VPRI: Efficient I/O Page Fault Handling via Software-Hardware Co-Design for IaaS Clouds

Kaijie Guo[1*], Dingji Li[2,3*], Ben Luo[1], Yibin Shen[1], Kaihuan Peng[1], Ning Luo[1], Shengdong Dai[1], Chen Liang[1], Jianming Song[1], Hang Yang[1], Xiantao Zhang[1], Zeyu Mi[2,3]

[1]*Alibaba Group*

[2]*Institute of Parallel and Distributed Systems, Shanghai Jiao Tong University*

[3]*Engineering Research Center for Domain-specific Operating Systems, Ministry of Education, China*

Abstract

Device pass-through has been widely adopted by cloud service providers to achieve near bare-metal I/O performance in virtual machines (VMs). However, this approach requires static pinning of VM memory, making on-demand paging unavailable. The hardware device I/O page fault (IOPF) capability offers an optimal solution to this limitation. Current IOPF approaches, using either standard IOMMU capabilities (ATS+PRI) or devices with independent IOMMU implementations, have not gained widespread adoption in public Infrastructure-as-a-Service clouds. This is due to high costs, platform dependency, and significant impacts on performance and service level objectives (SLOs). We present the Virtualized Page Request Interface (VPRI), a novel IOPF system developed through software-hardware collaboration. VPRI is not only platform-independent, free from address translation complexities, but also cost-effective, and designed to minimize SLO impact. Our work enables large-scale deployment of IOPF capability in Alibaba Cloud with negligible impact on SLOs. When integrated with memory management software, it significantly enhances memory utilization in public IaaS clouds, effectively overcoming the static memory pinning restriction associated with pass-through devices.

CCS Concepts: • **Information systems** → *Storage virtualization*; • **Networks** → *Cloud computing*; • **Software and its engineering** → **Memory management**.

Keywords: I/O Page Fault, Virtualization, Cloud Computing, Service Level Objective

SOSP '24, November 4–6, 2024, Austin, TX, USA

ACM ISBN 979-8-4007-1251-7/24/11…$15.00

https://doi.org/10.1145/3694715.3695957

1 Introduction

Device pass-through is universally deployed in modern Infrastructure-as-a-Service (IaaS) cloud data centers to deliver peak I/O performance for high-density virtual machines (VMs). To enforce inter-VM isolation, hypervisors must set up appropriate I/O page tables (IOPTs) for pass-through devices, restricting their direct memory access (DMA). However, unlike CPUs, most devices lack the capability to handle page faults. Consequently, current hypervisors statically pin and map all VM pages to prevent I/O page faults (IOPFs), leading to poor DRAM utilization. To quantify this issue, we conducted a comprehensive study in our production environment (Alibaba Cloud) as one of the largest cloud service providers (CSPs). Our data centers have demonstrated that the absence of IOPF support in existing I/O pass-through can result in up to 50% wasted DRAM. The study encompassed 1,000 servers, each equipped with 200GB to 1TB of physical memory and hosting over 20,000 VMs. We performed statistics based on 2MB granularity extended page table (EPT) scans over a week. The results revealed that the average percentages of cold pages not accessed within intervals of 2, 5, and 7 minutes were 45%, 42%, and 39%, respectively. Notably, 30% of the servers had an average daily cold page rate exceeding 50%. Worse, the static memory pinning constraint hinders the application of advanced memory management techniques such as memory overcommitment [50], lazy memory allocation [10], copy-on-write [51], and page migration [34].

The Address Translation Service (ATS) in conjunction with the Page Request Interface (PRI), as defined by the PCI-SIG standard [35], is currently the only publicly available method for enabling IOPF[1]. To detect an IOPF, a device requires an address translation (AT) result obtained through an IOPT walk performed by the I/O memory management unit (IOMMU). To deliver an IOPF to its handler, the device issues a page request to the IOMMU's PRI, triggering an interrupt that prompts the host to process the request. Although supporting IOPF can effectively mitigate the above DRAM cost

[*]Co-first authors.

[1]This paper focuses on IOPFs when accessing host memory. Supporting page faults for vendor-specific on-device memory [33] is beyond our scope.

problem, today's IaaS data centers rarely enable IOPF due to two major issues:

Compatibility issue: ATS+PRI necessitates AT capabilities on both CPU and device sides, which in turn require specific IOMMU features and on-device support. Currently, these IOMMU features are unsupported by over 90% of the CPUs within our global server fleet. Moreover, they typically require device-side translation look-aside buffers (DevTLBs) for AT performance, posing significant challenges in terms of both design complexity and hardware costs [26, 47]. Consequently, CSPs face a financial dilemma: either upgrade all existing CPUs and devices to support ATS+PRI, or continue tolerating low DRAM utilization without enabling IOPF.

Performance issue: To ensure the quality of cloud services, CSPs have established various service level objectives (SLOs) for I/O performance. These include thresholds such as maximum tail latency and minimum instantaneous throughput, setting baseline performance benchmarks over specified periods. Violations of these I/O SLOs can detrimentally affect the reputation and revenue of CSPs. Unfortunately, on hardware platforms equipped with ATS+PRI, the impact of IOPFs on I/O SLOs is significant. We have observed a substantial number of I/O SLO violations in our data centers due to uncontrolled IOPF rates and latency. Our experiments demonstrate that a VM enabled with IOPF can experience up to a 70% instantaneous reduction in I/O throughput when the IOPF rate surges in bursts to 10 per second.

Unlike CPU page faults, IOPFs are asynchronously delivered to their host handlers via interrupts. Our analysis reveals that this delivery latency can be up to three orders of magnitude longer than that of a CPU page fault. It can further lead to blocking within I/O queues and even result in network packet drops for milliseconds, potentially causing costly retransmissions and flow control issues that significantly degrade performance. The latency includes the time taken by the top-half interrupt service routine (ISR) and the subsequent scheduling of the bottom-half kernel thread for IOPF handling, which becomes the primary performance bottleneck. Consequently, merely enabling ATS+PRI is insufficient to fulfill SLO requirements due to the inherent IOPF processing latency.

To address static page pinning, prior efforts have attempted to decouple IOPF from IOMMU and minimize the performance impact of IOPF latency through either software or hardware approaches. However, implementing these solutions for CSPs operating under strict cost and SLO constraints presents significant challenges. The software approach utilizes the hypervisor as a software IOPT walker to respond to AT queries from devices [40]. While this method can handle IOPF without relying on IOMMU features, it introduces an unacceptable AT latency of approximately 10 µs for every DMA operation. Alternatively, the hardware approach offloads the IOPT to the device, similar to methods used in GPUs [2, 17, 38]. However, this approach requires a complex on-device MMU in addition to the DevTLB, which significantly increases both design complexity and hardware costs [26, 47]. Moreover, both approaches are hampered by significant latency during IOPF delivery, similar to that observed in hardware IOPF systems.

In this paper, we propose Virtualized Page Request Interface (VPRI), a platform-independent IOPF solution that offers both low cost and high performance for CSPs. We identify that the tight coupling between address translation and IOPF detection is the fundamental obstacle to compatibility. Our key observation is that determining page presence requires only two attribute bits from an IOPT entry, rather than the entire translation. VPRI enables the hypervisor to maintain an on-device bitmap that simply caches attribute bits from IOPTs. By querying this bitmap, a device can independently detect IOPFs, enabling both low latency and low cost.

To address the performance issue, given the inherent high latency of IOPF delivery, VPRI aims to minimize its impact by reducing IOPF frequency through adaptive memory pinning. Specifically, we propose a fine-grained, hardware-assisted I/O page access logging mechanism, facilitating the development of pinning strategies that leverage DMA locality. While a simple strategy might involve using a basic least-recently used (LRU) policy to pin hot DMA pages, this approach could lead to excessive memory pinning, hindering overall memory utilization. Instead, we create a heuristic-based dual-LRU policy that reduces the amount of memory pinned by up to 81% and significantly diminishes IOPF frequency.

We have successfully implemented VPRI within Alibaba Cloud's data processing units (DPUs) *without incurring any additional hardware costs*. This innovation facilitates hardware IOPF support across *all existing and future platforms* and reduces the frequency of IOPF by up to 99% by pinning only 5.2% of memory, thereby *meeting I/O SLO*. None of these benefits can be provided by the state-of-the-art ATS+PRI systems. As a new building block in our cloud infrastructure, VPRI enabled memory overcommitment that saved up to 15% in DRAM costs across several production clusters within the server fleets.

In summary, in this paper we:

- share our lessons in evaluating standard IOPF in a hyperscale IaaS, and design considerations to walk away from the standard solution (§2 and §3).
- analyze the core components of a more efficient IOPF system including simplified hardware, IOPF frequency reduction through fine-grained I/O footprint tracing (§4).
- disclose VPRI based on a hardware-software co-design that enables IOPF for all existing platforms at a low cost while ensuring high performance (§5 and §6).
- evaluate and demonstrate that VPRI can effectively diminish SLO impact on real-world workloads, maintaining high memory utilization (§7).

To the best of our knowledge, this work is the first to enable hardware IOPFs in public IaaS hosting for generic network and storage workloads on a massive production scale at one of the world's largest CSPs. It demonstrates compelling advantages over the existing standard solution in terms of complexity, compatibility, and performance.

2 Background

2.1 IOPF Workflow in the PCIe Standard

The PCIe 4.0 standards provide ATS and PRI features to support IOPF. ATS is utilized to detect IOPFs during DMAs. When an IOPF occurs, PRI is used to generate an asynchronous page request to the CPU. Using the Intel IOMMU as an example, Fig 1 depicts the standard two-stage process of the IOPF workflow.

Pre-DMA lookup stage: ① Before a DMA operation, the I/O engine performs a DevTLB lookup; a miss occurs if the hypervisor has unmapped the relevant page or if the translation is not cached in the DevTLB. ② The device sends an ATS request to the IOMMU over PCIe. ③ The IOMMU performs IOPT walk and finds the page is not mapped, which takes microseconds. ④ The IOMMU responds to the device with the ATS result. Note that ② and ④ together cost ~1 µs in PCIe transmissions.

Page requesting stage can be divided into three sub-stages:
1. Pre-fault sub-stage: ⑤ The device sends a page request message to the IOMMU over PCIe. ⑥ The IOMMU writes a descriptor to the page request queue, and then triggers a host interrupt to execute the IOPF ISR. ⑦ A bottom-half thread (either a threaded IRQ handler or a user-defined workqueue) is scheduled asynchronously to handle the fault. Note that the software fault handler could incur blocking operations and cannot be executed in an ISR.
2. Fault handling sub-stage: The memory subsystem manages the page fault, allocating a physical page, populating its contents, and configuring the page tables, akin to a CPU page fault.
3. Post-fault sub-stage: ⑧ and ⑨ involve sending a page response to the device through the IOMMU's invalidation queue. Afterwards, the DMA operation can resume.

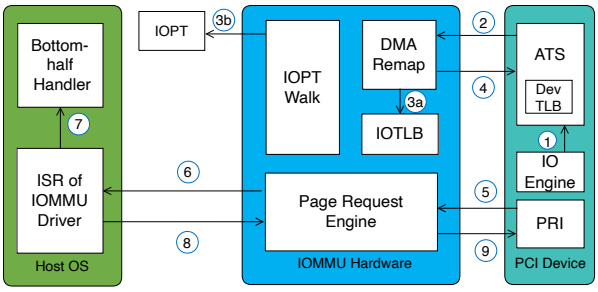


Figure 1. Standard IOPF flow.

