NUIOA not only affects I/O latency due to the longer signal path, but may also affect the maximum available I/O bandwidth when the workload is constrained by interconnect bandwidth. Therefore, for I/O intensive workloads to achieve optimal performance under NUIOA architectures, the tasks performing I/O (in both kernel-mode and user-mode) need to be located on the corresponding device's home node (see Fig 1b).

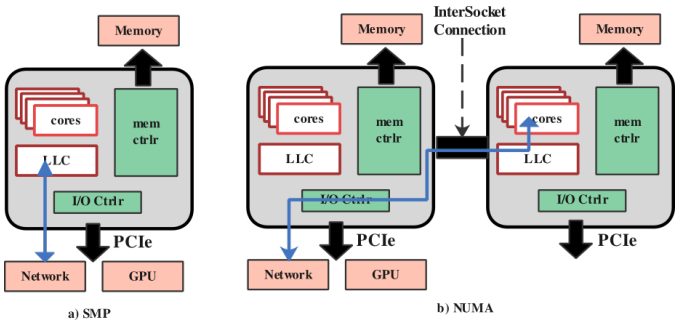

Fig. 1. I/O in a server based on a a) UMA (i.e. non-NUMA) architecture; b) NUMA architecture.

Hypervisors are still (mostly) not NUIOA-ready. Current OSes and hypervisors already support NUMA-aware scheduling [6], [7], i.e. they take into account NUMA distance costs (between memory and CPU) when scheduling

 DOI: 10.1109/CCGrid54584.2022.00053

multiple threads. However, NUIOA scheduling is far less often explored; while hypervisors are capable of acquiring home node information for each device (e.g. by using the proximity domain information exposed by ACPI), they do not use this information by default when scheduling VMs. As a consequence, a VM configured with device passthrough might be scheduled on a remote node, leading to lower performance than one might expect. Thus, two VMs with the same characteristics may have different I/O performance simply because one VM has direct access to the device's home node and the other not, leading to a *performance unpredictability*. This performance unpredictability is undesirable both in terms of impacting the SLA provided to consumers, whether internal (for private clouds) or external (for public clouds), and in terms of resource utilization and VM density. Certain solutions like OpenStack [8] make efforts to optimize their VMs for NUIOA; however, they are limited to scheduling whole guests on the corresponding home node, without targeting the applications that live inside these VMs [9]. Evidently, it is desirable to have an allocation strategy that takes into account NUIOA effects and remedies the issues coming from remote I/O operations.

Our contribution in this paper is twofold: (1) we carry out an exhaustive study of NUIOA impact on VM I/O performance; and (2) based on our findings, we propose a novel VM resource allocation strategy on NUIOA systems.

Study of NUIOA impact on performance. We first carried out I/O performance evaluations of VMs hosted by the Xen hypervisor. We experimented with various I/O workloads with different resource allocation configurations under two types of connections, namely Ethernet and InfiniBand. We focused on three different aspects of performance impacts caused by NUIOA architectures: (1) latency impacts caused by remote NUIOA accesses; (2) interconnect bandwidth limitations caused by neighboring workloads; and (3) the overhead caused by virtual NUMA configurations on VM performance. We found that as expected, I/O performance on VMs is optimal when they are located on the corresponding device's home node; moreover, the impact of NUIOA is especially relevant concerning high-speed networking, especially when coupled with an efficient I/O stack.

NUIOA-aware VM allocation scheme. Our second contribution is a NUIOA-aware VM allocation scheme (*NUIOA allocator* for short) for combating NUIOA effects. The basic idea behind our NUIOA-aware allocator is threefold. Firstly, we ensure that each VM assigned with a device is also provided with one part of the home node's CPU and memory resources. Secondly, we inform each VM of this association between device and resources by exposing a virtual NUMA (vNUMA) topology. Finally, we optimize the scheduling of I/O applications in VMs by locating them on home nodes to avoid any NUIOA effects while avoiding oversubscribing or wasting resources. We experimented with our NUIOA-aware allocation strategy on Xen; our results show that the NUIOA allocator improves application performance by up to 20% compared to resource allocation strategies in Xen.

The remainder of our paper is organized as follows. In Section II, we present the necessary background to understand our contributions, including the basic forms of virtualization, as well as the role of NUMA in virtualizing I/O devices. In Section III, we study existing software and hardware solutions that tackle the NUIOA problem, and discuss the various tradeoffs relevant to each solution. Section IV presents an evaluation of an I/O heavy application with multiple different NUIOA setups; from our evaluation, we observe which factors influenced NUIOA effects, and therefore I/O virtualization performance. Section V builds on our aforementioned observations to establish our contribution, a NUIOA-aware VM resource allocation strategy that involves a hypervisor-layer NUIOA allocator and a workload scheduling methodology. We also present the implementation of our resource allocation strategy in detail, and provide some discussions on potential venues for improvement. In Section VI, we show how our NUIOA-aware VM resource allocation strategy minimizes the NUIOA penalty of virtualized I/O in comparison to various other setups, and discuss the potential overhead of our solution. Finally, Section VII concludes our paper.

II. Background

In this section, we present concepts relevant to our contribution, including virtualization and the importance of NUMA/NUIOA topologies.

A. Virtualization

Virtualization technology allows the creation of multiple virtual machines (VMs) on a single physical host using a software called the hypervisor. Hypervisors distribute I/O services (virtual storage, virtual networking, etc.) to VMs via one of three main ways, ordered by increasing levels of performance: by emulating a real hardware device; by providing an optimized virtual I/O interface to the VM (also known as *paravirtualization*); or by completely delegating access to a physical device to the VM using *device passthrough*. In this work, we focus on device passthrough in the context of NUMA systems, the particularities of which we detail below.

B. NUMA and its relevance to I/O virtualization

Modern servers tend to use a NUMA architecture because it allows increasing the number of cores and memory available on the servers. A NUMA architecture consists of several independent nodes, each containing a subset of CPU cores and memory; an example of which is shown on Figure 1.

NUMA in virtualization. Hypervisors often use one of two approaches for handling VM resources on NUMA architectures: (1) memory interleaving and (2) vNUMA.

Memory interleaving is the default allocation strategy on hypervisor such as Xen [10]. It consists of allocating the VM memory by regions of 1 GB with a round-robin algorithm on each NUMA node, then presenting a Uniform Memory Architecture (UMA) to the VM.

vNUMA involves presenting to the VM a virtual NUMA topology which maps its resources to virtual NUMA nodes. It is supported by modern hypervisors such as Xen, VMware and

Hyper-V. vNUMA can take advantage of existing operating systems' NUMA awareness to improve VM guests' CPU and memory allocation locality.

