Aggregate VM: Why Reduce or Evict VM's Resources When You Can Borrow Them From Other Nodes?

Abstract

Hardware resource fragmentation is a common issue in data centers, and it is due to many factors. Traditional solutions based on migration or overcommitment are slow, and modern commercial or research solutions like Spot VM, may reduce or evict VM's resources anytime. We propose an alternative solution that solves both problems, the Aggregate VM. We introduce a new distributed hypervisor design, the resource-borrowing hypervisor, which creates Aggregate VMs: distributed VMs that temporarily aggregate fragmented resources belonging to different host machines, which require mobility of virtual CPU, memory and IO devices. We implement a prototype, FragVisor, that runs fully transparent guest software. We also propose slight modifications to the guest OS, which can enable additional significant performance gains. We evaluate FragVisor over a set of microbenchmarks and IaaS-style real applications. Although Aggregate VMs are not a perfect fit for every type of applications, some workloads enjoy significant speedups compared to overcommitted scenarios (up to 3.9x with 4 distributed vCPUs) and that FragVisor is faster than a state-of-the-art competitor, GiantVM (up to 2.5x).

1 Introduction

Hardware resource fragmentation in modern data centers is a critical problem [11, 27, 35, 38, 41, 70, 80, 89]. It is due to several factors: the broad variety of resource requirements for jobs; an increase in the average amount of resources required per job [26, 63]; job placement constraints such as software/hardware dependencies [81, 87]; and poor hardware elasticity/inadequate support for heterogeneity [85]. Even considering optimized scheduling and placement methods [26, 41], or attempts to defragment resources through migration, resource fragmentation still persists [35, 76, 85, 89]. Although disaggregated hardware [33, 42, 57, 58, 85, 89] appears to be a solution in the long term, it is not viable yet, and it is not a panacea: it requires a hardware refresh, a capital investment, and only solves the resource fragmentation problem for memory. Thus, even if cloud providers already attempted to monetize fragmented resources – e.g., with Spot VMs [10, 37, 62], they are still seeking solutions to further increase single-machine resource utilization without affecting the performance of Primary VMs [11, 46, 59, 96], i.e., VMs with guaranteed resources. Although Spot VMs, as well as more recent works [11, 96], do not guarantee fixed performance, but a certain minimum amount of resources, they can be evicted (VM kill) with minimal to no pre-advice. Hence, they are not suitable for Primary VMs' workloads. Moreover, with such solutions the data center scheduler still needs to exactly find a machine with that minimum amount of resources available.

Idea. This paper approaches the problem of data center resource fragmentation in a *fundamentally different way*: instead of exploiting the fractional resources of a single-machine, it aggregates fragmented resources available among multiple machines into a single distributed VM. To achieve that, we introduce a new type of VM, the *Aggregate VM*. Differently from previous works, it guarantees a certain fixed amount of resources at all time without eviction, however the guaranteed SLO is based on the type of workloads. Therefore, this paper is driven by the following questions: a) can Aggregate VMs be a solution to the fragmentation problem and at what overhead? b) for what workloads and at what SLO?

To answer these questions we introduced a new distributed VMM, the *resource-borrowing hypervisor*, which provides *Aggregate VMs* as first class VMs. This enables the traditional unit of resource allocation in the cloud, the Virtual Machine (VM), including processing units (virtual CPUs, vCPUs), (pseudo-)physical memory, and I/O devices, to be distributed over fragmented hardware resources belonging to different physical servers. This distribution is *transparent*, i.e., in its basic form, it requires no modification to guest software; as well as *temporary*, to reflect the dynamic nature of fragmentation, and therefore virtualized resources are *"mobile"* between servers.

FragVisor. We built FragVisor, a resource-borrowing hypervisor that runs on *existing* data center infrastructures, creating and maintaining Aggregate VMs. Aggregate VMs provide the exact amount of resources requested by a user for a VM, reducing the potential performance degradation versus solutions based on resource overcommitment, and eliminating the possibility of resource eviction of transient VMs (Spot VMs, Preemptible VMs, Harvest VMs, etc). Moreover, it also avoids downtimes due to potentially complex multi-VM migrations. However, as a distributed VM, it presents some unavoidable overheads when accessing remote resources. We demonstrate that these slowdowns are much lower compared to solutions based on resource overcommitment, focusing on the CPU – a resource for which fragmentation has been shown to be particularly problematic for VM allocation [11].

FragVisor is a distributed multi-hypervisor [74]. It extends Linux/KVM to run among multiple machines and leverages a Distributed Shared Memory (DSM) system to transparently present to the guest a unified and consistent view of its physical address space. FragVisor allows a vCPU on one physical machine to access memory as well as remote devices, such as virtual disks, on other machines. FragVisor is able to execute fully unmodified guest OSes, although some non-intrusive modifications to the guest OSes can bring performance improvements. More importantly, FragVisor does not require any modification to legacy user-space software, which instead needs rewriting to run over multiple machines.

Innovation. To our knowledge, this is the first work that considers the potentials of leveraging a distributed VM, the Aggregate VM, to solve the resource fragmentation problem in the data center. Although several distributed hypervisors have been proposed in the past [7, 24, 83, 90, 97], we found that those target a fundamental different working model: running scale-up workload on scale-out hardware, instead of (small to medium) VMs demanding at most what a physical today's mid-size server can offer. Hence, those systems lack mechanisms enabling VM resources mobility among nodes.

Similarly to such existing works, FragVisor relies on DSM to provide a (temporary) distributed VM with a coherent (pseudo-)physical memory view over several nodes. Historically, DSM has been known to scale poorly in workloads exhibiting medium to high levels of memory sharing [12, 23, 55]. Recent attempts at reviving DSM, even when they rely on modern high-speed interconnects, still target workloads with relatively low levels of sharing [51, 67, 97]. We observe that in data center IaaS settings, several classical workloads are made up of concurrent components executing in the same VM and exhibit very low degrees of sharing, e.g., web server/language runtime/database stacks, severless computing, etc. Our intuition is that for such workloads, an Aggregate VM can provide a similar SLO than a Primary VM.

To address the dynamic nature of fragmentation in the data center, along with memory mobility (using DSM), an Aggregate VM needs mobility of virtualized CPUs and I/O devices. A cross-node vCPU migration mechanism enables part-of or an entire VM to transparently move at runtime where hardware resources become available, *something not possible with existing distributed hypervisors.* This allows us to consolidate a VM over time on as few servers as possible – ideally one. We also exploit vCPU mobility and a distributed checkpoint/restart mechanism to tackle fault resilience, exacerbated by running a single VM on multiple machines. Finally, we propose a set of new techniques to support I/O device mobility, including single- and multiple-queue I/O delegation [82] on top of DSM, *DSM-bypass* I/O delegation, and distributed I/O.