In summary, the IOPF detection and delivery workflow requires both the platform and the device to have hardware support for ATS and PRI features. Additionally, IOPF processing experiences increased latency due to its interrupt-based model and the long distances involved.

2.2 CSP Efforts in I/O SLOs

At our data centers, Quality of Service (QoS) measures related to consistency and stability take precedence over average performance metrics. For real-time online services such as gaming, financial transactions, and video live streaming, even a brief and intermittent interruption of just 100 milliseconds can adversely affect the business of customers, ultimately impacting the reputation and revenue of CSPs.

Therefore, for I/O performance, CSPs have defined various I/O SLO metrics [12, 43, 53], such as tolerable tail latency thresholds (e.g., P99.9th), maximum tail latency, minimum instantaneous throughput or I/O operations per second (IOPS), and packet loss rates. In the production environment, each server deployed in the fleet is monitored for tail I/O occurrences across different time windows (minute, hour, day, etc.). Violations of I/O SLOs trigger alerts, with stricter SLOs associated with higher-end products.

To meet I/O SLOs, we have designed and deployed our own DPUs [55] that maxmize I/O performance by offloading I/O virtualization from CPU-side software. A DPU is a PCIe device capable of hardware emulation for potentially thousands of virtual functions (VFs) (such as virtio, NVMe[37], RDMA [39], etc.). These VFs can be passed through to VMs, delivering extremely high packets per second (PPS) and IOPS with minimal latency, while ensuring safety and performance isolation. Since being deployed, our DPUs have served millions of customers daily and have become a crucial component in the infrastructure of the hyperscale server fleet.

3 Lessons Learned from Supporting IOPF

Under the premise of ensuring QoS, CSPs aim to minimize costs as much as possible. To control DRAM costs and improve memory utilization, we need to enable IOPF on our DPUs. This will allow us to overcome the limitations of traditional device pass-through methods, which require memory pinning and thus prevent memory overcommitment.

Initially, we explored supporting IOPF through ATS+PRI in PCIe 4.0 standard. However, due to compatibility and performance issues, we moved away from this conventional route to develop a novel IOPF framework tailored for IaaS CSPs. This section presents the challenges CSPs face in making hardware compatible with ATS+PRI and the significant performance impact of IOPF with ATS+PRI enabled.

3.1 Difficulties in Hardware Compatibility

Scarce platform support: Only a small fraction of commodity platforms in data centers have IOMMUs that support

ATS+PRI. We discovered that **fewer than 10%** of servers in our fleet support the standard IOPF interface, indicating a transition to full hardware compatibility will take 5 to 10 years.

High device costs: Our experience indicates that, in the long term, even after all platforms are upgraded to support ATS+PRI in the IOMMU, enabling ATS on the device side in the public cloud still incurs significant costs due to hardware and design complexity. We estimate that achieving an optimal ATS implementation would increase DPU hardware costs by at least 20% due to upgrades in hardware circuits. This would also result in a 15% increase in energy consumption and CO2 emissions, exceeding budget constraints for DPU deployment. Additionally, the complexity of the design means that development could take years and face challenges related to compatibility and debugging across different platforms.

The increased costs and complexity primarily stem from the demand of DevTLB, which caches AT results to improve the performance of pre-DMA lookups. Specifically, ATS-enabled devices lacking DevTLB require each pre-DMA lookup to initiate an ATS request to the IOMMU (② and ④ in Fig 1). This process incurs a penalty of a round-trip PCIe message at the microsecond level [25, 40] for every DMA. As a result, DevTLB is necessary to mitigate this issue.

Nevertheless, DevTLB requires expensive SRAM and power-intensive set-associative lookup, making it difficult to scale up. This challenge is especially pronounced in public cloud environments, where large DRAM capacities result in poor TLB reach and higher TLB miss rates [13]. Worse, as tenant density [1, 19] and CPU cores increase, DPUs now support thousands of VFs, each contending for TLB resources. This heightens the risk of denial-of-service attacks and necessitates complex designs [18, 24, 26, 47] for VF-level isolation, dynamic TLB provisioning, and multi-level TLBs [6, 27, 54], significantly increasing device hardware costs.

3.2 Performance Issues of IOPF

Even with ATS+PRI supported, existing IOPF handling procedure can lead to various performance issues.

High IOPF handling latency: The pre-fault stage of IOPF incurs significant latency, ranging from tens to hundreds of microseconds, which is up to three orders of magnitude greater than that of a CPU page fault. This latency is primarily due to PCIe delays and the asynchronous scheduling latencies of both the ISR and the bottom half fault handler, as shown in Fig 2.

Firstly, ISR can be blocked by critical sections (e.g., by *spin_lock_irqsave*) or other ISRs, possibly extending the delay to hundreds of microseconds [20, 42] if the CPU is under heavy load. Secondly, the scheduling of the bottom-half (such as workqueues) can further increase the latency by tens to even hundreds of microseconds. This becomes evident when

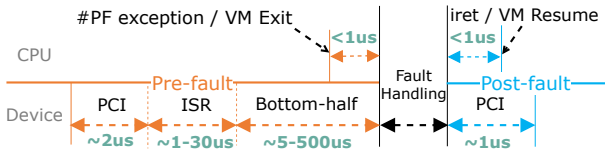


Figure 2. Breakdown of fault handling for CPU and device.

the host or hypervisor has intensive non-preemptive kernel workloads, such as page table scanning, page migration, and swap-out.

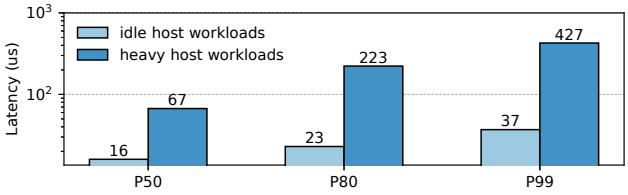


Figure 3. Round trip time of IOPF.

To demonstrate IOPF latency in real world, we injected 1,000 sequential IOPFs and measured the round-trip latency of each IOPF on the device side, under varying host workload intensities. On the host side, the IOPF ISR schedules a bottom-half worker. This worker transmits the page fault response over PCIe without performing additional operations, such as modifying the page table. Consequently, the round-trip latency closely matches the cumulative latency of both the pre-fault and post-fault operations.

Figure 3 illustrates that under minimal host kernel work intensity, the round-trip latency ranged from 10 to 40 µs consistently. This latency level, which represents the baseline latency of the interrupt-based IOPF model, is substantially longer than that of CPU faults, irrespective of the vendor or hardware implementation. Furthermore, when the host was running high-intensity kernel tasks in the background, such as compression, encryption, and page table scanning, the 99th percentile latency of IOPF spiked to 427 µs.

This latency is non-negligible, particularly when the latency of the actual fault handler is exceptionally low. This is evident in scenarios involving minor page faults or process swapping that employ high-speed technologies such as RDMA [14] or CXL [31], which exhibit latencies in the order of microseconds. For instance, even a few minor faults can cause a substantial performance slowdown of 2× or more on Intel IAA accelerators [19].

Queue being blocked during IOPF handling: When a device encounters an IOPF in a specific I/O queue, it must pause the processing of that queue until the IOPF is resolved to ensure the correct ordering of the queue. This can cause the queue to become blocked and lead to service interruptions for a duration determined by combined fault latency of

hardware and software. The software latency of a major fault in our data center varies from 20 to 6,000 µs, depending on the size of the mapping and which tier of swap media backs the faulted page. Furthermore, it is common for multiple CPU threads to share a single device or queue, leading to a one-to-many blocking scenario.

Fig 4 examines the throughput drop of realistic workloads in one minute under varying IOPF rates and latencies that might occur at CSPs. We observed that the impact of queue blocking on storage workloads is fundamentally different from that on network workloads.

For block devices, queue blocking causes a linear slowdown in average throughput that is proportional to the fault rate and latency. It also causes the unprocessed I/O requests in the queue to become long-tail requests. For NIC devices, queue blocking prevents the device from consuming the receive (RX) buffers posted by the driver for incoming packets. If the device stores these packets in its limited RX cache, which is shared among all queues, it can lead to RX cache overflow, impacting all other RX queues. A straightforward yet detrimental approach involves dropping packets that arrive at the faulted queue. This method relies on the sender's protocol, such as TCP, to retransmit the dropped packets after a timeout period. Previous solutions, such as backup rings [22], require modifications to the NIC driver, making them unsuitable for public IaaS environments.

Therefore, TCP-based workloads in IaaS clouds are particularly sensitive to IOPF, as frequent packet drops activate flow control on the sender's side [22], leading to a reduced transfer rate or even throttled for hundreds of milliseconds. We observed that the reduction in throughput or IOPS is influenced far more by the fault rate than by fault latency. For example, both multi-threaded Redis and Memcached experienced around a 15% drop in throughput when the IOPF rate reached two per second. In contrast, when these workloads encounter CPU faults in one of the working threads under the same fault conditions, there is no throughput degradation because the remaining threads collectively saturate the bandwidth of the VF. Additionally, we observed that regardless of the software latency, the end-to-end tail latency consistently hovered around the magic number of 200 ms. We speculate that this latency corresponds to the CentOS TCP stack's default retransmission timeout. The throughput degradation in iperf-UDP follows a similar pattern to that observed with block devices, as the UDP protocol is not sensitive to packet loss.

Resolving the queue blocking issue is challenging because it requires modifications to both the device interface and its underlying semantics to enable comprehensive out-of-order processing.

Scalability and isolation challenges: First, scalability is an issue as the IOPF handler usually uses a single interrupt vector. Software latency, defined as the time from when

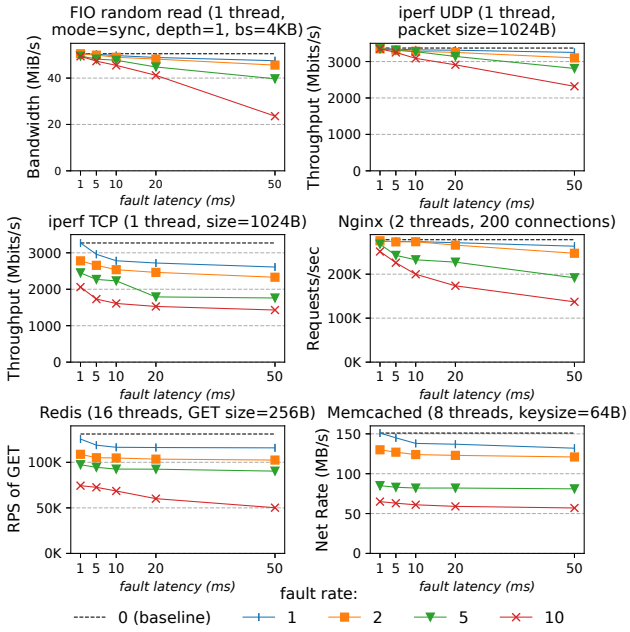


Figure 4. Throughput impact under varying IOPF rate and latency. Hardware configuration is shown in Table 1.

the IOPF's ISR executes to its complete processing by the memory subsystem, can significantly increase when multiple devices generate IOPFs simultaneously if processed by a single worker. Our evaluation shows that with one IOPF thread, software latency can rise up to 10.4× as the number of VFs increases to 16.

Second, cross-tenant isolation is critical. Enterprise tenants are allocated dedicated physical CPUs for consistent performance. A malicious tenant could launch performance or denial-of-service attacks if their IOPF worker operates on a CPU assigned to another tenant.