Locality of devices and virtualization. With ever-increasing performance of I/O devices and system buses such as PCI Express, servers are often equipped with only a single device of each I/O type. In other words, each server would have one network adapter, one NVMe storage device, and so on. Thus, I/O application performance both inside and outside VMs depend on the locality of the application in regards to the device it interacts with. By default, hypervisors do not relocate applications depending on their NUIOA affinity, leading to wasted performance in case of applications that perform direct device I/O (e.g. userspace drivers, RDMA applications, etc.) Moreover, some current hypervisors like Xen do not expose device locality to guest OSes, therefore causing VMs to be unaware of this association and unable to make the appropriate scheduling decisions.

Studies on NUMA in virtualized environments mostly target resource allocation of VMs on NUMA architectures [11], [12], or the scheduling of VM vCPUs on NUMA [10], [13]. These works focused mostly on CPU and memory locality and put aside device locality. In our work, we focus on the question of device locality and NUIOA in the context of virtualization.

III. Related work

Several research works have focused on the problem of NUIOA management, both in virtualized and non-virtualized environments. These works can be classified into two main categories: software- and hardware-based implementations. We detail examples of both implementation categories below.

A. Software solutions

Several studies propose software solutions to manage NUIOA mainly in non-virtualized environments. The existing state-of-the-art software solutions involve either (1) recommending users to manually pin I/O-intensive applications to their device's home node to ensure locality; or (2) implementing an automatic pinning mechanism [5], [14]–[17], combined with migrating threads away from their home node as needed for load balancing purposes [18]. Other studies investigate the implementation of NUIOA-aware schedulers [18], [19] that extend existing NUMA-aware schedulers to include PCIe locality constraints. While these solutions can manage the NUIOA problem, as we stated in Section I, some current hypervisors like Xen do not expose NUIOA locality to VMs, making these solutions inappropriate in a virtualized context.

Other works study I/O performance on NUMA architectures, particularly by investigating the resource allocation of VMs and its I/O services (e.g. privileged domains that provide I/O to other VMs) [11], [12]. These works mostly focus on colocating these privileged and user domains on the same physical NUMA nodes to ensure locality for these services; however, they do not study the problem of NUIOA in detail.

Existing systems like Red Hat Enterprise Linux or OpenStack provide recommendations for NUIOA performance optimization by allocating I/O-heavy VMs entirely on their home node [9], [20]. While this solution ensures optimal I/O performance, it also leads to processor/memory resource contention in cases where multiple VMs share the same device, where multiple I/O devices are connected to the same node, or where one VM is connected to multiple different devices. In other words, a simple 1:1 relation between VM:device restricts the scalability of such a solution.

B. Hardware solutions

Various works propose hardware-level solutions for mitigating NUIOA effects. Works from VMware [17], Squyres [21], and Moreaud et al. [22] propose a multihomed approach by equipping each NUMA node with its separate I/O devices, therefore ensuring that applications on each node will have access to its own local devices. While effective at mitigating NUIOA effects, this approach requires significant extra investment for each class of device (extra network cards, storage devices, GPUs...), leading to higher costs, higher energy consumption and lower hardware utilization. Additionally, as pointed out by Smolyar et al. [23], installing multiple devices is not a complete solution since it cannot ensure optimal performance during thread migration, and increasing I/O performance means that modern datacenters are trending towards providing a single NIC/disk/GPU per server.

Solutions such as Mellanox SocketDirect NICs [24] and IOctopus [23] propose to avoid NUIOA effects by connecting the same device to multiple NUMA nodes through a multi-interface hardware design combined with hardware-level switching support. While promising, these solutions require the use of multi-interface devices, which comes with additional costs compared to single-slot hardware, and utilizes extra PCIe slots that could otherwise be used with other devices.

C. Positioning of our work

Our work focuses on the problem of NUIOA scheduling, which has yet to receive widespread attention compared to (processor and memory-level) NUMA scheduling. In contrast to other works on NUIOA, we propose an unique scheme for optimizing resource allocation of I/O workloads that reduces the effect of NUIOA without requiring hardware modifications, while preserving the scalability benefits of NUMA architectures namely by not requiring I/O workloads to be wholly allocated on a single NUMA home node.

IV. Impact of NUIOA on VM performance

A. Methodology

In this section, we first evaluated various I/O workloads to see how NUIOA impacts application performance.

Hardware setup. Experiments ran on a cluster of Dell PowerEdge R630 servers, the configurations of which are detailed in Table I. Each server is equipped with two processors, and is therefore divided into two NUMA nodes numbered 0 and 1; both the Ethernet and InfiniBand devices are connected to node 0 of each server, or the **home node**.

Software setup. To measure I/O performance, we used the Sockperf benchmark tool [25]. We used Sockperf in one of two

TABLE I
HARDWARE CONFIGURATIONS USED FOR NUIOA EVALUATIONS.

Component	Characteristics
CPU	2x Intel Xeon E5-2630 v3 (8 cores per node)
Memory	128 GB (64 GB per node)
Ethernet adapter	Intel 82599ES 10 Gbps (node 0)
InfiniBand adapter	Mellanox MT27500 56 Gbps (node 0)
Storage	600 GB HDD

modes depending on what we wanted to examine: "latency under load" mode which measures packet latency under a certain network load level, and therefore is affected by NUIOA latency; and "throughput" mode which measures the maximum throughput delivered by the network card and is therefore affected by NUIOA bandwidth effects. For the purposes of virtual NUIOA evaluations, we deployed Sockperf on a VM running Ubuntu 20.04 on top of Xen [26] version 4.11. Each VM is equipped with 6 vCPUs, 8 GB of memory, and two network interfaces: one Ethernet and one Infiniband, both working in device passthrough mode. For each benchmark, the VM connects to a secondary machine with the same hardware configuration running bare-metal Linux.

Experiment details. As explained in Section I, NUIOA effects originate from the need for I/O operations to cross the NUMA interconnect. To see how these effects influence I/O-heavy applications, we set up the following experiments:

1) **NUIOA latency effects.** In this experiment, we run Sockperf in latency mode while varying the locality of the VM with regards to the network device's home node in order to investigate how NUIOA affects I/O latencies.
2) **NUIOA with bandwidth contention.** In this experiment, we discover how competing NUMA interconnect traffic influences NUIOA effects on application performance. We use a competing *neighbor* application to add load to the interconnect.
3) **vNUMA configurations.** In this experiment, we study the impact of NUMA configuration for the VM on I/O performance with NUIOA.