Key Prototype Results. We prototyped FragVisor and evaluated its performance on a computer cluster, over a set of micro- and macro-benchmarks. Among others, the results show that with four distributed vCPUs, an Aggregate VM offers significant speedups for compute-bound (up to 3.9x) and networking (up to 3.6x) applications, when compared to overcommitment. FragVisor is also faster than a state-of-the-art competitor, GiantVM [97]: up to 2.5x for compute workloads, and 1.3x for network applications. When running shared-memory multithreaded applications on top of an Aggregate VM, the SLO is impacted based on the degree of sharing. FragVisor's slowdown is generally acceptable (15%) although it is not a panacea for workloads relying heavily on shared memory, which may experience higher overheads.

Contribution. The paper makes the following contributions:

- We identify IaaS workloads that are less affected by running on a distributed VM. For those, the SLO of Aggregate VMs is similar to the one of Primary VMs;
- We propose the *Aggregate VM*, a new approach to solve fragmentation in the data center, avoiding resource evictions. An *Aggregate VM* leverages idle hardware resources belonging to different nodes to reduce fragmentation;
- We introduce the *resource-borrowing hypervisor* design, a new distributed hypervisor that provides Aggregate VMs;
- We build FragVisor, an implementation of the resource-borrowing hypervisor based on Linux/KVM , providing the mobility of virtual CPU, memory, and devices, while introducing new distributed hypervisor mechanisms, and guest kernel optimizations;
- We evaluate FragVisor, demonstrating its superior performance design over a baseline and a competitor.

Section 2 further motivates the work, while characterizing several IaaS workloads. Background is revised in Section 3. We present our architecture in Section 4, while Section 5 and Section 6 describe FragVisor's design and implementation, respectively. FragVisor is evaluated in Section 7, and contrasted with past works in Section 8. Section 9 concludes.

2 Motivation

Fragmentation in the Data Center. Researchers from Microsoft recently stated that *"Given Azure's scale, even 1% in fragmentation reduction can lead to cost savings in the order of $100M per year"* [41]. A study [38] analyzing traces on Facebook and Bing clusters noted that fragmentation and overprovisioning are responsible for a 45% increase in job makespan. The authors of a recent study [70], observing a cluster from the Eolas [32] cloud provider, noted an average of 17% of the physical resources wasted each day due to fragmentation. As a result, data center operators are still looking for solutions to reduce resource fragmentation [11, 46, 59, 96], and increase their returns on the data center investments. At the same time, commercial offers aiming at monetizing fragmentation exist from major hyperscalers [10, 37, 62], namely transient VMs (like Spot or Preemptible VMs), but those generate a reduced revenue compared to Primary VMs. This is because they cannot guarantee a certain constant level of provisioned resources, if not a minimal one – lesser SLO, and can be killed.

As mentioned above, fragmentation is due to many factors [11, 27, 35, 38, 41, 70, 80, 81, 85, 87, 89]. An ideal VM placement algorithm that could solve fragmentation is known to be a hard problem [35, 85, 89]. The same is true for attempting to "defragment" the data center by migrating running jobs on a smaller subset of physical machines [89]: migration has the additional downsides of requiring further data center resources (pre-/post-copy [60, 76]) or involving unacceptable downtimes (checkpoint/restart). Thus, in practice, when faced with the problem of accommodating more jobs on a saturated but fragmented set of hosts, the provider can either buy new

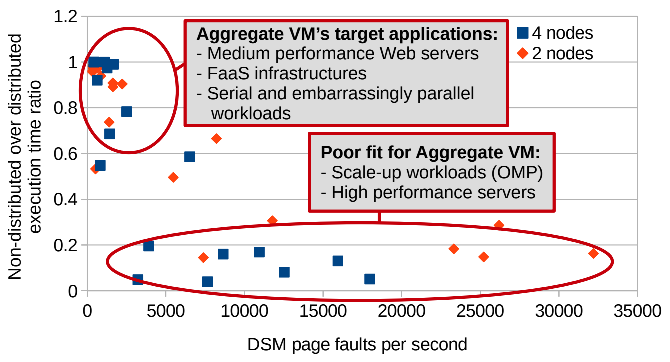

Figure 1. Single-machine (non-distributed) over DSM (distributed) execution time ratios as a function of the number of DSM page fault per second for various applications. An execution time ratio lower than 1 is a DSM slowdown.

machines or overcommit resources by packing more jobs on already busy hosts. Both solutions are suboptimal: buying new machines involves additional financial costs at a time when data center capital equipment costs are higher than operating ones [21], and overcommitment may lead to unacceptable performance degradation.

Early Study: DSM, Sharing, and Scalability. An Aggregate VM relies on DSM to implement virtualized memory mobility, and to provide the guest with a coherent view of a pseudo-physical address space. DSM is known to be slow when data sharing is high [12, 23, 51, 55]. Therefore, we investigated the degree of data sharing among vCPUs when running several IaaS workloads – with the goal of finding workloads that would be minimally impacted when running in a distributed VM sitting on multiple physical nodes.

To that aim, we run on an early prototype of our FragVisor a set of applications including serial (NAS Parallel Benchmarks [65]) and scale-up multithreaded workloads (NPB OpenMP – OMP), a LEMP stack (Nginx/PHP/MySQL) in which the time to generate an HTML page to answer a request varies between 25 and 500ms, benchmarked with ApacheBench, and an instance of the OpenLambda FaaS computing framework [43]. We run the DSM on 2 and 4 nodes, and in each case we set the followings equal to the number of nodes: the number of serial NPB instances, the number of OpenMP threads, the number of PHP workers, and the number of OpenLambda workers. We measure the slowdown of running on top of DSM vs. vanilla Linux as a function of the page faults per second due to DSM.

Results are synthesized in Figure 1. The slowdown increases with the level of sharing, or DSM contention – i.e., DSM faults per second. Applications showing low levels of sharing perform similarly on DSM and on a single-machine (from no slowdown up to 45%): unsurprisingly, serial NPB but also embarrassingly parallel OMP workloads fall within that category, as well as the FaaS infrastructure. Interestingly, LEMP stacks with a page generation latency superior to 40ms also exhibit a modest slowdown, (30% to no slowdown) when running on DSM vs on single-machine. On the other hand, applications with high degrees of sharing, such as most OMP benchmarks and high performance LEMP (page generation < 40ms) suffer significantly from DSM execution (up to 95% slowdown).

Thus, *certain IaaS workloads' performance will not be penalized when running atop an Aggregate VM – they will have a similar SLO to Primary VMs.* Hence, herein we mainly target such a subset of IaaS workloads, characterized by limited sharing among threads – an Aggregate VM is not a general solution, the SLO depends on the software running in the VM. Yet, we observe that the amount of cloud/IaaS applications presenting characteristics suitable for distributed execution is *non-negligible*: for example, regarding LEMP, according to W3Tech [93], WordPress (a LEMP stack) runs today more than 40% of the Internet's websites. Regarding serverless computing, expert estimate that its usage will skyrocket [47] in the next years.