3.3 Impact of IOPF on I/O SLOs

In the production environment, intermittent I/O jitter in the infrastructure (e.g., network congestion) can cause long-tail issues and packet loss, which impacts I/O SLO metrics. CSPs spend significant resources to keep I/O jitter within a tolerable level. However, with the performance issues of the current IOPF architecture, IOPFs can worsen I/O jitter, especially during rapid successive bursts, which can directly lead to I/O SLO violations.

Fig 5 shows that queue blocking combined with packet dropping can result in a network interruption of approximately 300 ms by only a single IOPF in the NIC's RX queue, even if the fault latency is 5 ms in software. This occurs because the sender's network stack activates flow control mechanisms that pause transmission. Even a few page faults on the NIC within a minute can result in a violation of SLO

metrics concerning minimum throughput and packet drop rates.

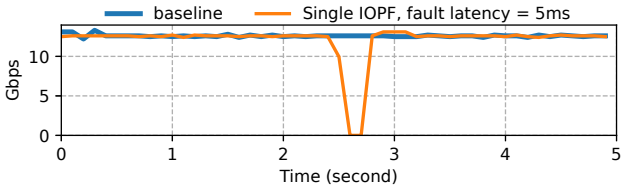


Figure 5. Interruption of throughput in iperf3 caused by a single IOPF with a fixed fault latency of 5 ms.

Fig 6 shows the significant changes in 99.9th and 99.99th pecentile latency of a block device following the injection of IOPF with a fixed latency of 5ms at various IOPF rates. We injected burst IOPFs at rates of 1 and 10 per second to simulate moderate and intensive burst IOPF scenarios, respectively. Observations:

- P99.9th latency: At the IOPF rate of 1/s, the latency increase is only noticeable when the block size is 2MB. This is because larger block requests have lower baseline IOPS. At the IOPF rate of 10/s, an increase in the latency is evident across all block sizes by 2-11×.
- P99.99th latency: The impact is evident across all block sizes at both IOPF rates.

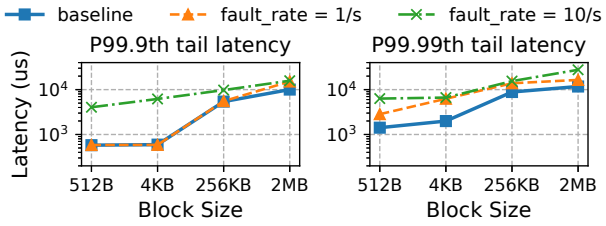


Figure 6. FIO tail latency affected by burst IOPFs over a duration of 1 minute. Hardware configuration is as Table 1.

3.4 IOPF Rate in Production

In our early production environment years ago, we had hundreds of servers (each with 200GB to 1TB of physical DRAM) running with a 10%-15% memory overcommitment for a few weeks with hardware IOPF. The memory reclaim ratio of each VM varies from 0% to 50%, balanced by the pressure stall information (PSI) [50] algorithm in Linux across VMs on the same server. The number of IOPF for a VM depends on several factors: 1) the VM's memory reclaim ratio and 2) the VM's I/O load and working set.

We selected 800 random VMs and illustrated the IOPF counts for both NIC and block devices using a cumulative distribution function (CDF), as depicted in Fig 7. Some key observations:

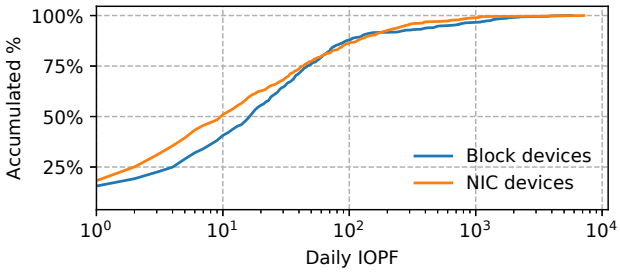


Figure 7. CDF of daily IOPF counts per device from 800 randomly selected VMs.

- The daily average and 99th percentile IOPF counts are 38 and 576 for NIC devices, and 66 and 983 for block devices, respectively.
- Within one day, over 10% of VMs encounter at least one burst in IOPF (more than 5 IOPF within 1 second). The peak instantaneous IOPF rate varies between 23 and 40 per second.

This reflects the baseline IOPF rate in production following the implementation of hardware IOPF and memory overcommitment. Under such an IOPF rate, we received warnings of SLO violation from both NIC and block devices in production, primarily originating from VMs with higher IOPF (e.g., P99th). The NIC devices were more severely affected by instantaneous throughput drops due to single IOPFs occurring with higher fault latency resulting from swaps to SSDs. Issues with block devices primarily arose from long-tail I/O counts exceeding minute-level P99.9th/P99.99th thresholds caused by bursts of I/O workload and IOPFs. To address I/O SLO violations, we had to lower the overcommitment ratio to below 5%.

3.5 Summary

Our lessons demonstrated that: 1) The tight coupling of AT capabilities and pre-DMA lookup is the root cause of the incompatibility on existing CPU platforms and the high DevTLB costs on the device side. 2) The frequency of IOPFs directly determines the severity of SLO violations, given the high latency inherent in the entrenched interrupt-based IOPF handling model. We concluded that, over the long term, the current standard IOPF solution falls significantly short of meeting our requirements in terms of compatibility, cost, and performance. To this end, we explored innovative approaches to redesign an optimized hardware IOPF solution.

4 Exploring More Efficient IOPF

We established two primary objectives for remodeling an optimized IOPF. First, we aim to create a platform-independent IOPF system that can be implemented across all existing platforms at minimal device cost. Second, we seek to minimize the impact of IOPF on I/O SLOs.

4.1 Decouple AT from Pre-DMA Lookup

The primary challenge in decoupling address translation from pre-DMA lookups lies in detecting IOPFs without obtaining AT results through IOPT walks. A fact that is often overlooked is that pre-DMA lookup requires only the page attributes in the PTE. Page attributes occupy a small fraction of the IOPT, merely 2 bits per leaf entry, which are present bit (bit 0 on x86) and R/W permission bit (bit 1 on x86). If we can extract only the page attributes from the IOPT and offload them to the device, then it is possible we can employ a simple and cost-effective mechanism to replace ATS for pre-DMA lookup.

An I/O address space (IOAS) has arbitrary mapping sizes (1GB/2MB/4KB). Regardless of the actual mapping size, we can divide the IOAS into 4KB blocks, which is the smallest page size on x86. We then compile a static table for this IOAS where each page attribute entry (PAE) stores the attribute bits of the corresponding 4KB block, indexed by page number (VA[12:63]). Each PAE consists of only 2 bits: a presence bit and a read/write permission bit, encompassing three valid states: *read-write (RW)*, *read-only (RO)*, and *unmapped*. We refer to this data structure as Page Attribute Bitmap (PA-BITMAP).

As illustrated in Fig 8, a 4KB mapped page has one unique PAE in the PA-BITMAP. A 2MB mapped page has 512 consecutive PAEs associated with it, all of the same value. 1GB mappings are similar. Assuming the PA-BITMAP within the device is perfectly synchronized with the IOPT, the device can efficiently conduct pre-DMA lookups for I/O virtual addresses (IOVAs) at 4KB granularity using a very simple semantic:

Read the PAE from the corresponding PA-BITMAP using the page number of the IOVA as the index. If the PAE is *unmapped* then a page fault needs to be triggered. Similarly if the PAE is *RO* and the incoming DMA is a write operation, a page fault also needs to be triggered.

The total bit capacity of a VM's PA-BITMAP is:

$$bit\_capacity = \frac{MAX\_IOVA}{4K} \times sizeof(PAE) \qquad (1)$$

1GB of IOVA space requires only 64KB of DRAM for storing the PA-BITMAP. As an example, a system with a total guest physical address space of 1TB for all VMs only requires 64MB for PA-BITMAPs, which is a small fraction of the total DRAM.

Due to the straightforward linear data structure of PA-BITMAP, performing a pre-DMA lookup at a 4KB granularity requires just a single memory access. Additionally, it allows the pre-DMA lookup to be 100% performed locally within the device, without the need to cross the PCIe boundary. Although the pre-DMA lookup latency is slightly longer than that of a fully/set-associative DevTLB during a hit, it offers a good trade-off in terms of hardware complexity, cost, and compatibility. Furthermore, the DRAM access latency can be effectively reduced by utilizing the DPU's processor cache[7]. Our evaluation shows that the impact of pre-DMA lookup based on PA-BITMAP on base performance is negligible, because: 1) I/O operations typically have long round-trip times, ranging from tens to thousands of microseconds and 2) I/O devices leverage pipelining and parallelism in the backend engine to maximize throughput while minimizing the effects of minor latency variations.

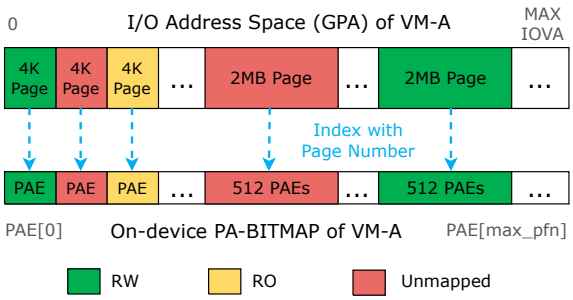


Figure 8. Concept of PA-BITMAP.

The second challenge is to remove the dependency on PRI for reporting faults. Our solution can be straightforward yet effective: the DPU provides a dedicated PCI function that supports delivering IOPFs from all its VFs. This approach eliminates the need for any modifications to the existing VFs or their device drivers. A driver in the host interfaces with this PCI function, facilitating page request interactions via MSI interrupts, thus eliminating the dependency on IOMMU PRI capability.

4.2 Reducing IOPF Rate for Generic Workloads

Frequent IOPFs pose significant performance impacts on generic workloads, especially networking, and optimizing fault latency is less effective (§3.2). The IOMMU typically uses batching to amortize the overhead among multiple PRI requests but only has limited effect [24].

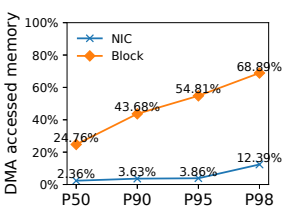
We propose that a more efficient approach to mitigate the impact of IOPFs is to reduce their frequency rather than their latency. This requires system software to track the DMA footprint and extract characteristics through software to find out memory regions or pages with a high tendency for IOPFs. Subsequently, dynamic pin protection can be applied to these pages to reduce the probability of IOPFs occurring. We will describe the analysis process, insights, and influence on the final design [3].

4.2.1 IO footprint and temporal locality. To optimize page pinning and minimize page faults, a promising strategy involves understanding the characteristics of the I/O footprint and identifying pages that are more likely to experience

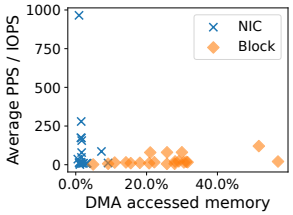
[3]In this paper, we focus on generic network and storage workloads. Special workloads such as RDMA and graphics are out of scope.

page faults. While from the CPU's perspective, this task is complicated by variable and vast workload patterns, I/O devices behave more predictably. Previous work [45] indicates that most devices demonstrate strong temporal locality in I/O access in current operating systems with proper driver support. Typically, DMA accesses are confined to a small, consistent portion of VM memory (e.g., under 5%) over an hour, with deviations mainly occurring during significant VM memory contention.