We define the configurations we used for the aforementioned experiments in Table II. Each row of the table corresponds to a distinct VM resource configuration. For example, in the first listed configuration named A^0_{Eth}, we provisioned a VM with its resources allocated on physical node 0 and a passthrough Ethernet adapter, with no interfering neighbor; moreover, we allocated the privileged domain (Xen Dom0) entirely on node 0. Note that since our VMs use the devices in passthrough mode, the privileged domain does not interfere in the communication path with the devices. Thus, all the results obtained would have been similar if the privileged domain resources were allocated on node 1.

For each configuration, we measured median network latency with Sockperf in both directions: reception (Rx) and transmission (Tx) while varying the network load (message size and rate). For the evaluation on Ethernet, we used the default Linux TCP stack as it is the case with most applications. With InfiniBand configurations, we set up Sockperf with libvma 9.0.2 [27] and MLNX_OFED 4.9 drivers to take advantage of the provided kernel-bypass features as is often done for HPC workloads. We repeat each experiment five times; as we observe low standard deviation among these runs, we report the average results.

TABLE II
VARIOUS CONFIGURATIONS AND ASSOCIATED ACRONYMS. (SEE SECTION IV-B)

Name	Node	Device	Dom0	Neighbor
A^0_{Eth}	Node 0	Ethernet	Node 0	No
A^0_{Ib}	Node 0	Infiniband	Node 0	No
A^1_{Eth}	Node 1	Ethernet	Node 0	No
A^1_{Ib}	Node 1	Infiniband	Node 0	No
N^0_{Ib}	Node 0	Infiniband	Node 0	Yes
$vN^{0,1}_{Ib}$	Nodes 0 & 1	Infiniband	Node 0	No
$U^{0,1}_{Ib}$	Nodes 0 & 1	Infiniband	Node 0	No

B. Experimental results

We consider one ideal configuration (A^0_{Eth} on Ethernet and A^0_{Ib} on Infiniband) as the reference configuration ref; for each other configuration X, we calculate the performance impact deg using the following formula for latency (lat) and bandwidth (bw):

$$deg_{lat} = \frac{lat_X - lat_{ref}}{lat_{ref}} \quad (1)$$

$$deg_{bw} = \frac{bw_{ref} - bw_X}{bw_{ref}} \quad (2)$$

with lat_X and bw_X being the latency and bandwidth results of configuration X, respectively.

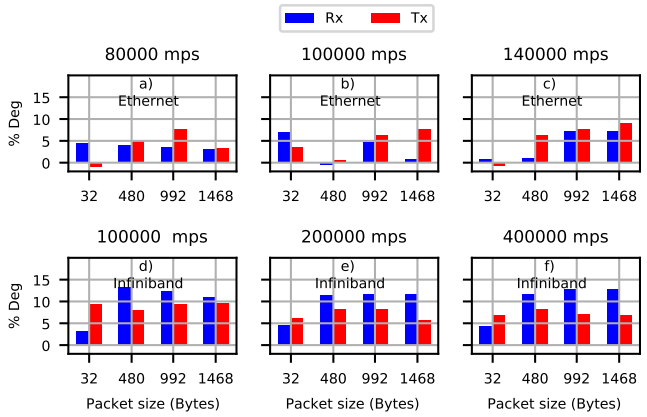

Fig. 2. NUIOA latency effects on Sockperf performance for A^0_{Eth} versus A^1_{Eth} and A^0_{Ib} versus A^1_{Ib} in function of messages per second (mps) and packet size. Rx and Tx stand for reception and transmission respectively.

NUIOA latency effects. To assess the impact of the NUMA interconnect distance on I/O performance, we compare the network latency and bandwidth figures obtained when a single VM is allocated on the home node versus when it is allocated on the remote node. This corresponds to comparing the performance of A^0_{Eth} versus A^1_{Eth} for Ethernet devices,

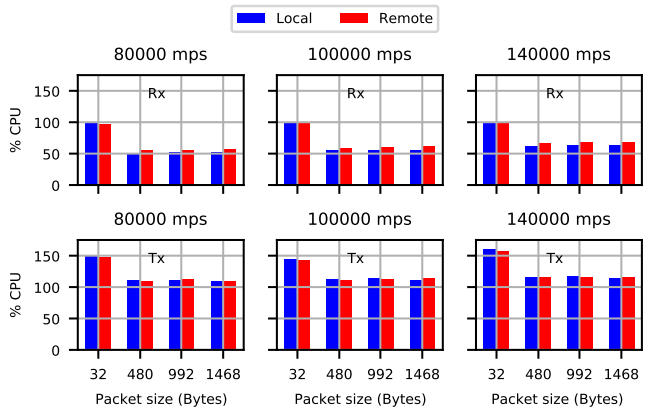

Fig. 3. CPU consumption for A^0_{Eth} versus A^1_{Eth} on Ethernet.

and A^0_{Ib} versus A^1_{Ib} for the InfiniBand device respectively. Fig 2 presents NUIOA latency impacts on both Ethernet and InfiniBand devices; similarly, Fig 4 presents NUIOA bandwidth impacts.

For Ethernet devices (first row of Fig 2), we observe that these devices experience relatively consistent NUIOA performance degradation, at around 7% for both transmission and reception regardless of packet rate. This is explained by the longer I/O path of Sockperf with Ethernet devices, which involves the Linux kernel's network stack and therefore serves to mask NUIOA effects. In fact, without a kernel-bypass solution, Sockperf on Ethernet shows little to no bandwidth impact from NUIOA at any setting (see Fig 4a).

Concerning CPU usage with Ethernet (first row of Fig 3), we observe fairly equal CPU consumptions between the two configurations on the transmit path, with higher CPU usage while sending 32-byte packets. Conversely, on the receive path, we observe a small (approximately 5%) but consistent difference in CPU consumption between the two configurations at packet sizes higher than 32 bytes; this can be explained by the higher cost of cross-node memory copies needed for packet reception.