3 Background

Virtualization Technologies. Two virtualization architectures are widespread: *Type-1*, in which the hypervisor directly runs on the hardware, e.g., Xen [18], and *Type-2*, where the hypervisor coexists with or is a OS service, e.g. KVM [53]. With the latter, virtualized CPUs run as host OS threads, and their physical address space (Guest Physical Addresses, GPAs) corresponds to the host OS threads' address space (Host Virtual Addresses, HVAs). In order to let the guest to address its own virtual memory (Guest Virtual Addresses, GVAs), a virtual memory management unit (vMMU) is needed. While the vMMU was implemented using shadow page tables in software [8, 77], today's CPUs provide hardware vMMU – such as Extended Page Tables (EPT) on Intel [66], which provide two levels of hardware MMU translations. A page fault in the guest OS can be handled by the guest MMU, but if there is no host page backing it, the fault will be handled by the host MMU.

Regarding CPU virtualization, manufacturers introduced a new execution mode for the guest VM itself. Each manufacturer has an instruction to switch execution mode, and a data structure has to be filled before switching (e.g., VMX in Intel) – which lists events that make the CPU exit virtual machine mode execution. We refer to this as "full virtualization", while virtualization without any hardware support that requires modification of the guest software is "paravirtualization".

Virtualizing Devices. With the introduction of CPU and memory hardware–assisted virtualization, manufacturers also added hardware virtualization features to devices [29]. Yet, devices with hardware virtualization features supports only a limited number of VMs, or are expensive. In these cases, paravirtualized hardware devices are preferred.

VirtIO [82] is the de-facto standard paravirtualized device technology, and implements different classes of devices, including network interface cards (*virtio-net*), storage devices (*virtio-blk*), and consoles (*virtio-console*). A *VirtIO* device appears to the guest software as a PCIe device and requires the

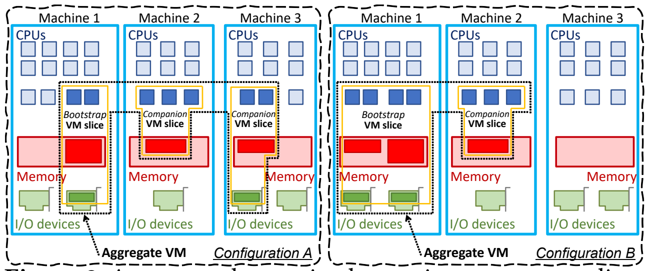

Figure 2. A resource-borrowing hypervisor aggregates slices (yellow boxes) of physical resources from several physical nodes into a distributed VM (dotted black box), which dynamically adjusts its slices based on resource availability (cf. A vs B).

software to instantiate at least a couple of in-memory ring buffers – a transmission (TX) and a receiver (RX) ring . *vHost* is an evolution of *VirtIO* targeting *Type-2* virtualization, it moves a device emulation mechanisms from user- to kernel-space to avoid the user-kernel switches, boosting performance [75].

VM Migration and Distributed Execution. VMs offer a convenient abstraction for software mobility. VM state can be extracted and moved to another physical machine by VM migration [25]. Migration can be used to consolidate multiple VMs on a single server plenty of hardware resources.

Sometimes, instead of assigning the resources of a single server to multiple VMs, it is interesting to aggregate resources of multiple servers into a single, larger, VM. This has been done before (see Section 8) mainly introducing distributed protocols to aggregate hardware resources available on different physical machines. Similarly, the Popcorn Linux project [15–17, 49, 73, 84], and Kerrighed [50], aggregates hardware resources at the OS level versus at the hypervisor level.

4 Aggregate VM Design and Architecture

With the goals of a) fast VM provisioning (faster than delayed execution), b) minimal VM overheads (lower than resource overcommitment and VM migration, aiming at SLOs similar to Primary VMs), c) guaranteed resources provisioning (no reduced resources, nor evictions, like transient VMs), and d) data center–wide resource exploitation, we propose a new distributed hypervisor design that *provisionally* aggregates "slices" of fragmented hardware resources belonging to multiple physical machines into a single (distributed) VM – the *resource-borrowing hypervisor*.

Design Principles. The resource-borrowing hypervisor is based on the following design principles: a) aggregate at least the hardware resources that are needed for a VM to run at each instant; b) reduce over-provisioning; c) have minimal overhead and interference with other VMs or applications; d) be high-performance; and e) be (transparently) compatible with existing software, i.e., guest OSes and applications.

Operational Principles. A resource-borrowing hypervisor creates VMs aggregating hardware resources belonging to different server machines. Hence, as depicted in Figure 2, such *Aggregate VM* may exploit physical CPUs, RAM, and I/O devices owned by different servers (M1, M2, M3 in Figure), and it is *dynamically reconfigurable* to leverage resources on different nodes for the same VM (cf. right and left of Figure).

When a resource-borrowing hypervisor creates an Aggregate VM, a hypervisor instance is started on each of a set of servers. Instances manage subsets of hardware resources from various servers, *VM slices*, which contribute to the whole VM.

One of the hypervisor instances is responsible for establishing the connection among all and is responsible for starting the guest execution – thus, it should provide at least one virtualized CPU. Such an instance will be started with additional information about the other instances (e.g., hosts IP addresses), as well as the disk image(s), and eventually the kernel and bootloader. We call this instance a *bootstrap VM slice*; other instances are called *companion VM slices*. After the bootup phase, all VM slices are peers. A VM slice may include virtual CPUs, virtual RAM, and virtual devices of any type such as IO interfaces, accelerators, etc. However, a VM slice can be composed of just memory (like previous work [30, 39, 71]); or just a device, such as a GPU or TPU (like GPUDirect [54]).

System Architecture. A resource-borrowing hypervisor is a distributed multiple-hypervisor [74]. Each VM slice runs on top of a different hypervisor instance – running on one server. Hypervisor instances communicate between each other using a communication layer, e.g., based on message-passing. The communication layer is exploited by a set of distributed hypervisor services that provide the illusion of a single VM among multiple hypervisors – Aggregate VM. To stick to the design and operational principles, a resource-borrowing hypervisor is based on, and *introduces*, the concept of *mobility of virtualized memory, CPU, and device.* While all the memory, CPUs, and devices of a VM may be distributed and show as a single VM (all slices can access all resources), the "temporary" aspect of borrowing demands for *mobility* – which in turn, demands several old and new mechanisms. VM RAM (vRAM) mobility can be implemented with inter-machine memory copy, or distributed shared memory (DSM). VM CPUs (vCPUs) mobility can be implemented with thread remote creation, or migration. VM devices mobility can be implemented as proxying, delegation, tunnelling, etc. Finally, not all assigned resources to a VM should be allocated for a VM entire lifetime.