This supports our design direction. If this rule applies to both NIC and block devices in production, we can track I/O footprints and manage recently accessed DMA buffers in an LRU set with pinning, potentially preventing most IOPFs. We conducted a study in our production server fleets, selecting 2000 VMs and implementing DMA tracking software in the DPUs to log detailed DMA access data, including timestamps, BDF, DMA addresses, sizes, and device types. Logs were collected for 30 minutes, and we calculated the percentage of 2MB pages accessed by the NIC and block devices. The results are shown in Fig 9, revealing distinct differences between the two.



(a) Percentile footprint of NIC and block devices on 2000 VMs



(b) Devices of 20 VMs from a server. Each marker is a device

Figure 9. I/O footprints from NIC and block devices in 30 minutes.

O1: The I/O footprint of most NIC devices is quite small, with the DMA buffer being less than 4% at the 95th percentile over a 30-minute window, and it is independent of workload intensity. A larger footprint (12.39%) is observed only at the 98th percentile, but it is still much smaller compared to block devices.

O2: Block devices produce large footprints, which we speculate is caused by the page cache. In Linux, new page caches are allocated in the kernel for newly accessed blocks during file read and write operations, and DMA is performed directly on these page caches to reduce buffer copying.

Moreover, over the course of weeks, the average footprint grows to 17% for NICs and 53% for block devices, respectively, indicating a gradual shift in the footprints. It can be speculated that if DMA buffers were to be pinned equally, the majority of the system memory would become pinned over time, an outcome that is unacceptable. A straightforward solution can employ a simple LRU set of bounded size (e.g., 5% of VM memory) for each VM, to track and pin the most recently used DMA - but it lacks isolation. For example, block devices have large footprints, causing the DMA buffers of NIC devices to be frequently evicted from the same LRU set. This leads to suboptimal IOPF protection for NIC devices, even though they have a small footprint.

Existing CPUs failed to meet the demand for optimal I/O footprint tracking. The latest Intel IOMMU (Sapphire Rapids) introduces the second-level access and dirty (SLAD) feature [16], which is akin to the CPU's approach of managing A/D bits in page tables. However, it is not without its shortcomings: 1) The IOPT is shared by all devices within the same VM, and the set of A/D bits in the IOPT entry does not indicate which specific device accessed the page and 2) the periodic scanning of IOPT in the software is inaccurate, potentially leading to the loss of details regarding temporal locality and 3) scanning page tables, particularly with large physical memory, incurs significant overhead and can impact I/O performance due to IOTLB shootdowns. Modern CPUs provide event-based mechanisms such as precise event-based sampling (PEBS) and page modification logging (PML) for software [21, 31] to optimize memory access tracking. However, they cannot be used for tracking DMA accesses.

Thus, our first key insight is: tracking I/O footprints needs to differentiate between devices and implement protection strategies on a per-device basis with low overhead. To this end, we developed a fine-grained I/O Page Access Logging (IO-PAL) engine (§5.3) within a DPU.

4.2.2 Simulated IOPF.

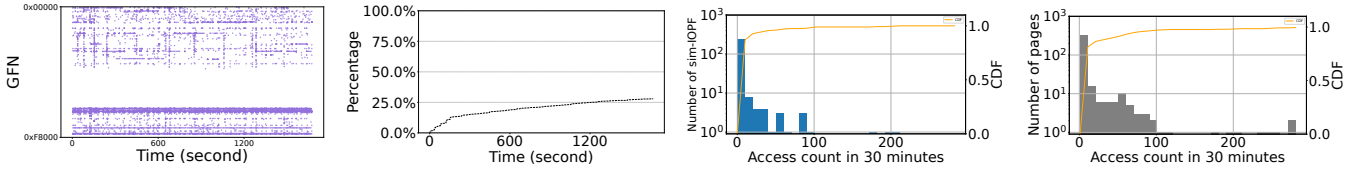
This section conducts an in-depth analysis of the differences between NIC and block devices to evaluate the efficiency of using per-device LRU sets in reducing IOPF.

To quantitively study the effectiveness of a pinning strategy, we define *IOPF* ***R****eduction* to ***P****in proportion* ***R****atio* (**RPR**)– A metric to reflect the efficiency of the protection policy, the higher the better.

$$RPR = \frac{percentage\_of\_reduced\_iopf}{percentage\_of\_pinned\_pages} \quad (2)$$

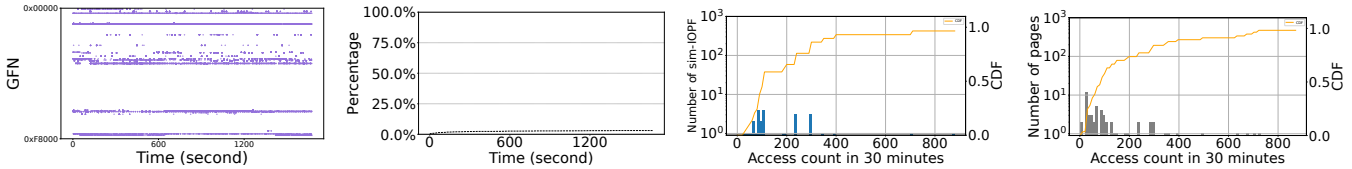
For example, achieving a 50% IOPF reduction by pinning 10% of a VM's memory results in an RPR of 5. Achieving a 30% IOPF reduction by pinning 30% of a VM's memory results in a worse RPR of 1.

We extract temporal locality-related characteristics for each 2MB page, including interval of consecutive accesses, and access count, from the data collected in §4.2.1. We use the access interval of each page to form a distribution of simulated IOPFs. The method to simulate IOPF is as follows: if two consecutive accesses of a page exceed a certain threshold, it is considered that an IOPF has likely occurred. This threshold depends on the reclamation policy and the swap-out ratio in the memory subsystem. In this simulation, we set the threshold at 300 seconds. Despite imprecision, due to the exclusion of other factors such as CPU access, memory



(a) Raw footprints **(b)** Percentage of 2M pages accessed by IO **(c)** Histogram of simulated IOPF bucketed by access count **(d)** Number of pages bucketed by access count

Figure 10. I/O locality analysis for a typical block device over 30 minutes. The memory accessed by DMA increases gradually without bounds to approximately 28% of the VM's memory. 87.3% of pages are accessed fewer than 10 times, contributing to approximately 94.3% of simulated IOPFs. Meanwhile, 70.3% of the pages are accessed more than twice.



(a) Raw footprints **(b)** Percentage of 2M pages accessed by IO **(c)** Histogram of simulated IOPF bucketed by access count **(d)** Number of pages bucketed by access count

Figure 11. I/O locality analysis for a typical NIC device over 30 minutes. The memory accessed by DMA is within a clearly bounded range (3% of VM's memory). The number of simulated IOPFs is significantly lower than that of the block device.

manager's LRU policy/ordering, and the actual reclaim ratio, the simulated count of IOPFs is still a good indicator of the likelihood of IOPFs occurring.

Fig 10 is from a typical block device. Observations:
O3: For block devices, most pages accessed by DMA exhibit weak temporal locality, with more than 83% of pages accessed fewer than 10 times in 30 minutes (Fig 10d). Once a page is accessed by a block device, there's a high probability (>70%) it will be accessed again by the same device in the next 30 minutes. Simulated IOPFs mostly occur on the pages (Fig 10c) of low access count, with pages accessed fewer than 50 times contribute to nearly all of the simulated IOPFs. Since a block device has an unbounded and growing DMA footprint (Fig 10b), frequent eviction occurs at the tail of the LRU set. Nevertheless, the evicted pages, with longer recency, are more prone to page faults than the pages at the head of the LRU set. This necessitates an expansion of the LRU capacity (along with the number of pinned pages) to achieve a meaningful reduction in IOPFs, which may impact the memory utilization rate and result in a low RPR score. §7.3 demonstrates that the RPR of simple LRU set on block devices falls within a range of 1.2 to 1.5.

By comparison, Fig 11 is from a typical NIC device:
O4: For most NIC devices, DMA pages are bound to 2%-5% of VM memory (Fig 11b). By pinning these pages with an LRU strategy, we can achieve an IOPF reduction ratio close to 100%, with the exception of IOPFs that occur upon the first access to a page. Consequently, the estimated RPR could be roughly between 20 and 50.

Our second insight from further analysing the temporal locality is: simple LRU works poorly for block devices as protection policy, but can serve NIC devices well. Whether for NICs or block devices, a better policy is needed (§6.2) to minimize the IOPF rate while reducing pinned memory.

5 VPRI Hardware Design

Based on our analysis and insights, we developed a novel IOPF system on the DPU, termed VPRI, as illustrated in Fig 12. VPRI is a physical function on the DPU, offering multiple virtio rings that are managed by drivers in the hypervisor/host environment. The key features of VPRI include:
1. **PA-BITMAP offloading**: The DPU stores page attributes for all IOVAs in its DRAM, synchronized with the host IOPT via an event message queue (EMQ). This facilitates pre-DMA lookups, removing the reliance on the IOMMU ATS.
2. **Sideband IOPF interface**: The DPU transmits IOPF messages via a shared page fault queue (PFQ) for all VFs within the same DPU, removing the reliance on the IOMMU PRI.
3. **I/O Page Access Logging (IO-PAL)**: Offers the hypervisor fine-grained tracking capabilities and minimal overhead for the I/O footprint of all VFs within the DPU.

5.1 PA-BITMAP Synchronization

In DPU design, equipping DRAM to expand storage and compensate for limited BRAM/SRAM resources is common [49]. DDR DRAM has higher latency (~50-100ns) than BRAM/SRAM but offers much larger space in GBs. We placed the PA-BITMAP in the DPU's DRAM because its simple data structure (§4.1) allows for a single DRAM access per 4K DMA

range during pre-DMA lookups. The evaluation shows that such bitmap lookup latency has a negligible impact on base performance across all configurations of our DPU models, which operate at a maximum line rate of 50-200 Gbps.

To fully synchronize the IOPTs with PA-BITMAPs, the following requirements need to be satisfied:

R1: The creation/destruction of a PA-BITMAP within the device should be synchronized with the creation/destruction of the VM and the corresponding IOPT.

R2: When the IOPT mapping changes, the corresponding PAEs in the device's PA-BITMAP must be updated coherently and atomically.

R3: Before a DMA, the IO engine atomically looks up in the PA-BITMAP to determine the presence of the memory region to be accessed.

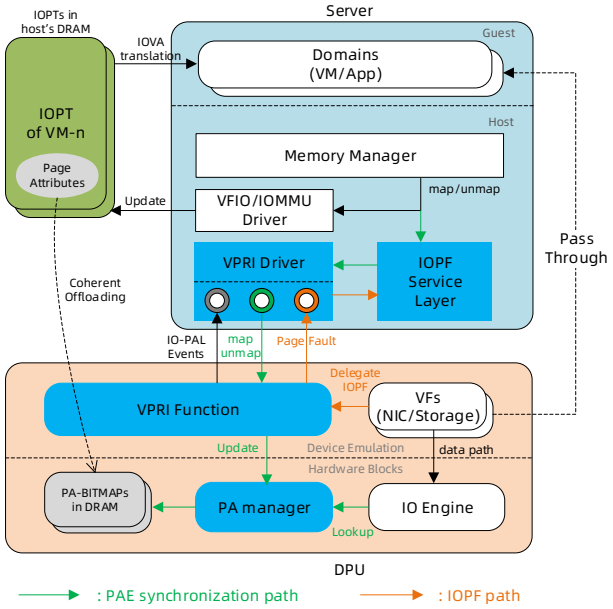


Figure 12. VPRI system overview.