In the case of InfiniBand networking, Sockperf communicates directly with the device through the use of libvma-based kernel-bypass; the optimized I/O path of libvma allows us to observe NUIOA effects in more detail. Firstly, we observe that Sockperf's CPU consumptions under all configurations are identical, since Sockperf on InfiniBand makes use of active polling to minimize latency. Note that high-speed Ethernet devices are still subject to the same NUIOA effects, especially with optimized I/O stacks like DPDK (Data Plane Development Kit) or libvma (Mellanox's Messaging Accelerator). In general, we observe an approximately 13% latency degradation on the receive path, compared to 6-10% on the transmission path. This is explained by an extraneous memory copy on the Rx path resulting from zero-copy receives being disabled in Sockperf. This justifies the lower Rx performance impact with 32-byte packets. Regardless, we still observe an effect on packet latency on the Tx path, which holds true regardless of message size and rate, suggesting that this impact is inherent to NUIOA and not dependent on memory bandwidth usage.

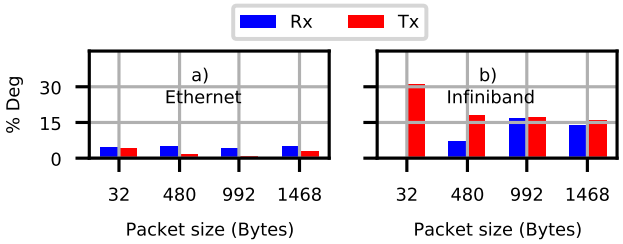

Fig. 4. Socket-interconnect impact on Rx/Tx bandwidth.

vNUMA configurations. VMs that span multiple physical NUMA nodes are typically unaware of the underlying physical topology, and therefore are not able to make appropriate scheduling decisions. Hypervisors can remedy this issue in using one of two approaches as described in Section II: (1) interleaving the memory of the VM between the allocated nodes and presenting a UMA architecture to the VM (the default on Xen); or (2) exposing a virtual NUMA topology that mirrors the underlying topology used to allocate the VM's resources. In our experimental configurations, we allocated 50% of the VM's resources on each physical NUMA node.

We started by comparing the default interleaved configuration $U^{0,1}_{Ib}$ (i.e. without vNUMA) to the reference configuration A^0_{Ib}. The first row of Fig 5 shows the recorded latency. We observe nearly the same or even worse performance compared to the previous experiment (13% for Rx and 16% for Tx) even though parts of the VM are located in the network device's home node. This is explained by the VM not having knowledge of NUMA topologies nor of device locality, resulting in I/O applications being scheduled on a remote node, combined with remote NUMA accesses caused by the lack of memory/CPU locality.

Next, we studied Sockperf performance on VMs with vNUMA enabled (see configuration $vN^{0,1}_{Ib}$ of Table II). We configured our VM with two virtual NUMA nodes, each equipped with 4 GB of RAM and 3 vCPUs. We present our results in the second row of Fig 5. We observe no performance degradation on the Rx path compared to the reference configuration, since the VM is capable of utilizing the vNUMA topology to avoid remote memory accesses during packet copies; however, the Tx path shows significant performance degradation of up to 20%, in the same fashion as seen in the UMA configurations presented above. We observe that in this configuration, neither Xen nor the guest take into account NUIOA effects while scheduling tasks; as a result, the Tx thread of Sockperf is occasionally scheduled onto a remote node, causing performance degradation.

C. Lessons learned

Following the aforementioned experiments, we list our observations of I/O application performance running on VMs under NUIOA conditions.

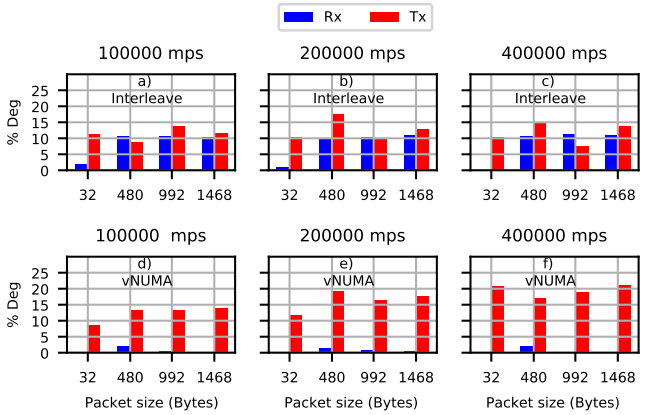

Fig. 5. Impact of interleaved ($U_{Ib}^{0,1}$) and vNUMA ($vN_{Ib}^{0,1}$) configurations on Sockperf latency compared to the reference A_{Ib}^{0}.

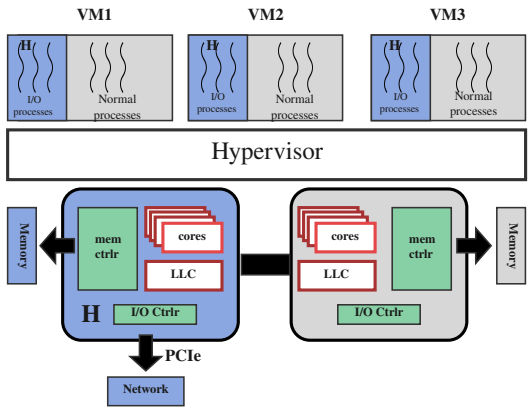

Fig. 6. Overview of our NUIOA-aware allocation strategy. I/O processes of each VM are allocated on the home node H (colored in blue)

1) As expected, I/O applications reach maximum performance when they are located on the corresponding device's home node.
2) Performance impacts caused by NUIOA effects are most notable with high-speed I/O, such as InfiniBand devices with kernel-bypass libraries.
3) NUMA interconnect utilization by neighboring workloads has little impact on I/O performance.
4) While a UMA VM configuration can cause performance degradation of I/O applications, vNUMA can serve to improve performance under certain conditions (as seen in the vNUMA Rx benchmarks).

V. NUIOA-aware VM resource allocation strategy

In this section, we present our resource allocation approach for mitigating the impacts of NUIOA effects on I/O workloads.

Overview. Our allocation strategy relies on the observation that ideally, the best resource allocation strategy for I/O applications is to place all of them on their corresponding home node. However, CPU and memory constraints (e.g. physical core counts and memory module sizes limits) mean that this strategy is not always applicable in practice. We work around this issue by ensuring that each aforementioned VM is allocated parts of the resources belonging to its device's home node. In other words, if a network device is connected to node H then any VM with I/O applications using that device would receive parts of the resources of node H, as illustrated in Fig 6. Additionally, we ensure that I/O workloads are scheduled on the correct node for its device to avoid any associated NUIOA costs. We describe below our allocation strategy in detail.