System Orchestration. Based on the scale of a data center, what hardware resources among what nodes will be assigned to an Aggregate VM is decided either by the resource-borrowing hypervisor itself, or by an external entity, such as a data center scheduler/orchestrator, e.g. Protean [41]. In the former case, the resource-borrowing hypervisor is predeployed on all machines of the data center, and knows about the resource availability of every machine. Instead, when an external entity governs resource assignments, the resource-borrowing hypervisor is not required to always run on all machines of the data center, but only on the ones with fragmented resources to share. A data center scheduler is an example of external entity, which is anyway monitoring servers' usage, and knows about the managed cluster's already allocated hardware resources and resource requirements of incoming tasks. However, this does require data center schedulers to be *extended* because current schedulers cannot exploit partial/fragmented resources. In this case, the scheduler will inform the resource-borrowing hypervisor to move a VM slice (or part of it) between hypervisor instances (for power, performance, demand of new resources, reliability, etc).

Reliability. Running a VM on top of multiple physical machines is less reliable than on top of a single one. In other words, assuming one machine is 99.9% reliable, two are 99.8%, three are 99.7%, etc. However, it is not just the hardware to guarantee a specific reliability, but also the software [88]. Hence, a single software stack running among several machines, like in an Aggregate VM, may be more reliable than a software stack per machine [91]. From the point of view of the hardware, a resource-borrowing hypervisor cannot change hardware's reliability, but it can exploit state-of-the-art hardware monitoring and logging subsystems (e.g., Intel MCA/AER) to preemptively force migrate a VM slice from a likely-to-fail server to another. Other fault-tolerance techniques, such as periodic checkpointing and restart on failure, can be used instead.

5 FragVisor

FragVisor is an implementation of the resource-borrowing hypervisor targeting a *Type-2* full virtualization architecture for traditional monolithic UNIX-like OSes. It is designed around common state-of-the-art data center hardware, where servers, i.e., compute nodes with multicore CPUs and accelerators, are mainly interconnected via high-speed network(s). Therefore, FragVisor's communication layer is based on message-passing, which in order to reduce the user-kernel switches and improve performance is located in the host kernel, similarly to bespoken multiple-kernel OSes [15, 20]. Hypervisor services run in a distributed fashion atop the communication layer, strictly in *kernel space*, for performance reasons. Different services may require different consistency level to maintain their distributed state and provide mobility.

In the data center, FragVisor does not make any decision itself about what machines will be used by an Aggregate VM – i.e., FragVisor has no placement-like capability. This should be the role of a data center scheduler/orchestrator, such as Protean [41]. We suggest extending, but not changing, existing schedulers. We propose a prototype orchestrator and a policy in Section 6.5.

5.1 Distributed Pseudo-Physical Memory

To provide *virtualized memory mobility* as well as the illusion of *shared memory* (single system) among machines, FragVisor moves memory blocks between different machines.

Modern *Type-2* hypervisors hold the guest pseudo-physical address space (virtualized memory) as a subset of the virtual address space of a host user-space application (the VMM). To make such part of the address space available, and consistent for the guest OS, among different machines, as well as mobile, FragVisor adopts DSM. Parts of the address space that do not belong to the guest pseudo-physical memory area, but to emulated devices, are handled outside the DSM protocol (see Sect. 5.3). Software DSM has been criticized in the past, mainly due to its consistency overheads and poor scalability. Indeed, we already disclosed that because of DSM an Aggregate VM may not provide the same SLO as a Primary VM, but there exist certain workloads that are less or not impacted by DSM. Also, faster networks are increasingly available in data centers [45], speeding up DSM, motivating it even further. Anyway, to alleviate criticism, FragVisor introduces several DSM optimizations, including a contextual DSM protocol, and run-time NUMA topology updates – see below.

Contextual DSM. The hypervisor knows a lot about the content of the guest physical address space, especially about CPU-dependent memory areas, including the position of the page table, interrupt table, etc. In SMP OSes, these are shared among all CPUs, and are highly used. Therefore, it is of the utmost importance to avoid the DSM protocol from slowing down their access. We propose a contextual DSM protocol that leverages information about the memory content to reduce DSM traffic.

NUMA Topology Updates. Modern OSes use NUMA information to optimize operations such as scheduling and memory allocation. With the goal of reducing DSM traffic, FragVisor informs the guest software about the non-uniform access latencies by exposing a NUMA topology that reflects the placement of hardware resources on different server machines. Such NUMA topology is updated at runtime.

5.2 Distributed vCPU

To enable *virtualized CPU mobility*, a FragVisor Aggregate VM runs vCPUs as distributed threads over the guest pseudo-physical memory, kept consistent among vCPUs using DSM. Each vCPU has its own set of registers and include a local interrupt controller and timer (e.g., x86's local APIC timer). There is usually no shared state among different vCPUs if not for processor-wide registers, such as some MSR registers in x86. These are kept consistent among hypervisor's instances.

CPUs notify each other via IPI, including MSI. Thus, each hypervisor instance keeps track of the machine where each vCPU is. IPIs are turned into messages to hypervisor instances.

Finally, live slice migration, necessary for consolidation or fault-tolerance purposes, requires thread migration [15] to move a vCPU between different servers while running.

Distributed vPIC. Other than the CPU local interrupt controller, one or more non-local interrupt controllers may exist in a VM (e.g., x86' IO-APIC). Since a non-local interrupt controller is usually an interrupt broker, and interrupts are converted into messages, this may be kept as non-replicated on the machine with the highest number of physical devices.

5.3 Delegated Virtual Devices

In FragVisor, virtual device access is mainly based on the concept of delegation, i.e., guest VM software running on a VM slice should be able to access any device exposed by the Aggregate VM, but the actual communication with the physical device happens only on the hypervisor instance running on the same physical server as the device. This guarantees mobility.

Devices may communicate with the CPU either via memory-mapped IO or IO ports. We focused on the former. Specifically, we support PCIe-based devices by establishing a distributed PCIe root complex and devices emulator. Many PCIe devices, including the virtual ones we support, instantiate ring buffers in vRAM. Should these buffers be managed by the DSM, the high amount of contention it creates would significantly hurt performance. Thus, in FragVisor devices are using per-CPU TX/RX queues when possible, and we introduce *DSM-bypass*.

Multiqueue VirtIO. To contain our engineering efforts, we focus on paravirtualized hardware (*VirtIO*-based), but our design can be trivially extended. We explain how FragVisor supports virtio-net, but the same applies to virtio-blk, etc.

A paravirtualized network device exposes TX/RX ring buffer pairs that the guest uses to enqueue/dequeue packets. Because these ring buffers are on the VM virtual RAM, the DSM protocol maintains them consistent across hypervisor instances. To reduce DSM traffic, vCPUs running on different physical nodes should avoid accessing the same TX/RX pairs. Hence, FragVisor adopts multiqueue [94] technology, where each TX/RX pair, is mapped to a different vCPU (supported by most OSes) and hypervisor instance.

This solution allows VM slices that do not own a network device to delegate the transmission of network packets to other VM slices by simply writing a packet on DSM and sending an interrupt to notify a new packet has been written.

DSM-bypass. However, adopting multiqueue is not enough: the synchronization operations made of both ends of a communication channel generate a high DSM overhead. Thus, as an optimization we bypass the DSM for TX/RX pairs. We make the paravitualized network device on each VM slice writing and reading the related TX/RX pairs, and send or receive the information to or from the VM slice with the physical network card. Thus, for a TX event the paravirtualized network device piggybacks the network packet(s), read from the memory, to the TX interrupt sent to the VM slice with the physical network card. Thus, the DSM is excluded from the data path.