We use QEMU-KVM as an example for synchronization. In virtualization, a VM involves four sets of page tables that need to be synchronized:

- The host page table of the QEMU process, responsible for HVA to HPA translation.
- The EPT, which is responsible for GPA to HPA translation in non-root mode.
- The IOPT of the IOMMU, responsible for the IOVA (GPA) to HPA translation of I/O operations.
- Device specific attributes, which in the case of ATS would be the DevTLB, and in the case of VPRI, would be the PA-BITMAP.

To properly manage race conditions during a page unmap operation involving inflight I/O, the hypervisor must invalidate the page table entries in a specific sequence before safely reclaiming the memory. Specifically, the VPRI driver must invalidate the corresponding bit in the PA-BITMAP **before** it proceeds to unmap the IOPT. This is done by issuing a synchronized bitmap invalidation command to the DPU through the EMQ of the VPRI function. This command will invalidate the corresponding PAEs (1 for a 4KB page and 512 for a 2MB page), ensuring that the device cannot access the memory that is in the process of being reclaimed. Before the device can respond, the PA manager in the DPU must atomically set the corresponding PAEs to the *unmapped* state, and any (including inflight) I/O requests associated with this page will trigger an IOPF, thereby being synchronized by the host software. Then the hypervisor can safely unmap IOPT and reclaim the physical page.

5.2 IOPF Interface

When the backend I/O engine needs to report a page fault from a queue, it generates a page request message. This message contains metadata related to the identification of the faulted queue/device and the fault address. The message is placed in the PFQ of the VPRI device, prompting an interrupt to the virtual machine monitor (VMM) via the VPRI device. The I/O queue that encountered the fault is paused to avoid out-of-order I/O processing, until the I/O engine gets a page response from the driver. The interface and semantics of VFs do not need to be changed, ensuring that IOPF remains transparent to VMs and device drivers. The way VPRI completes an IOPF is similar to the standard PRI of IOMMU, but it is more efficient and most importantly, no longer depends on the platform IOMMU.

5.3 IO-PAL

IO-PAL operates through a dedicated virtio-ring on the VPRI device. Only when the hypervisor needs to receive PAL events does it batch and submit a set of descriptors to the available ring. The device reduces CPU utilization through batch processing and interrupt aggregation. Specifically, the device asynchronously fills these descriptors with PAL events and moves them to the used ring. When the available descriptors in the ring are nearly exhausted, an interrupt is triggered, prompting the driver to process the PAL events in bulk and refill the descriptors. Since IO-PAL can tolerate event loss in most cases (only affecting the accuracy of I/O footprint tracking), the driver can regulate the rate at which descriptors are added to the available ring to manage CPU consumption. If the available ring's descriptors are empty, the DPU discards the IO-PAL events.

6 VPRI Software Design

To ensure that the IOPF core processing functions in the software can be compatible with different IOPF hardware simultaneously, we have abstracted the software portion of the IOPF process into a kernel module called IOPF service

layer to specifically handle scheduling and DMA protection, independent of any IOPF hardware. Fig 13 shows the design.

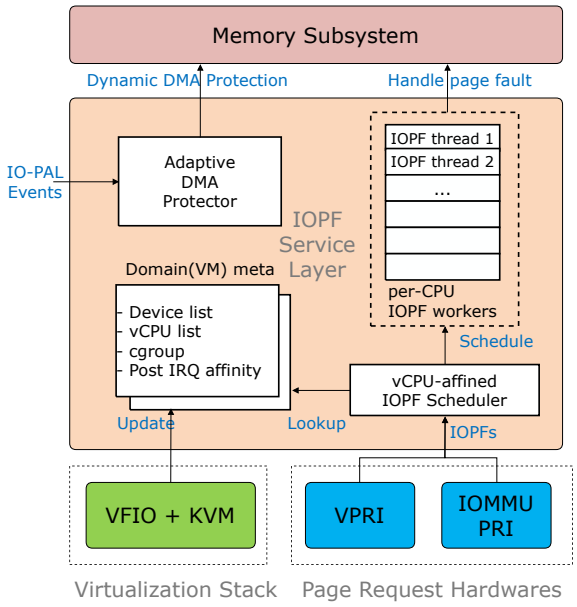


Figure 13. Virtualization IOPF software overview.

6.1 vCPU-affined IOPF Scheduling

The vCPU-affined scheduler in the IOPF service layer mitigates scalability and isolation issue of IOPF by taking advantage of parallelism and out-of-order processing for IOPFs. It supports both VPRI-based VFs and devices that utilize the standard IOMMU PRI. When a hardware IOPF request (either from VPRI or IOMMU) is dispatched to the IOPF service layer, the vCPU-affined scheduler picks the most suitable physical core to schedule the bottom-half IOPF thread.

Firstly, it identifies the set of physical cores owned by the tenant that triggered the IOPF, ensuring that this IOPF does not interrupt the vCPUs of other tenants. Secondly, it attempts to identify the physical CPU on which the faulted device registers its MSI interrupt using the IOMMU post-interrupt (PI) descriptor [8]. For polling mode devices that do not register interrupts, the scheduler selects a physical core from the set based on the current CPU load or employs a round-robin mechanism when the CPU loads are evenly distributed. Finally, once a physical core is chosen, the scheduler initiates a VM-Exit by sending an inter-processor interrupt to this core, prompting it to execute the bottom half of the IOPF handling.

6.2 Adaptive DMA Protector

To effectively reduce the IOPF rate while minimizing pinned memory overhead (§4.2), we designed a heuristic LRU policy called the adaptive DMA protector (ADP).

The concept behind ADP is to exploit temporal locality and to more rigorously reduce the duration for which a page is pinned, rather than pinning each page equally within the LRU set. With simple LRU, when a memory page is accessed, its page metadata moves to the most recently used position in the LRU set and is pinned immediately. However, it should not be immediately pinned, because the memory subsystem will also be aware of this access (by CPU) during the page table (EPT in virtualization) scan and will treat it as a hot page. Only if this page is not accessed again by DMA after a prolonged interval will the memory subsystem be likely to reclaim it. Thus, we only pin the page if the interval since its last access surpasses a specific threshold. This filters out pages that do not need to be pinned at the moment, thereby reducing the number of pinned pages.

Based on this heuristic policy, we designed a filtering-based dual LRU policy that adapts to both NIC and block devices. The design of ADP is two-fold, as shown by Fig 14:

1. Each device has its own protection domain. When IO access logs are received, the software directs them to the corresponding protection domain of the device, using the device's BDF as the routing indicator.

2. Each protection domain has two doubly-linked LRU lists, the active block and inactive block, to store pages with short recency (or access interval) and long recency respectively. Only pages in the inactive block are pinned to reduce the number of pinned pages.

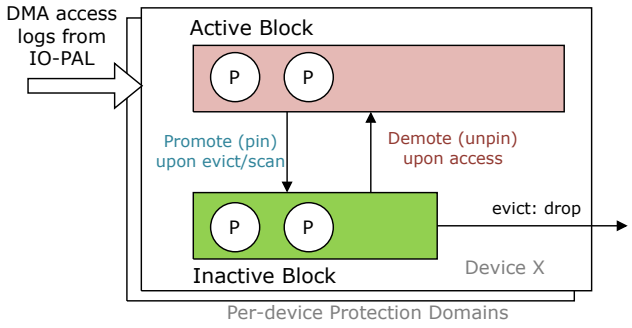


Figure 14. Adaptive DMA Protector.

When a device accesses a (2MB) page for the first time, ADP creates metadata for it and stores it in the active block of the corresponding device's protection domain. This metadata records the last access timestamp and the IOVA page number. The active block can hold up to 30% of the VM's total memory, while the inactive block allows up to 5% of pages to be pinned per device. Metadata in each block is sorted by the last access interval from shortest to longest.

A page can be promoted from active block to inactive block in two ways: 1) when the active block evicts a page from the tail when it overflows or 2) when the interval since the last access to the page exceeds the threshold determined by the promotion daemon thread. To reduce scanning overhead, the promotion thread scans in reverse every 20 seconds within

each protection domain, starting from the tail, and stops when it encounters a page metadata entry with a last access interval shorter than the threshold. The threshold reflects the reclaim policy in the memory subsystem; we set it to 180 seconds empirically. Furthermore, if an IO-PAL event occurs for a page whose metadata is stored in the inactive block, it indicates that the page is transitioning from a cold state to a hot state. The memory subsystem will recognize this change and will no longer require protective measures for that page. Consequently, the corresponding metadata will be demoted to the active block after a delay of 30 seconds to compensate for the delay of the page table scan in the memory subsystem.

ADP is orthogonal to the LRU/MGLRU in the memory subsystem, yet highly efficient in guiding the memory subsystem's swap policy to avoid reclaiming pages with high risk of causing IOPF. Additionally, this isolation helps to prevent the memory subsystem from becoming overly complicated, and maintains its agnosticism towards I/O semantics.

7 Evaluation

In this section, we will demonstrate that VPRI effectively addresses the aforementioned challenges of IOPF by quantitatively answering the following questions.

Q1: Does the low-cost PA-BITMAP mechanism pose any negative performance impact on the critical data path? (§7.1)

Q2: How critical is the scalability issue of IOPF? Does vCPU-affined scheduling effectively address the scaling challenges associated with concurrent IOPFs? (§7.2)

Q3: Is ADP able to significantly reduce IOPFs to alleviate performance issues with a negligible pinning tax? (§7.3)

Q4. In production environments, does VPRI effectively minimize the impact on SLOs within an acceptable range? (§7.4)

The hardware configurations are shown in Table 1. All VFs are on the same DPU and operate in pass-through mode.

Type	Hardware Configuration
Host	Intel(R) Xeon(R) CPU Platinum 8269CY CPU: 52 cores/104 threads@2.50GHz in 2 sockets 512GB DRAM, 1TB SSD
DPU	Connection: PCIe GEN3, 8 lanes Device emulation: up to 2300 virtio-net, virtio-blk VFs Max physical network bandwidth: 200 Gb/s
VM	4 vCPUs, 8GB RAM CentOS Linux release 7.9.2009 NIC device: dual queue virtio-net x1 10Gb/s max throughput, 190,000 max PPS Block device: virtio-blk x1 400MB/s max throughput, 26800 max IOPS

Table 1. Evaluation configuration.

7.1 PA-BITMAP Micro-benchmark

Table 2 shows the lookup and update performance of PA-BITMAP. The bitmap lookup in the DPU resides on the critical path and can potentially impact baseline I/O performance. In this experiment, we disabled the DPU's caching during pre-DMA lookup operations to assess the worst-case impact on baseline I/O performance. Measurements indicate that the lookup time ranges between 50-100 ns. We tested I/O-intensive benchmarks listed in Fig 4 on 50-200 Gbps DPU models with varying packet sizes, thread counts, and I/O depths and found no differences in maximum throughput, PPS, or IOPS metrics after introducing bitmap lookup. Only a negligible difference of less than 0.1% was casually observed with smaller packet size.