Hypervisor-layer resource allocation. Our first contribution is a hypervisor-side allocator that ensures I/O-heavy VMs are provided with resources backed by an appropriate NUMA node. We first gather information about the system's NUIOA topology, including the relationship from each PCI device to the NUMA node it is attached to. The system administrator defines each VM as being I/O-heavy by configuring device passthrough for that VM. For each such VM, we configure its resources allocation using vNUMA to provide it with parts of the NUMA node that its passthrough device is attached to. If the VM is not declared as an I/O-heavy VM, we simply fall back to the default allocation behavior. Otherwise, given a VM with a memory size of m and n vCPUs, we calculate the memory and CPU resources m_H and n_H allocated to the VM on its home node H using the following formulas:

$$m_H = \left\lfloor \frac{M_H^f * m}{M^f} \right\rfloor \tag{3}$$

$$n_H = \left\lfloor \frac{N_H^f * n}{N^f} \right\rfloor \tag{4}$$

where M_H^f and N_H^f are the free memory and CPU resources of the home node, and M^f and N^f are the free memory and CPU resources of the whole server, respectively. In short, the proportion of NUMA home node resources allocated to each I/O VM is proportional to the VM's total size.

NUIOA workload scheduling. As we observed in Section IV, simply configuring the VM with vNUMA is insufficient to ensure optimal I/O performance. Instead, both hypervisor and VM levels of scheduling need to be NUIOA-aware. We first identify I/O-heavy workloads running on the host; we propose three approaches for identifying these workloads, the details of which are described in Section V-A. After identifying I/O-heavy workloads, we ensure that they are prioritized to run on the I/O home node whenever possible.

A. Implementation

We implemented our NUIOA-aware allocator on Ubuntu 20.04 and Debian 10 running on top of Xen 4.11. Our prototype consists of a set of userspace tools for topology and workload discovery. These tools interact directly with existing libraries hypervisors and OS interfaces, meaning they can be easily reused across different hypervisor technology stacks, or integrated into virtualization platforms such as OpenStack.

Hypervisor-layer implementation. On the hypervisor, our allocator mainly involves detecting existing resources and their

relative topologies. For example, for each device concerned by NUIOA allocation, we examine its hardware topology, including IOMMU group information as exposed by the hypervisor as well as the affinity to its home NUMA node. Next, we correlate the association of NUIOA devices to VMs, also using existing hypervisor APIs. We can directly detect Xen device passthrough by using Xen's libxl facilities, or detect VFIO (Virtual Function I/O)-based device passthrough by probing open VFIO device files, then cross-referencing them with their IOMMU group information.

VM implementation. Upon VM boot, we first gather information about the virtual PCI topology, in order to discover the NUMA node affinities of each NUIOA passthrough device. Following our observations in Section IV that kernel-bypass solutions are most impacted by NUIOA effects both in terms of latency and throughput, we therefore focus our detection of I/O-heavy applications on detecting the presence of the aforementioned kernel-bypass platforms:

1) If a workload relies on libraries that perform direct device communication (e.g. libvma), we automatically assume this workload to be NUIOA-prone. Detection of these libraries is simply done by scanning each process's memory mapping (i.e. /proc/pid/maps)
2) We use techniques similar to hypervisor-layer NUIOA detection on I/O applications. Namely, DPDK applications can utilize kernel drivers such as UIO, VFIO or specific drivers (e.g. mlx4-core) for direct hardware communication; these drivers are detectable using the same technique as previously described.

Once we identify any NUIOA-prone workloads running on the system, we configure vNUMA on the VMs hosting these workloads, and then assign them a NUIOA affinity using OS-level tools like numactl. To maximize resource usage across all NUMA nodes while avoiding oversubscribing a single NUMA node when hosting many I/O applications, we use a *soft-affinity* scheme, where the NUIOA-prone workload is pinned to its home node when there's sufficient resources (free CPU time, free memory) to fit the workload in question. Otherwise, the workload is allowed to span multiple NUMA nodes and make use of the resources it requested.

B. Discussion

As stated in Section II, vNUMA is the main mechanism for exposing NUMA/NUIOA topology information to VMs. This topology information is prone to changes during the lifetime of the VM, either for resource utilization fairness and optimizations implemented by the hypervisor (e.g. CPU load balancing, VM live migration, memory ballooning), or by the use of certain communication mechanisms (e.g. page flipping). However, existing hypervisors and guest OSes are not designed to handle changes to their NUMA topologies, often requiring a reboot and redetection of the exposed topology.

To avoid the issue of topology changes, our approaches could be combined with those of Bui et al. [13] and Gauthier et al. [10] that propose dynamic vNUMA mechanisms which can be updated during VM execution. In particular, as a VM's underlying topology is changed by these operations, the hypervisor can provide the VM with topology updates that trigger reevaluation of workloads running inside the VM for NUIOA properties, and relocate them if necessary.

VI. Evaluation

This section covers the evaluation of our NUIOA-aware allocation strategy. Our goal is to estimate the impact of our strategy on application performance. For this, we used a set of benchmarks as described in Table III. We examined several VM allocation schemes with regards to NUMA topologies:

1) VM fully allocated on the device's home node (our best-case, reference configuration);
2) VM fully allocated on the remote node (noted *Remote*);
3) VM split between both the home and remote nodes (noted *UMA*);
4) VM split between both the home and remote nodes, vNUMA enabled (noted *vNUMA*);
5) VM split between both the home and remote nodes, vNUMA enabled, with our NUIOA-aware allocation (noted *NUIOA-aware*).

We provided the VM used in our experiment with 6 vCPU, 8 GB of RAM; additionally, we allocated the privileged domain's resources entirely on the machine's node 0. If not otherwise specified, our experimental environment (software and hardware) is identical to that of Section IV.

TABLE III
List of Benchmarks Used for NUIOA Evaluation.

Benchmark (metric)	Description
Sockperf (Latency, Bandwidth)	See Section IV.
Perftest [28] (Latency)	Generate a synthetic stream of RDMA operations: read, send, write, atomic.
Memcached [29] (Transactions per second)	Remote client sending request to an in-memory key-value store.