6 Implementation

We implemented a prototype of FragVisor based on Linux kernel 4.4.137, on x86. To avoid reinventing the wheel, we based our implementation on different Linux kernel components from the Popcorn Linux project [15], including thread and process migration, kernel-level DSM, and the messaging layer. Our target *Type-2* hypervisor is Linux/KVM for Intel x86-64 platforms, and we extend *kvmtool* (commit c57e001) as the user-space tool for creating and managing guest VMs.

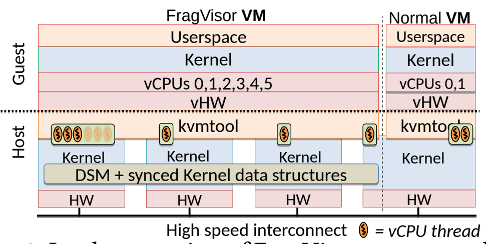

Figure 3. Implementation of FragVisor vs a normal VM.

Our implementation contributes a total of about 13,125 LoC in the OS kernel and 6,050 LoC in user-level tools[1]. Figure 3 compares our prototype to a classic hypervisor.

All our distributed/delegated mechanisms rely on FragVisor's communication layer, that exploits Remote Direct Memory Access (RDMA) over high-speed InfiniBand to minimize inter-server messaging overheads. This required an almost-complete rewrite of the Popcorn Linux messaging layer.

6.1 Distributed VM Memory

The kernel-level DSM of Popcorn Linux has been rewritten to fully support the IVY's invalidation-based DSM protocol [56], contextual DSM, and NUMA topology updates.

The DSM handles the memory consistency of the entire *kvmtool* including guest's virtual RAM (pseudo-physical memory). Thus, unlike traditional DSMs that handle host page table faults only, our DSM also handles EPT faults. Also, traditional DSMs only invalidate the host page table by making the page table entry (PTE) non-present and flushing the corresponding TLB. However, with a second level of translation, our DSM needs to invalidate EPT's SPTEs and flush the secondary TLB.

A guest OS page fault requires a GVA to GPA translation. If an entry exists in the guest's page table, the guest will then access the GPA. However, accessing a GPA whose page is not present on a machine causes a host page fault. If a GPA to HPA mapping exists in the EPT – maintained by the hypervisor, the VM directly accesses the host physical address (HPA) without causing a VM exit. Otherwise, VM exits – due to EPT violation. This is reported to the DSM that eventually fetches the page, if it exists, from other nodes.

Troubleshooting DSM Traffic. When traditional software for SMP runs on FragVisor, it may show additional overheads. To debug that, our DSM tracks, for each page fault, the guest physical and virtual addresses – then correlated with sources.

Learning from previous works [7, 24, 48, 90, 97], we traced the guest VM executing different applications on Linux. That helped identify several uncorrelated kernel data structures that were placed in the same memory page and creating unnecessary DSM traffic (false sharing). We patched the guest OS kernel to solve the problem, reducing the DSM traffic[2].

[1] *In addition* to the components borrowed from the Popcorn Linux project.
[2] This patch will be released open-source. We believe this is beneficial for this work and related work, and we prove it beneficial for SMP.

Optimizations. By tracing, we identified another source of DSM traffic: hardware dirty bit management. The EPT subsystem can be configured to additionally set dirty bits in its own translation, which generates additional DSM traffic. Hence, we disabled dirty bit tracking – anyway, this is already taken care by the DSM itself, thus redundant in this project.

FragVisor implements the notion of *contextual DSM*. For example, for page tables, in order to reduce the network traffic for TLB shootdown it piggybacks the page table modifications with the shootdown interrupt message – reducing DSM traffic.

FragVisor implements *NUMA topology updates*. Namely, static APIC tables show one NUMA zone per VM slices, when VM slices move, coalesce or split, we trigger an ACPI System Resource Affinity Update notification [92] to reflect changes.

6.2 Migrating vCPUs

Our prototype either lets the *Bootstrap* VM slice create all vCPU threads and then migrates those vCPU threads to *Companion* VM slices, or creates remote vCPUs threads (at boot time only). This allows us to reuse part of the existent Popcorn task migration to distribute vCPUs among multiple machines. To provide distributed vCPUs we augment task migration to account for additional vCPU-related state and metadata as described before – most doesn't need to be kept consistent among machines. Finally, the prototype maintains for each VM slice a replicated array that tracks the position of every vCPU and is updated at each migration event. This is fundamental to implement mobility and inter-server messaging.

Interrupts. vCPU threads are the recipients of interrupts. In the prototype, we modified the hypervisor for interrupt dispatching in order to check if the target vCPU is local or remote. If it is remote, a message with the description of the interrupt event is sent to another VM slice instance – via the communication layer. Otherwise, the traditional code path is followed.

6.3 Distributed VM Devices

We rewrote most *virtio*-based and emulated *kvmtool* devices (all but 9p, balloon, VESA) in order to work atop the communication layer. Thus, a device physically located within a specific VM slice can be used by all VM slices. For performance reasons we targeted the kernel implementation of *virtio*, which is *vhost*. However, because *kvmtool* doesn't support multiqueue with *vhost*, we had to write a patch to extend it[3]. As *kvmtool* doesn't support virtio-GPU, we cannot showcase the feature of borrowing an accelerator. However, this is just a technical limitation, advantages of such technology has been already commercially proved, e.g., NVIDIA GPUDirect [72].

Network. The prototype leverages Linux's vhost-net. While our distributed PCIe layer takes care of the physical PCIe address ranges, the DSM replicates the TX/RX pairs on physical memory, and the multiqueue reduces DSM traffic. With DSM-bypass TX/RX pairs are not replicated. We also need to properly handle the notifications between the guest and the host in a distributed environment. Thus, each node has to install the corresponding file descriptors (*ioeventfd* and MSI *irqfd*) to notify the guest and host to access TX/RX pairs.

When the guest VM sends a network packet, it enqueues the packet's information on a TX vring, and passes its index to the hypervisor. FragVisor will deliver the event to the VM slice with the physical device that will fetch the packet using DSM. With DSM bypass, the packet is sent by the hypervisor to its destination through the communication layer.

Similarly, when a packet for the guest VM is received, vhost-net copies it into the guest memory, and injects an IRQ based on the TX/RX pair number. The VM slice managing that IRQ will move in the packet using DSM. With DSM bypass, the packet is sent to the VM slice managing that IRQ, which copies the data into guest memory.

Storage. The current prototype is capable of using virtual RAM as a backing storage (tmpfs), or vhost-blk. tmpfs exploits DSM for consistency, while the vhost-blk works like the vhost-net implementation, including multiqueue, and DSM bypass.