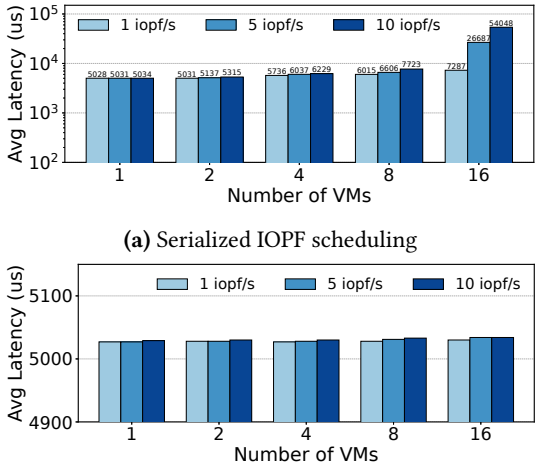
Metric	On-device lookup (ns)	Host update (ns)
Average	69	1,742
P99th	92	2,341

Table 2. Latency of PA-BITMAP operations.

The results remained consistent even after we manually injected an additional 100 ns latency into the PA-BITMAP lookup engine. This demonstrates that the pre-DMA lookup based on PA-BITMAP has a negligible impact on the I/O pipeline, primarily due to the longer I/O round-trip times and the parallelism inherent in the device (§4.1).

The end-to-end bitmap update from the host takes approximately 2 µs, primarily due to PCIe latency. This latency is factored into the unmap/invalidation phase of the reclaim procedure, which is not on the critical path. Similarly, the standard ATS+PRI also incurs PCIe overhead for DevTLB invalidation. In the IOPF path, the device updates the bitmap automatically upon receiving a page fault response from the software, eliminating the need for the software to explicitly update it.

7.2 vCPU-affined Scheduling



(a) Serialized IOPF scheduling

(b) vCPU-affined IOPF scheduling

Figure 15. Concurrent IOPF service latency.

To demonstrate IOPF's scaling issue, we launched varying numbers of VMs coexisting on the compute node. Each VM was injected with page faults by the hypervisor in parallel at different rates, causing the I/O engine to generate concurrent IOPFs via the VPRI-PCI device. We set the page fault latency to 5,000 µs on the software side to match the tail case swap latency in our production environment. Fig 15a illustrates the average latency when all IOPFs are queued in a single-threaded IOPF worker, showing a significant 10.4× increase in latency as the fault rate reaches 10 per second with 16 VMs. Conversely, Fig 15b demonstrates that when vCPU-affined IOPF scheduling (§6.1) is enabled, latency remains consistently stable across all fault rates. Moreover, it ensures that the IOPF thread is isolated within the tenant's CPU set.

7.3 IOPF Reduction and Pinned Memory

We create a synthetic environment with a mixture of I/O-intensive workloads to evaluate the effect of ADP on both IOPF reduction and the percentage of pinned memory in this experiment. We also test a simple per-device LRU strategy with a static pin ratio of 10% to compare with ADP. The synthetic environment is as follows:

- 14 VMs, each with memory sizes randomly ranging from 8 to 64 GB and assigned to a specific workload.
- We selected a diverse range of in-house workloads encompassing data analytics, caching, serving, graph analytics, media processing, streaming, web search, and web serving. Of the 14 VMs, 4 are assigned to networking-intensive workloads and 10 are assigned to storage-intensive workloads.
- We test three overall memory overcommit ratios: 10%, 20%, and 30%. The reclaim ratio for each VM is dynamically adjusted from 0% to 50% to match the production environment.
- All VMs run concurrently for one hour as a round of reproducible testing.
- After each round, all VMs are recreated to clear any residual state.

This evaluation uses the total IOPF numbers from all NICs and block devices under varying overcommit ratios without any pinning as baseline. The workload intensity is set to 80%, higher than typical production environments, to ensures each testing round generates significant number of IOPFs. Although the IOPF distribution and workloads may not perfectly mirror real-world conditions, this setup provides valuable insights into IOPF reduction and pin ratios.

Fig 16 shows the IOPF reduction and pinned footprint of NIC and block devices, respectively. Overall, ADP reduced IOPFs by 95-97% on NIC devices and by 36-49% on block devices, with an average pinned memory per VM of approximately 4.3%. Conversely, the simple LRU strategy achieved similar IOPF reduction on NIC devices but only reduced

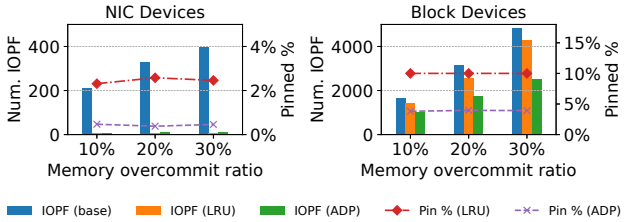


Figure 16. Number of IOPFs and pinned footprints. The left Y-axis represents the total IOPFs of all devices in a single test round, while the right Y-axis indicates the average proportion of pinned memory in each device.

IOPFs by 12-15% on block devices, with an average pinned memory per VM reaching around 12%. Regarding the RPR metric, which reflects pinning efficiency, ADP outperformed simple LRU by 6.28× on NIC devices and 10.56× on block devices.

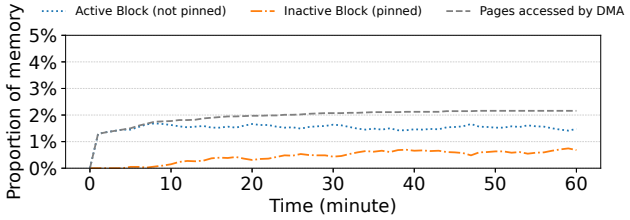


Figure 17. Utilization capacity over 60 minutes for a typical NIC device in an ADP domain.

Fig 17 is an example of a typical NIC device that showcases the changes in capacity for active and inactive blocks inside an ADP domain over 60 minutes. The fluctuations in the inactive block reflect the dynamic pin ratio of ADP. Conversely, with simple LRU, The pin ratio corresponds to the growth trajectory of the I/O footprints.

7.4 VPRI in Production

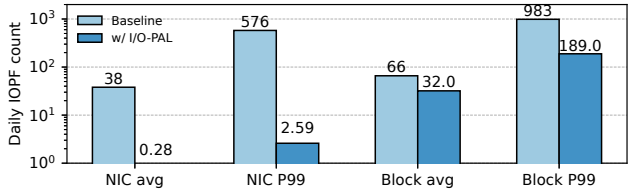


Figure 18. Daily IOPF per device of 5000 VMs from production with VPRI.

With VPRI's I/O-PAL feature and ADP in the hypervisor, we were able to maintain the memory overcommit ratio at 10-15% in our data centers. We collected IOPF data from 5,000 VMs in the production environment and compared it with

earlier data without I/O-PAL under the same overcommitment ratio, as shown by Fig 18. Notably, the average IOPF reduction was over 99% on NIC devices and 51% on block devices, while maintaining an average pinned memory per VM at 5.2%. After a thorough observation in the production environment over the course of a year, we have concluded that the near-elimination of IOPFs on NIC devices and the significant reduction of IOPFs on block devices, particularly at the 99th percentile, has resulted in only a minimal increase in I/O jitters caused by IOPFs. As a result, their impact on I/O SLO metrics has been reduced to an acceptable level. This allows the hardware IOPF system to coexist with memory overcommitment at a profitable ratio without any customer complaints about I/O issues caused by IOPFs in our data centers, unlocking even greater potential.

8 Related and Future Work

Tackling static memory pinning. Apart from ATS+PRI, there has been considerable research focused on avoiding static memory pinning. Previous studies [23, 39, 52] have provided IOPF for RDMA NICs using interrupt-based model. Ivan Tanasic et al. [44] propose an efficient IOPF mechanism for GPUs that allows code to preempt and restart execution. IOGuard [9] dedicates a host CPU to enable software-based IOPFs. Other research, like coIOMMU [28, 46], leverages cooperative DMA memory tracking for dynamic memory pinning.

However, IOPF designs specific to RDMA NICs [23, 39], disaggregated memory systems [15], and GPUs [44] do not apply to generic block and network devices. The pitfalls of RDMA-specific IOPF can degrade performance in existing software systems [11]. GPU-specific IOPF relies on costly on-device MMUs and TLBs. IOGuard [9] uses valuable CPU resources to handle periodic IOPFs. Software-based dynamic memory pinning [28, 46, 48] is not transparent to VMs. In contrast, VPRI enables platform-independent, high-performance, and cost-effective IOPFs for public clouds.

Avoiding packet drops for NIC during IOPFs. One potential solution for eliminating packet drops during IOPFs without driver modification is to leverage out-of-order RX buffer/descriptor consumption on the device side. This approach would allow efficient processing of incoming packets without modifying the NIC driver, thus remaining agnostic to the specific driver being used.

Optimization for block devices. ADP reduces IOPF rates on block devices by half on average but is still suboptimal compared to NIC devices. Furthermore, it lacks efficiency for different workload characteristics. We propose that future optimization can be achieved by allowing ADP to dynamically adjust its parameters based on online profiling to better suit varying workloads. Additionally, we suggest exploring approaches that leverage spatial locality on top of I/O-PAL and applying prefetching techniques.

Platform compatibility. We currently only offer support for VPRI on x86 platforms, but extending this support to other platforms, such as ARM, would require minimal effort.

More types of devices and workloads. This paper targets IOPF for virtio-based devices. We have also fully supported IOPF for RDMA and NVMe over fabrics [5, 32, 37] on the VPRI interface in our next product using the same methodology. Supporting IOPF for AI workloads on GPU also needs to be explored.

Cacheability of PA-BITMAP. For future low-latency device interfaces like CXL [41], the PA-BITMAP lookup latency in DRAM may become non-trivial. Modern DPUs' processors (e.g., ARM, RISC-V, Intel Atom) have dedicated caches that can help reduce this latency. While the PA-BITMAP may reach hundreds of MB in high-density cloud environments, the active section usually represents under 5% within a minute, similar to the I/O footprint. This active portion easily fits into the DPU's cache, leading to a high cache hit rate and minimizing I/O latency impact.

Nested IOPF. The modern IOMMU facilitates nested IOPF through the transmission of IOPF interrupts from the host to the guest via a para-virtualized IOMMU [4]. However, this approach results in a significantly complex vIOMMU emulation [3, 27, 29, 30, 36] in the hypervisor, making it difficult for CSPs to support a variety of CPU vendors. Additionally, it introduces latency overhead due to VM-Exits and hypervisor overhead for IOPF interrupt injection. Potentially, the VPRI interface can be modified and passed through to VMs, allowing for nested page fault support in VMs without requiring hypervisor modifications or vIOMMU support.

9 Conclusion

This paper presents VPRI as a compelling alternative for CSPs and device vendors to support IOPF in a manner that is more cost-effective, compatible, and performant. We implemented VPRI in our existing DPUs in under six months without requiring additional hardware upgrades, effectively addressing the cost concerns of CSPs. Additionally, it enabled hardware IOPF capability on previously released x86 systems from Intel and AMD that lacked ATS and PRI capabilities, effectively resolving compatibility issues. Meanwhile, VPRI effectively mitigated the impact of hardware IOPFs on SLOs, ensuring they remained within an acceptable range when memory commitment was enabled in our data centers.

Acknowledgments

We sincerely thank our shepherd Andrew Baumann and anonymous reviewers for their insightful suggestions. We would also like to acknowledge the contributions of Dao Ren, Zelong Wang, Jie Ji, Dongdong Huang and numerous other engineers from the Alibaba X-Dragon team who directly worked on VPRI. This work was partially supported by NSFC (No. 62372287) and Alibaba AIR project. Zeyu Mi (yzmizeyu@sjtu.edu.cn) is the corresponding author.