A. Result analysis

Sockperf. Our first evaluation aims to validate our allocator strategy with Sockperf. Like as shown in Section IV, we used the benchmark to measure both network latency and bandwidth. Figs 7 and 8 respectively show the latency degradation on InfiniBand and Ethernet networking; Figs 9 and 10 show bandwidth figures for both interfaces. The left side of each figure shows the performance degradation on the Rx path (i.e. the VM is receiving packets); conversely, the right side of each figure corresponds to the Tx path. We note that when using the InfiniBand interface, the performance degradation *% Deg* under our NUIOA-aware allocator is lower than that of other VM allocation schemes on both the Rx and Tx paths. For instance, performance degradations reach approximately 20% at 200000 and 400000 messages per second with the vNUMA configuration, compared to below 3% under our strategy. On the other hand, workloads using Ethernet adapters with kernel I/O (Fig 7) show more variability and less overall performance impact. While our allocation strategy shows little improvement

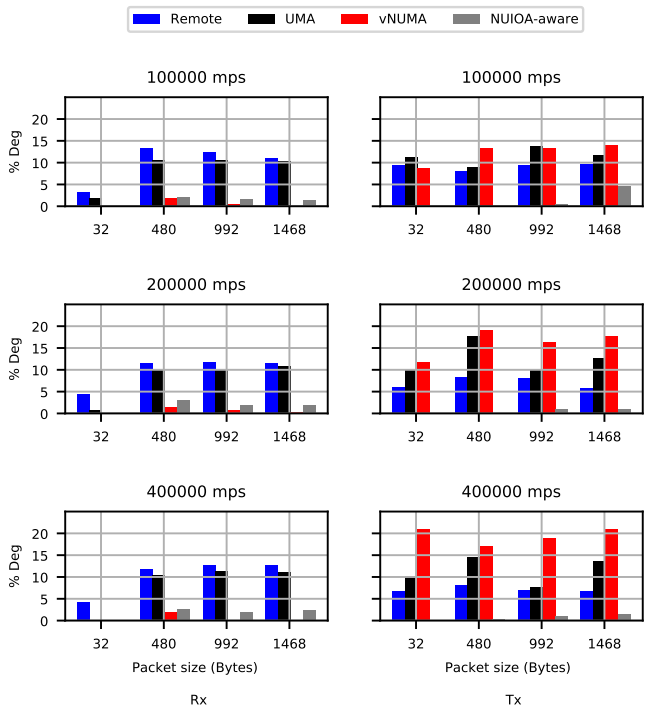

Fig. 7. Sockperf latency evaluation on InfiniBand networking.

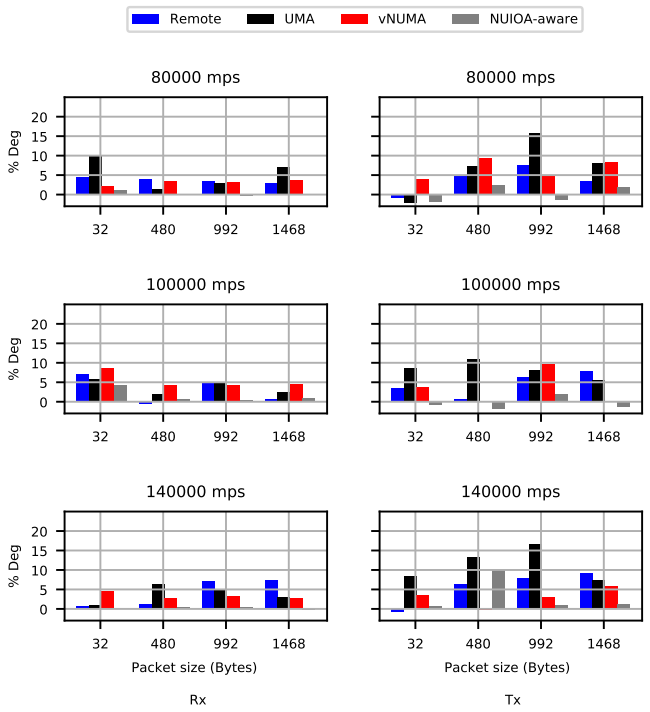

Fig. 8. Sockperf latency evaluation on Ethernet networking.

over other allocation schemes, it remains comparable to that of our best-case configuration, showing that our NUIOA allocator causes little to no performance degradation on this particular workload. Note that the *UMA* allocation scheme garners a performance impact on many configurations since the lack of NUMA-awareness causes remote memory accesses between the application and guest kernel; however, UMA allocation is a very common configuration for large VMs, and is in fact the default on Xen when vNUMA is not configured.

Figs 9 and 10 present the bandwidth results under our NUIOA allocator compared to other allocation schemes. Similarly to our latency results presented above, we note from Fig 9 that our allocation strategy shows nearly zero bandwidth degradation on InfiniBand networking, while on the Ethernet device (Fig 10) the impact of the NUIOA-aware strategy is lower, yet our scheme remains competitive with our reference.

Perftest RDMA. Perftest [28] is a software suite for benchmarking the performance of RDMA verbs over RDMA-capable network interfaces such as InfiniBand and RoCE. We evaluated the performance of all four main operations: *send* (destination node chooses data location), *RDMA read*, *RDMA write* and *RDMA atomic fetch+add* over two operation directions: transmit (Tx) where the VM-under-test *initiates* the RDMA operation, as well as receive (Rx) where the VM-under-test *responds to* the RDMA operation. Fig 11 shows the results obtained from Perftest. We observe that receiving RDMA writes causes a higher degradation than receiving RDMA reads, whereas the degradation is generally comparable when transmitting different types of RDMA operations.

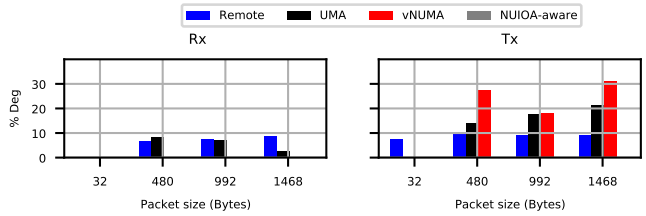

Fig. 9. Sockperf bandwidth results on InfiniBand networking.

Nevertheless, in all cases, our NUIOA allocator shows identical performance to the reference configuration, while other configurations show anywhere from 3-10% of degradation.

FIO. We assessed the impact of our allocation strategy with a file system performance benchmark named **fio** [30]. This benchmark is used to compute throughput for a realistic workload on an active disk. We deployed fio on our test VM configured with a infiniband device, and the benchmark performs IO requests to the remote machine with the data loaded into the ramdisk in order to not have the disk latency as a bottleneck. Therefore, all I/O requests use the network as it is the case with distributed file systems. The results obtained are presented in Fig 12. We can observe that our NUIOA-aware allocator shows almost identical performance to the reference configuration, while other configurations show anywhere from 3-7% of performance degradation.