Serial Console. Remote nodes can print host user-space KVM logs. To achieve this, each thread migrated to a remote node has to reopen a PTY. Furthermore, the system has only one pseudo-terminal worker thread emulating a serial UART chip.

6.4 Distributed Checkpoint/Restart

In an effort to offer some form of fault tolerance in FragVisor, we implemented distributed VM Checkpoint/Restart (C/R). We first enabled traditional C/R system in *kvmtool* that originally did not support it. We then integrate C/R with FragVisor. FragVisor accepts C/R requests via UNIX sockets. Once a request is received, our *kvmtool* stops all running vCPUs by broadcasting signals to all vCPU threads, then it saves virtualized memory, CPU, and devices state to files. Another signal is sent to all the vCPU threads to continue VM execution. vCPU states are transferred to the node requesting the checkpoint.

6.5 Scheduler/Orchestrator Extension

To demonstrate the viability of FragVisor, we implemented a basic homogeneous-cluster best-fit FIFO VM-scheduler inspired by previous works [26, 41, 44, 52]. It tracks free resources of each machine in a cluster, while machines update it for VM terminations and current load. We then extended such BFS scheduler to handle Aggregate VMs in this way: a) when BFS fails to allocate a VM, we search for the minimum or maximum (2 different policies) amount of nodes with fragmented resources to satisfy the allocation and start FragVisor on such nodes, if not enough resources are available the scheduler delays the VM placement; b) on termination of any VM co-located with at least an Aggregate VM's slice, we evaluate if the released resources can be allocated to one (or more) of the VM slices – one policies favors minimizing the overall fragmentation while the other minimizes the number of nodes on which an Aggregate VM runs, and eventually triggers a FragVisor migration; c) when all resources of an Aggregate VM reside on a node, the VM is given back to the BFS scheduler.

[3]This patch will be submitted to the kernel mailing list upon publication.

We believe this can be integrated in a production scheduler because open-source ones [52] enable policy extensions, but orchestrators may need rewriting to trigger migrations.

7 Evaluation

In Section 2 we already demonstrated that the SLO of software running into an Aggregate VM depends on the type of workload, specifically on the level of sharing – due to the dependency on DSM. Herein, we will further highlight that, while answering (1) if an Aggregate VM can be used to solve fragmentation (at least compared with overcommitment); (2) what are the eventual overheads (to motivate our solution vs delayed-allocation, migration, etc.); (3) how FragVisor, and its mechanisms, quantitatively differentiate from previous work.

Testbed. To evaluate FragVisor, we created Aggregate VMs on a computer cluster composed of multiple servers equipped with a Xeon E5-2620 v4 (2.1 GHz) and 32 GB of RAM. Servers run Linux 4.4 (necessary for Popcorn Linux) as the host kernel, and are connected via InfiniBand using Mellanox Connect-X4 adapters (56 Gbps). VMs are created with enough RAM to satisfy the various workloads they execute and use a ramdisk as the root filesystem, if not indicated differently. Unless stated otherwise, FragVisor VMs use our optimized version of Linux 4.4 as the guest kernel. In most experiments we vary the number of vCPUs per VM and their distribution among hosts. For all experiments, vCPUs are pinned on pCPUs.

Tests. We use microbenchmarks to evaluate (a) the cost of accessing remote memory and I/O, i.e., memory and device mobility (DSM-backed and DSM-bypassed network delegation); as well as (b) the overhead of our distributed checkpointing mechanism. We evaluate (c) the overall performance of Aggregate VMs running on FragVisor executing real-world compute-/memory-intensive (NAS Parallel Benchmarks [65], PARSEC [78]) and I/O-intensive (LEMP server, serverless framework) benchmarks representative of modern data center workloads. For such real-world workloads, we compare when possible the performance of an Aggregate VM on FragVisor vs. a distributed VM on GiantVM [97] – the state-of-the-art open-source distributed hypervisor, and a single-machine (non-distributed) VM – using *kvmtool*. Finally, we demonstrate the benefits of (d) our version of Linux specialized for distributed execution atop of FragVisor, as well as (e) our vCPU migration mechanism – CPU mobility, triggered by a cluster scheduler.

Several papers [31, 40] already show the benefits of memory borrowing. Hence, we herein omit such evaluation, and focus on CPU borrowing, as CPUs are the scarcest resource [11].

Test Measurements. We mainly report execution time of experiments, and their ratios. At the same time, we recorded the CPU, memory, and IO usage of the physical machines (will be released as part of a TR). We noticed that differently from GiantVM, FragVisor do not consume any additional machine CPU resources other than the pCPUs on which vCPUs are running – *this is because of GiantVM's helper threads of QEMU.* Hence, FragVisor does not add any interference to other pCPUs [59] potentially running Primary VMs – *not possible for GiantVM without affecting the performance of other VMs, or reducing the numbers of VMs on a server.* We report the best numbers for GiantVM, either with helper threads co-placed on the same pCPUs of the vCPUs, or on additional pCPUs.

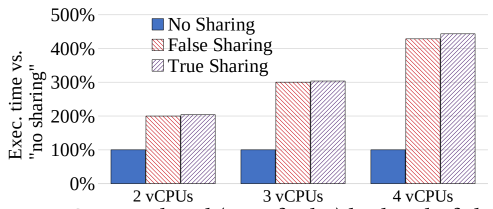

Figure 4. DSM overhead (EPT faults) by level of sharing.

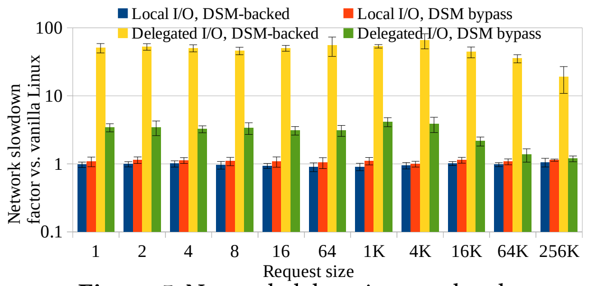

Figure 5. Network delegation overhead.

7.1 Microbenchmarks

DSM Fault Traffic. The DSM overhead in FragVisor takes the form of relatively costly EPT faults due to its consistency protocol. We designed a microbenchmark to understand that overhead. We created a program where each thread reads and writes in a loop at a configurable memory location in an array. By tuning this location and running a single instance of the program among several vCPUs, each on a different server/VM slice, we create 3 test scenarios: (1) true sharing – all threads access the same location, (2) false sharing – all threads access a different location but on the same page, (3) no sharing – threads access locations in different pages. We ran the experiment on Aggregate VMs with 2 to 4 vCPUs – the most common sizes [41], and measured the loop execution time.

Results are shown in Figure 4, with the loop execution time on the Y-axis normalized to the "no sharing" scenario. When remote memory is accessed, the execution time increases linearly with the number of nodes involved, i.e., it doubles for 2 nodes, triples for 3, etc. As expected, false and true sharing cases result in the same behavior. This gives a performance upper bound when running an application with a lot of sharing. (All DSM micro-benchmarks will be released as TR.)