References

[1] Alexandru Agache, Marc Brooker, Alexandra Iordache, Anthony Liguori, Rolf Neugebauer, Phil Piwonka, and Diana-Maria Popa. 2020. Firecracker: Lightweight Virtualization for Serverless Applications. In *17th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2020, Santa Clara, CA, USA, February 25-27, 2020*, Ranjita Bhagwan and George Porter (Eds.). USENIX Association, 419–434. https://www.usenix.org/conference/nsdi20/presentation/agache

[2] Tyler N. Allen and Rong Ge. 2021. In-depth analyses of unified virtual memory system for GPU accelerated computing. In *International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2021, St. Louis, Missouri, USA, November 14-19, 2021*, Bronis R. de Supinski, Mary W. Hall, and Todd Gamblin (Eds.). ACM, 64. https://doi.org/10.1145/3458817.3480855

[3] Nadav Amit, Muli Ben-Yehuda, and Ben-Ami Yassour. 2010. IOMMU: Strategies for Mitigating the IOTLB Bottleneck. In *Computer Architecture - ISCA 2010 International Workshops A4MMC, AMAS-BT, EAMA, WEED, WIOSCA, Saint-Malo, France, June 19-23, 2010, Revised Selected Papers (Lecture Notes in Computer Science, Vol. 6161)*, Ana Lucia Varbanescu, Anca Mariana Molnos, and Rob van Nieuwpoort (Eds.). Springer, 256–274. https://doi.org/10.1007/978-3-642-24322-6_22

[4] Lu Baolu. 2024. IOMMUFD: Deliver IO page faults to user space. https://lwn.net/Articles/971820/ https://lwn.net/Articles/971820/.

[5] Shoaib Basu and Deepak Nadig. 2024. Offloading NVMe over Fabrics (NVMe-oF) to SmartNICs on an at-scale Distributed Testbed. In *10th IEEE International Conference on Network Softwarization, NetSoft 2024, Saint Louis, MO, USA, June 24-28, 2024*. IEEE, 316–318. https://doi.org/10.1109/NETSOFT60951.2024.10588915

[6] Abhishek Bhattacharjee, Daniel Lustig, and Margaret Martonosi. 2011. Shared last-level TLBs for chip multiprocessors. In *17th International Conference on High-Performance Computer Architecture (HPCA-17 2011), February 12-16 2011, San Antonio, Texas, USA*. IEEE Computer Society, 62–63. https://doi.org/10.1109/HPCA.2011.5749717

[7] Xuzheng Chen, Jie Zhang, Ting Fu, Yifan Shen, Shu Ma, Kun Qian, Lingjun Zhu, Chao Shi, Yin Zhang, Ming Liu, and Zeke Wang. 2024. Demystifying Datapath Accelerator Enhanced Off-path SmartNIC. *CoRR* abs/2402.03041 (2024). https://doi.org/10.48550/ARXIV.2402.03041 arXiv:2402.03041

[8] Intel Corporation. 2022. Intel® Virtualization Technology for Directed I/O. https://cdrdv2-public.intel.com/671081/vt-directed-io-spec.pdf https://cdrdv2-public.intel.com/671081/vt-directed-io-spec.pdf.

[9] Yiyuan Dong and Zeyu Mi. 2024. IOGuard: Software-Based I/O Page Fault Handling with One CPU Core. In *Proceedings of the 15th Asia-Pacific Symposium on Internetware* (Macau, China) *(Internetware '24)*. Association for Computing Machinery, New York, NY, USA, 337–346. https://doi.org/10.1145/3671016.3671394

[10] Takuya Fukuoka, Shigeyuki Sato, and Kenjiro Taura. 2021. Pitfalls of InfiniBand with On-Demand Paging. In *IEEE International Symposium on Performance Analysis of Systems and Software, ISPASS 2021, Stony Brook, NY, USA, March 28-30, 2021*. IEEE, 265–275. https://doi.org/10.1109/ISPASS51385.2021.00049

[11] Takuya Fukuoka, Shigeyuki Sato, and Kenjiro Taura. 2021. Pitfalls of InfiniBand with On-Demand Paging. In *2021 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. 265–275. https://doi.org/10.1109/ISPASS51385.2021.00049

[12] Google. [n. d.]. Google SRE Book. https://sre.google/sre-book/service-level-objectives/.

[13] Krishnan Gosakan, Jaehyun Han, William Kuszmaul, Ibrahim N. Mubarek, Nirjhar Mukherjee, Karthik Sriram, Guido Tagliavini, Evan West, Michael A. Bender, Abhishek Bhattacharjee, Alex Conway, Martin Farach-Colton, Jayneel Gandhi, Rob Johnson, Sudarsun Kannan, and Donald E. Porter. 2023. Mosaic Pages: Big TLB Reach with Small Pages. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3, ASPLOS 2023, Vancouver, BC, Canada, March 25-29, 2023*, Tor M. Aamodt, Natalie D. Enright Jerger, and Michael M. Swift (Eds.). ACM, 433–448. https://doi.org/10.1145/3582016.3582021

[14] Juncheng Gu, Youngmoon Lee, Yiwen Zhang, Mosharaf Chowdhury, and Kang G. Shin. 2017. Efficient Memory Disaggregation with Infiniswap. In *14th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2017, Boston, MA, USA, March 27-29, 2017*, Aditya Akella and Jon Howell (Eds.). USENIX Association, 649–667. https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/gu

[15] Zhiyuan Guo, Yizhou Shan, Xuhao Luo, Yutong Huang, and Yiying Zhang. 2022. Clio: a hardware-software co-designed disaggregated memory system. In *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems* (Lausanne, Switzerland) *(ASPLOS '22)*. Association for Computing Machinery, New York, NY, USA, 417–433. https://doi.org/10.1145/3503222.3507762

[16] https://lore.kernel.org/. [n. d.]. Access/Dirty bit support for SL domains. https://lore.kernel.org/all/20220428210933.3583-19-joao.m.martins@oracle.com/.

[17] Aamer Jaleel, Eiman Ebrahimi, and Sam Duncan. 2019. DUCATI: High-performance Address Translation by Extending TLB Reach of GPU-accelerated Systems. *ACM Trans. Archit. Code Optim.* 16, 1 (2019), 6:1–6:24. https://doi.org/10.1145/3309710

[18] Dario Korolija, Timothy Roscoe, and Gustavo Alonso. 2020. Do OS abstractions make sense on FPGAs?. In *14th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2020, Virtual Event, November 4-6, 2020*. USENIX Association, 991–1010. https://www.usenix.org/conference/osdi20/presentation/roscoe

[19] Nikita Lazarev, Varun Gohil, James Tsai, Andy Anderson, Bhushan Chitlur, Zhiru Zhang, and Christina Delimitrou. 2024. Sabre: Hardware-Accelerated Snapshot Compression for Serverless MicroVMs. In *18th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2024, Santa Clara, CA, USA, July 10-12, 2024*, Ada Gavrilovska and Douglas B. Terry (Eds.). USENIX Association, 1–18. https://www.usenix.org/conference/osdi24/presentation/lazarev

[20] Gyusun Lee, Seokha Shin, Wonsuk Song, Tae Jun Ham, Jae W. Lee, and Jinkyu Jeong. 2019. Asynchronous I/O Stack: A Low-latency Kernel I/O Stack for Ultra-Low Latency SSDs. In *2019 USENIX Annual Technical Conference, USENIX ATC 2019, Renton, WA, USA, July 10-12, 2019*, Dahlia Malkhi and Dan Tsafrir (Eds.). USENIX Association, 603–616. https://www.usenix.org/conference/atc19/presentation/lee-gyusun

[21] Taehyung Lee, Sumit Kumar Monga, Changwoo Min, and Young Ik Eom. 2023. MEMTIS: Efficient Memory Tiering with Dynamic Page Classification and Page Size Determination. In *Proceedings of the 29th Symposium on Operating Systems Principles, SOSP 2023, Koblenz, Germany, October 23-26, 2023*, Jason Flinn, Margo I. Seltzer, Peter Druschel, Antoine Kaufmann, and Jonathan Mace (Eds.). ACM, 17–34. https://doi.org/10.1145/3600006.3613167

[22] Ilya Lesokhin, Haggai Eran, Shachar Raindel, Guy Shapiro, Sagi Grimberg, Liran Liss, Muli Ben-Yehuda, Nadav Amit, and Dan Tsafrir. 2017. Page Fault Support for Network Controllers. In *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS 2017, Xi'an, China, April 8-12, 2017*, Yunji Chen, Olivier Temam, and John Carter (Eds.). ACM, 449–466. https://doi.org/10.1145/3037697.3037710

[23] Ilya Lesokhin, Haggai Eran, Shachar Raindel, Guy Shapiro, Sagi Grimberg, Liran Liss, Muli Ben-Yehuda, Nadav Amit, and Dan Tsafrir. 2017. Page Fault Support for Network Controllers. In *Proceedings of the Twenty-Second International Conference on Architectural Support for*

Programming Languages and Operating Systems (Xi'an, China) *(ASPLOS '17)*. Association for Computing Machinery, New York, NY, USA, 449–466. https://doi.org/10.1145/3037697.3037710

[24] Bingyao Li, Jieming Yin, Youtao Zhang, and Xulong Tang. 2021. Improving Address Translation in Multi-GPUs via Sharing and Spilling aware TLB Design. In *MICRO '21: 54th Annual IEEE/ACM International Symposium on Microarchitecture, Virtual Event, Greece, October 18-22, 2021*. ACM, 1154–1168. https://doi.org/10.1145/3466752.3480083

[25] Yang Lin, Dunbo Zhang, Chaoyang Jia, Qiong Wang, and Li Shen. 2021. Reducing TLB Miss Penalty on GPUs via Unified Multi-level PWB and PWC. In *12th International Symposium on Parallel Architectures, Algorithms and Programming, PAAP 2021, Xi'an, China, December 10-12, 2021*. IEEE, 1–8. https://doi.org/10.1109/PAAP54281.2021.9720477

[26] Yanqiang Liu, Jiacheng Ma, Zhengjun Zhang, Linsheng Li, Zhengwei Qi, and Haibing Guan. 2021. MEGATRON: Software-Managed Device TLB for Shared-Memory FPGA Virtualization. In *58th ACM/IEEE Design Automation Conference, DAC 2021, San Francisco, CA, USA, December 5-9, 2021*. IEEE, 1213–1218. https://doi.org/10.1109/DAC18074.2021.9586197

[27] Daniel Lustig, Abhishek Bhattacharjee, and Margaret Martonosi. 2013. TLB Improvements for Chip Multiprocessors: Inter-Core Cooperative Prefetchers and Shared Last-Level TLBs. *ACM Trans. Archit. Code Optim.* 10, 1 (2013), 2:1–2:38. https://doi.org/10.1145/2445572.2445574

[28] Chen Lv, Fuxin Zhang, Xiang Gao, and Chen Zhu. 2022. LAvIOMMU: An Efficient Hardware-Software Co-design of IOMMU Virtualization. In *2022 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking (ISPA/BDCloud/SocialCom/SustainCom)*. 246–253. https://doi.org/10.1109/ISPA-BDCloud-SocialCom-SustainCom57177.2022.00038

[29] Alex Markuze, Adam Morrison, and Dan Tsafrir. 2016. True IOMMU Protection from DMA Attacks: When Copy is Faster than Zero Copy. In *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS 2016, Atlanta, GA, USA, April 2-6, 2016*, Tom Conte and Yuanyuan Zhou (Eds.). ACM, 249–262. https://doi.org/10.1145/2872362.2872379

[30] Alex Markuze, Igor Smolyar, Adam Morrison, and Dan Tsafrir. 2018. DAMN: Overhead-Free IOMMU Protection for Networking. In *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS 2018, Williamsburg, VA, USA, March 24-28, 2018*, Xipeng Shen, James Tuck, Ricardo Bianchini, and Vivek Sarkar (Eds.). ACM, 301–315. https://doi.org/10.1145/3173162.3173175

[31] Hasan Al Maruf, Hao Wang, Abhishek Dhanotia, Johannes Weiner, Niket Agarwal, Pallab Bhattacharya, Chris Petersen, Mosharaf Chowdhury, Shobhit O. Kanaujia, and Prakash Chauhan. 2023. TPP: Transparent Page Placement for CXL-Enabled Tiered-Memory. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3, ASPLOS 2023, Vancouver, BC, Canada, March 25-29, 2023*, Tor M. Aamodt, Natalie D. Enright Jerger, and Michael M. Swift (Eds.). ACM, 742–755. https://doi.org/10.1145/3582016.3582063

[32] Darren Ng, Andrew Lin, Arjun Kashyap, Guanpeng Li, and Xiaoyi Lu. 2024. NVMe-oPF: Designing Efficient Priority Schemes for NVMe-over-Fabrics with Multi-Tenancy Support. In *IEEE International Parallel and Distributed Processing Symposium, IPDPS 2024, San Francisco, CA, USA, May 27-31, 2024*. IEEE, 519–531. https://doi.org/10.1109/IPDPS57955.2024.00052

[33] Nvidia. [n. d.]. InfiniBand Networking Solutions. https://www.nvidia.com/en-us/networking/products/infiniband/ https://www.nvidia.com/en-us/networking/products/infiniband/.