Memcached benchmarks. Memcached is a popular in memory key-value store that can be used as a temporary object cache. Being an in-memory database, Memcached is

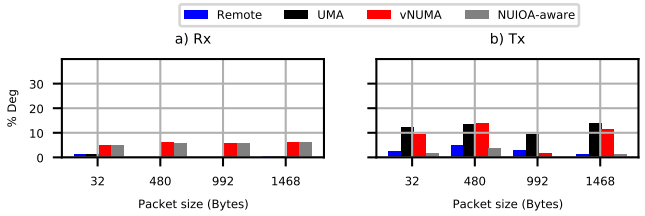

Fig. 10. Sockperf bandwidth results on Ethernet networking.

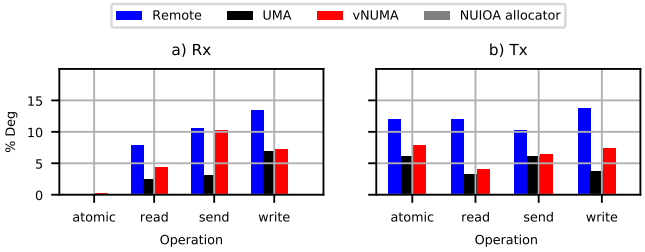

Fig. 11. Perftest evaluation results.

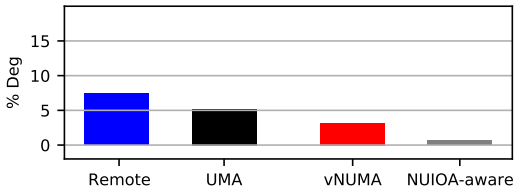

Fig. 12. Fio evaluation results.

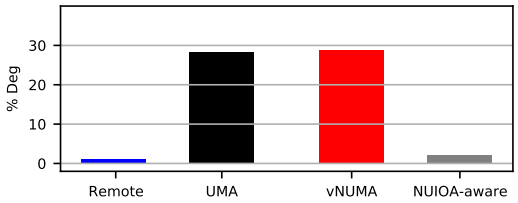

Fig. 13. Memcached evaluation results.

mostly CPU- and memory-intensive; we therefore used the program to demonstrate that our NUIOA-aware allocation strategy can be used for all classes of applications, including CPU- and memory-heavy ones. We configured Memcached to listen over the Infiniband interface for any requests, then used the Memaslap load generator client on another physical machine to stress the Memcached server. Fig 13 presents the performance degradation *% Deg* for each VM configuration compared to our reference configuration. We note that putting Memcached on the remote note presents nearly no performance penalty; however, running Memcached on a *UMA* or *vNUMA* VM causes a staggering 30% slowdown. This implies that the penalty caused by these configurations come from NUMA remote accesses themselves rather than any NUIOA effect, and that even vNUMA is by itself insufficient to stop this overhead. Regardless, our NUMA affinity-based strategy ensures similar performance as long as Memcached fits on one NUMA node, further stressing the importance of NUMA-aware workload scheduling.

B. Scalability evaluation

In this section, we demonstrate the ability of our allocation strategy to identify I/O-heavy workloads and schedule them on their home nodes when colocated with other applications in the same VM. Using our allocation strategy, we compare the obtained results in two scenarios: a) when running Sockperf alone, and b) when colocating the Sockperf VM with another VM hosting several instances of Sysbench's CPU benchmark. Table IV presents the results with each line representing an execution scenario: the first column contains the number of running instances of Sysbench; the second column presents the resulting Sockperf latency increase compared to when it is executed alone; and the last column shows the average QPS (Query Per Second) for all Sysbench instances. We observe that the performance degradation of Sockperf is close to zero in all scenario, regardless of how many Sysbench instances were running. This implies that regardless of the Sysbench VM's CPU activity, our workload detector correctly identified I/O-heavy workloads running inside the VM and applied the correct affinity settings. Moreover, the performance of Sysbench is not impacted, further demonstrating that our allocation strategy does not hurt other running workloads.

TABLE IV
PERFORMANCE IMPACT ON SOCKPERF WITH VARYING NUMBER OF SYSBENCH INSTANCES

No. instances Sysbench	Sockperf slowdown (%)	Sysbench QPS
1	0.59	685.12
2	0.55	684.68
4	0.93	683.55

C. Overhead of NUIOA-aware allocation

While our NUIOA-aware allocation strategy causes minimal application overhead as seen from our evaluations, it remains that our NUIOA allocator can introduce several issues involving both performance and administration overhead aspects:

- System operators must define a VM as an I/O-heavy VM. This can be done semi-automatically using the approaches described in Section V-A, or by the user simply by offering different classes of VMs (e.g. CPU-heavy, memory-heavy, I/O-heavy).
- Our soft-affinity pinning strategy only kicks in when there exists sufficient free resources on the workload's home node, deferring to the VM scheduler otherwise. Techniques such as scheduler-level soft affinity [31] can be used in the guest to prioritize scheduling I/O-extensive workloads on the home node.
- Our allocator requires enabling vNUMA on each I/O-heavy VM, which may introduce performance penalties to

both I/O and non-I/O applications. However, as discussed in Section V-B, our allocator can be combined with dynamic vNUMA solutions to reduce this penalty.

VII. Conclusion

In this paper, we studied in detail the various effects of NUIOA on I/O performance using multiple different workload and device types.

We thoroughly evaluated the impact of NUIOA on application performance in VMs and showed that current systems still experience performance impacts caused by NUIOA, with up to a 20% increase in latency and 10% decrease in bandwidth. We provide recommendations to resolve this issue.

We proposed a NUIOA-aware allocation strategy that distributes VMs over multiple physical NUMA nodes while informing VMs of NUIOA affinities, and showed that our scheme improves I/O performance similar to an optimal behavior, therefore preventing the 20% impact on various operations compared to the default VM allocation strategies.

Acknowledgments

This work is supported by the French *Agence nationale de la recherche* under the ANR WalkIn (ANR-20-CE25-0005) and ANR PicNic (ANR-20-CE25-0013) projects. Experiments presented in this paper were carried out using the Grid'5000 testbed, supported by a scientific interest group hosted by Inria and including CNRS, RENATER and several Universities as well as other organizations (see https://www.grid5000.fr).

References

[1] I. Advanced Micro Devices, "IOMMU Architectural Specification." https://kib.kiev.ua/x86docs/AMD/IOMMU/48882-2.00.pdf, 2021.

[2] M. Mahalingam, "I/O Architectures for Virtualization." http://download3.vmware.com/vmworld/2006/tac0080.pdf, 2021.