I/O Delegation Overhead. Another overhead of FragVisor is network I/O delegation. To evaluate that, we compare the network throughput of a NGINX web server where the worker runs (1) on a vCPU that is local to the host's virtual switch used to communicate with the client (local I/O) and (2) on a vCPU that is on a remote node (delegated I/O). The client runs ApacheBench [3] on a node on the same 1Gb network

– simulating a request from outside the datacenter. It sends 1000 requests with 10 concurrent connections to the web server. We vary the served web page/object size and get request throughput. We present numbers for both delegation mechanisms, i.e., DSM-backed and DSM-bypass.

Results are in Figure 5. On the Y-axis the measured throughput is normalized to that of a vanilla Linux VM. DSM-backed delegated I/O is slow, due to a high DSM contention on the ring buffers used for delegation: the slowdown is on average 53x for request sizes below 4 kB. Increasing the request size helps to amortize the slowdown, however it is still of 19x for a size of 256 kB. Bypassing the DSM leads to a significant performance boost: the slowdown is only of 3.5x for sizes below 4 kB, and becomes negligible when the size increases. The throughput of FragVisor when performing local I/O is also the same as Linux's. These results demonstrate the efficiency of bypassing the DSM when delegating I/O in a distributed VM.

Similarly, our storage I/O delegation, which has been evaluated with *fio* on 500 MB/s SSD, shows the same behavior.

Checkpointing Performance. To evaluate the overhead brought by our distributed checkpointing mechanism, we created several Aggregate VMs varying the amount of RAM (10, 20 and 30 GB) and the number of vCPUs (2, 3 and 4), each CPU being placed on a physically distinct host. We compared the time to take a checkpoint in FragVisor versus in non-distributed vanilla VMs of similar characteristics. Before taking the checkpoint, to distribute the memory, we run on each vCPU one instance of NPB IS class C (700 MB dataset size). We found the overhead of FragVisor over vanilla is always 10% or less. We observed that the main bottleneck is the disk, a traditional SATA SSD with a 200 MB/s throughput. As a result, accessing remote memory from the checkpointing node in the case of FragVisor does not weight much in the total checkpointing time.

7.2 Real-World Applications

Without an Aggregate VM, a common way to pack more jobs on a saturated but fragmented cluster, with no possibility of VM evictions, is to overcommit resources [96] per-machine. Even Burstable VMs overcommit system resources for limited amount of time. We evaluated FragVisor with real-world applications by comparing the performance of Aggregate VMs on FragVisor versus (1) single-machine VMs with overcommitted vCPUs, and (2) a distributed VM[4] running on top of GiantVM [97]. We varied the number of vCPUs to be 2, 3, and 4 – these are the most common VM sizes [41] in data centers. In the case of the distributed VMs (FragVisor and GiantVM), each vCPU runs on a different host. Indeed, Spot VMs and recent works [11, 96] also exploit idle cluster resources, but they are subjected to evictions. We believe such approaches that guarantee only minimal resources can be fairly approximated to overcommitment when no evictions happen.

[4]An Aggregate VM without "mobility" features.

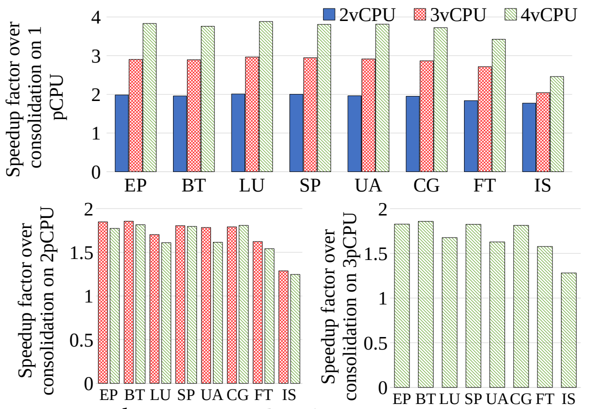

Figure 6. Multi-process NPB, Aggregate VM on FragVisor vs. overcommitting on 1 (top), 2 and 3 (bottom) pCPUs.

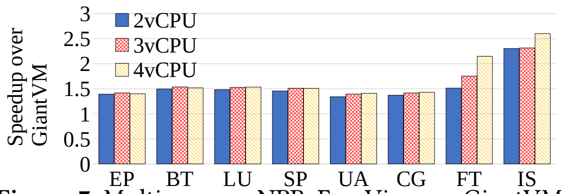

Figure 7. Multi-process NPB: FragVisor vs. GiantVM.

Serial HPC Applications (No Sharing). We ran the NPB suite, selecting for each benchmark a data set size that would result in an execution time of at minimum 10 seconds. For each benchmark and VM type, we run in parallel one instance of the serial version of the benchmark for each vCPU and measure the total execution time of this set of instances.

The results are shown in Figure 6. The y-axis represents FragVisor's Aggregate VM speedup normalized to overcommitting case where a traditional VM's vCPUs are overcommitted on a single, two, and three pCPUs (different sub-graphs). Overcommitting allows fitting additional jobs on a saturated (and potentially fragmented) cluster [96]. Adopting an Aggregate VM brings significant speedups compared to overcommitting: compared to consolidating on 1 pCPU, they range from 1.8x to 3.9x, and most applications see their performance scaling close to linearly with the number of vCPUs. However, such scaling is not as pronounced for IS and, to a lesser extent, FT. We noticed that these applications include a memory allocation phase which execution time is non-negligible compared with the computation phase length. We estimate that their performance behavior is due to DSM contention that results from kernel data structure synchronization during that allocation phase. When comparing with over-committing on 2 and 3 pCPUs, naturally Aggregate VM's speedups are lower, generally being around 1.75x. Note that there is no increase in speedup going from 3 to 4 vCPUs (i.e., adding one instance of the benchmark) when compared to consolidating on 2 pCPUs: this is expected – for relatively small numbers of long-running jobs such as NPB, running 4 instances on 2 compute units yields approx the same execution time as running 3 instances.

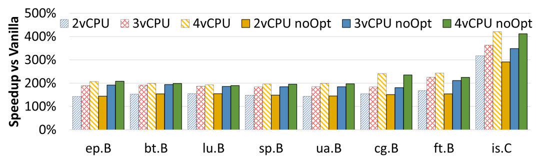

Figure 8. Optimized and not-optimized guest kernel speedup.

GiantVM doesn't implement the mobility features required by an Aggregate VM, but can indeed run an Aggregate VM that doesn't move – i.e., a bare distributed VM. Figure 7 reports the results of comparing FragVisor to GiantVM. FragVisor is faster – on average by 1.6x, for all benchmarks. More precisely, although for most applications FragVisor is about 1.5x faster than GiantVM independently of the number of vCPUs, for IS and FT the performance difference is higher: FragVisor is on average 2x faster than GiantVM for IS, and 1.8x for FT. We found that *the performance difference between FragVisor and GiantVM to be due to several factors.* GiantVM's is implemented partially in user-space (DSM in kernel) while ours lives completely in the host kernel, avoiding costly user/kernel transitions. Furthermore, FragVisor benefits of a guest kernel optimized for distributed execution and exposes a NUMA topology mapping to the physical nodes' distribution.