[34] Jihun Park, Donghun Jeong, and Jungrae Kim. 2023. UVMMU: Hardware-Offloaded Page Migration for Heterogeneous Computing. In *Design, Automation & Test in Europe Conference & Exhibition, DATE 2023, Antwerp, Belgium, April 17-19, 2023*. IEEE, 1–6. https://doi.org/10.23919/DATE56975.2023.10137307

[35] PCI-SIG. 2009. Address Translation Services Revision 1.1. https://pcisig.com/specifications/iov/ats/ https://pcisig.com/specifications/iov/ats/.

[36] Omer Peleg, Adam Morrison, Benjamin Serebrin, and Dan Tsafrir. 2015. Utilizing the IOMMU Scalably. In *2015 USENIX Annual Technical Conference, USENIX ATC '15, July 8-10, Santa Clara, CA, USA*, Shan Lu and Erik Riedel (Eds.). USENIX Association, 549–562. https://www.usenix.org/conference/atc15/technical-session/presentation/peleg

[37] Bo Peng, Cheng Guo, Jianguo Yao, and Haibing Guan. 2023. LPNS: Scalable and Latency-Predictable Local Storage Virtualization for Unpredictable NVMe SSDs in Clouds. In *2023 USENIX Annual Technical Conference, USENIX ATC 2023, Boston, MA, USA, July 10-12, 2023*, Julia Lawall and Dan Williams (Eds.). USENIX Association, 785–800. https://www.usenix.org/conference/atc23/presentation/peng

[38] Bharath Pichai, Lisa Hsu, and Abhishek Bhattacharjee. 2014. Architectural support for address translation on GPUs: designing memory management units for CPU/GPUs with unified address spaces. In *Architectural Support for Programming Languages and Operating Systems, ASPLOS 2014, Salt Lake City, UT, USA, March 1-5, 2014*, Rajeev Balasubramonian, Al Davis, and Sarita V. Adve (Eds.). ACM, 743–758. https://doi.org/10.1145/2541940.2541942

[39] Antonis Psistakis, Nikos Chrysos, Fabien Chaix, Marios Asiminakis, Michalis Gianioudis, Pantelis Xirouchakis, Vassilis Papaefstathiou, and Manolis Katevenis. 2022. Optimized Page Fault Handling During RDMA. *IEEE Trans. Parallel Distributed Syst.* 33, 10 (2022), 3990–4005. https://doi.org/10.1109/TPDS.2022.3175666

[40] Edward Richter and Deming Chen. 2022. Qilin: Enabling Performance Analysis and Optimization of Shared-Virtual Memory Systems with FPGA Accelerators. In *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design, ICCAD 2022, San Diego, California, USA, 30 October 2022 - 3 November 2022*, Tulika Mitra, Evangeline F. Y. Young, and Jinjun Xiong (Eds.). ACM, 23:1–23:9. https://doi.org/10.1145/3508352.3549431

[41] Henry N. Schuh, Arvind Krishnamurthy, David E. Culler, Henry M. Levy, Luigi Rizzo, Samira Manabi Khan, and Brent E. Stephens. 2024. CC-NIC: a Cache-Coherent Interface to the NIC. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1, ASPLOS 2024, La Jolla, CA, USA, 27 April 2024- 1 May 2024*, Rajiv Gupta, Nael B. Abu-Ghazaleh, Madan Musuvathi, and Dan Tsafrir (Eds.). ACM, 52–68. https://doi.org/10.1145/3617232.3624868

[42] Woong Shin, Qichen Chen, Myoungwon Oh, Hyeonsang Eom, and Heon Y. Yeom. 2014. OS I/O Path Optimizations for Flash Solid-state Drives. In *2014 USENIX Annual Technical Conference, USENIX ATC '14, Philadelphia, PA, USA, June 19-20, 2014*, Garth Gibson and Nickolai Zeldovich (Eds.). USENIX Association, 483–488. https://www.usenix.org/conference/atc14/technical-sessions/presentation/shin

[43] Junyi Shu, Kun Qian, Ennan Zhai, Xuanzhe Liu, and Xin Jin. 2024. Burstable Cloud Block Storage with Data Processing Units. In *18th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2024, Santa Clara, CA, USA, July 10-12, 2024*, Ada Gavrilovska and Douglas B. Terry (Eds.). USENIX Association, 783–799. https://www.usenix.org/conference/osdi24/presentation/shu

[44] Ivan Tanasic, Isaac Gelado, Marc Jorda, Eduard Ayguade, and Nacho Navarro. 2017. Efficient Exception Handling Support for GPUs. In *2017 50th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 109–122.

[45] Kun Tian, Yu Zhang, Luwei Kang, Yan Zhao, and Yaozu Dong. 2020. coIOMMU: A Virtual IOMMU with Cooperative DMA Buffer Tracking for Efficient Memory Management in Direct I/O. In *2020 USENIX*

Annual Technical Conference, USENIX ATC 2020, July 15-17, 2020, Ada Gavrilovska and Erez Zadok (Eds.). USENIX Association, 479–492. https://www.usenix.org/conference/atc20/presentation/tian

[46] Kun Tian, Yu Zhang, Luwei Kang, Yan Zhao, and Yaozu Dong. 2020. coIOMMU: A Virtual IOMMU with Cooperative DMA Buffer Tracking for Efficient Memory Management in Direct I/O. In *2020 USENIX Annual Technical Conference, USENIX ATC 2020, July 15-17, 2020*, Ada Gavrilovska and Erez Zadok (Eds.). USENIX Association, 479–492. https://www.usenix.org/conference/atc20/presentation/tian

[47] Pirmin Vogel, Andrea Marongiu, and Luca Benini. 2019. Exploring Shared Virtual Memory for FPGA Accelerators with a Configurable IOMMU. *IEEE Trans. Computers* 68, 4 (2019), 510–525. https://doi.org/10.1109/TC.2018.2879080

[48] Yaohui Wang, Ben Luo, and Yibin Shen. 2023. Efficient Memory Overcommitment for I/O Passthrough Enabled VMs via Fine-grained Page Meta-data Management. In *Proceedings of the 2023 USENIX Annual Technical Conference, USENIX ATC 2023, Boston, MA, USA, July 10-12, 2023*, Julia Lawall and Dan Williams (Eds.). USENIX Association, 769–783. https://www.usenix.org/conference/atc23/presentation/wang-yaohui

[49] Zeke Wang, Hongjing Huang, Jie Zhang, and Gustavo Alonso. 2020. Benchmarking High Bandwidth Memory on FPGAs. *CoRR* abs/2005.04324 (2020). arXiv:2005.04324 https://arxiv.org/abs/2005.04324

[50] Johannes Weiner, Niket Agarwal, Dan Schatzberg, Leon Yang, Hao Wang, Blaise Sanouillet, Bikash Sharma, Tejun Heo, Mayank Jain, Chunqiang Tang, and Dimitrios Skarlatos. 2022. TMO: transparent memory offloading in datacenters. In *ASPLOS '22: 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Lausanne, Switzerland, 28 February 2022 - 4 March 2022*, Babak Falsafi, Michael Ferdman, Shan Lu, and Thomas F. Wenisch (Eds.). ACM, 609–621. https://doi.org/10.1145/3503222.3507731

[51] Johannes Wünsche, Sajad Karim, Michael Kuhn, David Broneske, and Gunter Saake. 2023. Intelligent Data Migration Policies in a Write-Optimized Copy-on-Write Tiered Storage Stack. In *Proceedings of the 3rd Workshop on Challenges and Opportunities of Efficient and Performant Storage Systems, CHEOPS 2023, Rome, Italy, 8 May 2023*, Jean-Thomas Acquaviva, Shadi Ibrahim, and Suren Byna (Eds.). ACM, 17–26. https://doi.org/10.1145/3578353.3589543

[52] Jian Yang, Joseph Izraelevitz, and Steven Swanson. 2020. FileMR: Rethinking RDMA Networking for Scalable Persistent Memory. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*. USENIX Association, Santa Clara, CA, 111–125. https://www.usenix.org/conference/nsdi20/presentation/yang

[53] Weidong Zhang, Erci Xu, Qiuping Wang, Xiaolu Zhang, Yuesheng Gu, Zhenwei Lu, Tao Ouyang, Guanqun Dai, Wenwen Peng, Zhe Xu, Shuo Zhang, Dong Wu, Yilei Peng, Tianyun Wang, Haoran Zhang, Jiasheng Wang, Wenyuan Yan, Yuanyuan Dong, Wenhui Yao, Zhongjie Wu, Lingjun Zhu, Chao Shi, Yinhu Wang, Rong Liu, Junping Wu, Jiaji Zhu, and Jiesheng Wu. 2024. What's the Story in EBS Glory: Evolutions and Lessons in Building Cloud Block Store. In *22nd USENIX Conference on File and Storage Technologies, FAST 2024, Santa Clara, CA, USA, February 27-29, 2024*, Xiaosong Ma and Youjip Won (Eds.). USENIX Association, 277–291. https://www.usenix.org/conference/fast24/presentation/zhang-weidong

[54] Xiaohui Zhang, Ming Cong, and Guangqiang Chen. 2011. Software and Hardware Co-designed Multi-level TLBs for Chip Multiprocessors. In *11th IEEE International Conference on Computer and Information Technology, CIT 2011, Pafos, Cyprus, 31 August-2 September 2011*. IEEE Computer Society, 609–614. https://doi.org/10.1109/CIT.2011.17

[55] Xiantao Zhang, Xiao Zheng, Zhi Wang, Hang Yang, Yibin Shen, and Xin Long. 2020. High-density Multi-tenant Bare-metal Cloud. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems* (Lausanne, Switzerland) *(ASPLOS '20)*. Association for Computing Machinery, New York, NY, USA, 483–495. https://doi.org/10.1145/3373376.3378507