[3] S. V. Doren, "Abstract - hoti 2019: Compute express link," in *2019 IEEE Symposium on High-Performance Interconnects (HOTI)*. Los Alamitos, CA, USA: IEEE Computer Society, aug 2019, pp. 18–18. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/HOTI.2019.00017

[4] C. Lameter, "An overview of non-uniform memory access," *Communications of the ACM*, vol. 56, no. 9, pp. 59–54, 2013.

[5] B. Goglin and S. Moreaud, "Dodging non-uniform i/o access in hierarchical collective operations for multicore clusters," in *Proceedings of the 2011 IEEE International Symposium on Parallel and Distributed Processing Workshops and PhD Forum*, ser. IPDPSW '11. USA: IEEE Computer Society, 2011, p. 788–794. [Online]. Available: https://doi.org/10.1109/IPDPS.2011.222

[6] F. Gaud, B. Lepers, J. Funston, M. Dashti, A. Fedorova, V. Quéma, R. Lachaize, and M. Roth, "Challenges of memory management on modern numa systems," *Commun. ACM*, vol. 58, no. 12, p. 59–66, Nov. 2015. [Online]. Available: https://doi.org/10.1145/2814328

[7] Libnuma, "A NUMA API for LINUX*." http://developer.amd.com/wordpress/media/2012/10/LibNUMA-WP-fv1.pdf, 2013.

[8] T. Rosado and J. Bernardino, "An overview of openstack architecture," in *Proceedings of the 18th International Database Engineering & Applications Symposium*, ser. IDEAS '14. New York, NY, USA: Association for Computing Machinery, 2014, p. 366–367. [Online]. Available: https://doi.org/10.1145/2628194.2628195

[9] OpenStack, " I/O (PCIe) based NUMA scheduling," https://specs.openstack.org/openstack/nova-specs/specs/kilo/implemented/input-output-based-numa-scheduling.html, 2021.

[10] G. Voron, G. Thomas, V. Quéma, and P. Sens, "An interface to implement numa policies in the xen hypervisor," in *Proceedings of the Twelfth European Conference on Computer Systems*, ser. EuroSys '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 453–467. [Online]. Available: https://doi.org/10.1145/3064176.3064196

[11] D. Mvondo, B. Teabe, A. Tchana, D. Hagimont, and N. De Palma, "Closer: A new design principle for the privileged virtual machine os," in *2019 IEEE 27th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, 2019, pp. 49–60.

[12] B. Teabe, A. Tchana, and D. Hagimont, "Billing system cpu time on individual vm," in *2016 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, 2016, pp. 493–496.

[13] B. Bui, D. Mvondo, B. Teabe, K. Jiokeng, L. Wapet, A. Tchana, G. Thomas, D. Hagimont, G. Muller, and N. DePalma, "When extended para - virtualization (xpv) meets numa," in *Proceedings of the Fourteenth EuroSys Conference 2019*, ser. EuroSys '19. New York, NY, USA: Association for Computing Machinery, 2019. [Online]. Available: https://doi.org/10.1145/3302424.3303960

[14] D. Berrangé, "Openstack performance optimization: Numa, largepages & cpu pinning," https://www.linux-kvm.org/images/0/0b/03x03-Openstackpdf.pdf, 2014.

[15] Y. Ren, T. Li, D. Yu, S. Jin, and T. Robertazzi, "Design, implementation, and evaluation of a numa-aware cache for iscsi storage servers," *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 2, pp. 413–422, 2015.

[16] W. Rödiger, T. Mühlbauer, A. Kemper, and T. Neumann, "High-speed query processing over high-speed networks," *Proc. VLDB Endow.*, vol. 9, no. 4, p. 228–239, Dec. 2015. [Online]. Available: https://doi.org/10.14778/2856318.2856319

[17] V. T. Publications, "Tuning vCloud NFVfor data plane intensive workloads, Open Stack Edition." https://docs.vmware.com/en/VMware-vCloud-NFV-OpenStack-Edition/3.0/vmwa-vcloud-nfv30-performance-tunning.pdf, 2019.

[18] A. Banerjee, R. Mehta, and Z. Shen, "Numa aware i/o in virtualized systems," in *Proceedings of the 2015 IEEE 23rd Annual Symposium on High-Performance Interconnects*, ser. HOTI '15. USA: IEEE Computer Society, 2015, p. 10–17. [Online]. Available: https://doi.org/10.1109/HOTI.2015.17

[19] L. Shelton, "High performance I/O with NUMA systems in Linux." 2013.

[20] Red Hat, Inc., "libvirt NUMA Tuning," https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html/virtualization_tuning_and_optimization_guide/sect-virtualization_tuning_optimization_guide-numa-numa_and_libvirt#sect-Virtualization_Tuning_Optimization_Guide-NUMA-VCPU_Pinning.

[21] J. Squyres, "Process and memory affinity: why do you care? High Performance Computing Networking." http://blogs.cisco.com/performance/process-and-memory-affinity-why-do-you-care, 2013.

[22] S. Moreaud, B. Goglin, and R. Namyst, "Adaptive MPI multirail tuning for non-uniform input/output access," in *European MPI Users' Group Meeting*. Springer, 2010, pp. 239–248.

[23] I. Smolyar, A. Markuze, B. Pismenny, H. Eran, G. Zellweger, A. Bolen, L. Liss, A. Morrison, and D. Tsafrir, "Ioctopus: Outsmarting nonuniform dma," in *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 101–115. [Online]. Available: https://doi.org/10.1145/3373376.3378509

[24] NVIDIA, "NVIDIA Mellanox Socket Direct Adapters," https://www.nvidia.com/en-us/networking/ethernet/socket-direct/, 2021.

[25] M. N. B. marking Utility, "Mellanox Technologies," https://github.com/Mellanox/sockperf, 2021.

[26] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," *SIGOPS Oper. Syst. Rev.*, vol. 37, no. 5, p. 164–177, Oct. 2003. [Online]. Available: https://doi.org/10.1145/1165389.945462

[27] Libvma, "Libvma." https://github.com/Mellanox/libvma, 2021.

[28] Perftest, "Perftest Package." https://community.mellanox.com/s/article/perftest-package, 2013.

[29] A. Soliman, *Getting Started with Memcached*. Packt Publishing, 2013.

[30] fio, "File system performance benchmarking." https://docs.gitlab.com/ee/administration/operations/filesystem_benchmarking.html, 2021.

[31] Oracle, "Soft Affinity - When Hard Partitioning Is Too Much," https://blogs.oracle.com/linux/post/soft-affinity-when-hard-partitioning-is-too-much, 2019.