Optimized Linux Guest. To measure the performance benefits brought by our optimized version of the Linux kernel, we ran that in a FragVisor Aggregate VM and compared it with a non optimized kernel, normalized to overcommitment on one pCPU (vanilla), using the NPB benchmarks.

Results are in Figure 8. Despite the obvious speedups vs overcommitment, the differences between the optimized and non-optimized (noOpt) guest kernels are minimal. In fact, our optimizations became more beneficial (up to 30% in IS) for workloads with a larger amount of sharing and synchronizations – related to DSM traffic, as the recorded time focuses on compute phase. (Full numbers will be available in a TR.)

Multithreaded HPC Applications. We already shown that the SLO provided by an Aggregate VM to scale-up workloads – especially if with a lot of sharing, cannot be the same as a Primary VM because of the underlay DSM. Here we quantify the performance degradation of HPC shared-memory multithreaded applications [19] when running in an Aggregate VM. We created a VM with 4 vCPUs (most common size other than 2 vCPUs [41]) for FragVisor with and without optimized guest kernel and GiantVM, each vCPU being located on a physically distinct node. We launched the multithreaded version of the PARSEC [78] benchmark suite, setting the number of threads to 4. We compare the performance of FragVisor and GiantVM to that of a vanilla non-distributed VM with 4 vCPUs mapped to 1 pCPU (overcommitment).

Figure 9 presents the results. FragVisor can be up to 6.5x and 5.3x (non-optimized and optimized guest kernel) slower than vanilla (Streamcluster), but on average just 2.6x and 2.3x. For a few benchmarks, Blackscholes, Ferret, and Freqmine, distributed execution is up to 4x faster. FragVisor also performs

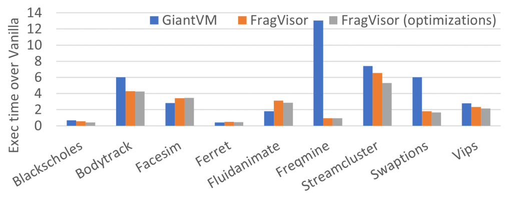

Figure 9. Multithreaded Parsec slowdowns vs. Vanilla.

overall better than GiantVM, whose average slowdown is 4.5x, with a peak of 13.1x. *We collected several statistics to understand the different behavior of GiantVM and FragVisor,* including number of page faults and messages, messages' size, and message's latency. GiantVM sends up to 300x more messages than FragVisor (Freqmine), and message sizes are always small, on average of 2 kB vs. 4 kB for FragVisor. This shows the higher efficiency of FragVisor's distributed protocol.

Network Application: LEMP Stack. LEMP [2] is an open-source web service software consisting of NGINX [5], PHP [6], and MySQL [4]. NGINX acts as a front-end HTTP server receiving requests from clients. PHP is invoked in response to requests, fetching back-end data and performing various processing operations to create web pages. MySQL is a database for storing data for services – this is unused in this experiment.

We run the LEMP stack in an Aggregate VM as follows. First, NGINX is configured with a single worker thread running on the vCPU that is local to the virtual switch used by the VM to access the network (vCPU0), to avoid network delegation overheads demonstrated in Section 7.1. Upon receiving a client request, NGINX invokes PHP which runs a worker thread on each vCPU except the one running the NGINX thread. In other words, the 2 vCPUs configuration has 1 NGINX worker on the first vCPU, and 1 PHP worker on the other. Similarly, the 3 vCPUs configuration has 1 NGINX worker on the first vCPU and 2 PHP workers on the others.

We set the served page size to 2 MB, the average page size on the web according to recent statistics [13]. Each request triggers the execution of a PHP benchmarks that we adapted from [1], realizing some string manipulation operations – a common operation in PHP-enabled web servers. By varying the amount of iterations of this benchmark, we can customize the request processing time and observe its impact on the performance. We vary that processing time from 25 ms to 500 ms, which is representative of modern servers' response times [28], averaging from 200 to 500 ms [68]. ApacheBench (AB) is used as the client, running on the host node that runs the pCPU where the vCPU running the NGINX worker is mapped. AB is configured to make 100 requests with 10 concurrent connections. We collect the reported throughput in requests per second and compare the results of an Aggregate VM on FragVisor, or on GiantVM, to an overcommitment scenario where all vCPUs run on a single pCPU.

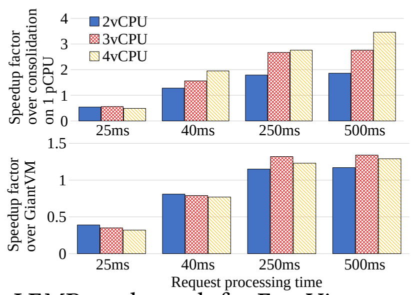

Figure 10. LEMP stack result for FragVisor, normalized to overcommitting (top) and GiantVM (bottom).

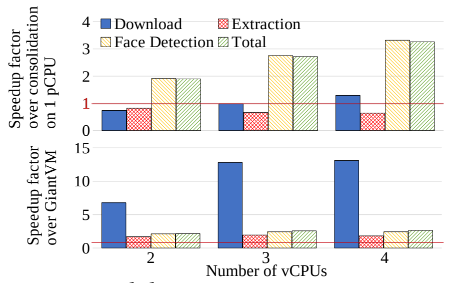

Figure 11. OpenLambda: FragVisor vs. overcommitting (top) and GiantVM (bottom).

The results are presented in Figure 10, where FragVisor's and GiantVM's throughput are normalized to the overcommitment case. With short processing times, our system does not perform well due to expensive communication between NGINX and PHP workers (local socket within the guest), as the communicating processes run on two separate physical machines. Starting from 40 ms of processing per request, the communication overheads become lower compared to the computing time, and are amortized by the benefits of leveraging remote computing resources: the throughput of an Aggregate VM becomes higher than the consolidated case. The speedup increases with both the demand for computation time (processing time) and the number of vCPUs. For example, with 4 vCPUs and a 500 ms processing time, the speedup is of 3.5x.

For fast request processing times, FragVisor is slower than GiantVM: for 25 ms requests, FragVisor's throughput is on average 35% of GiantVM's, and for 40 ms that number is 79%. However, for longer requests, FragVisor's throughput becomes higher than GiantVM's by a factor of 1.23x for 250 ms requests and 1.27x for 500 ms ones. This indicates that, although GiantVM remote vCPU communication is faster, which is important for short requests, for longer ones FragVisor is better at exploiting the parallelism brought by these vCPUs.

Serverless Computing. Function as a Service (FaaS) computing is an emerging paradigm where users upload and execute small pieces of code, or *functions*, in the cloud [9, 36, 61].

We used the OpenLambda [43] FaaS runtime to run a typical serverless computing applications [34], varying its resources. Providers allocate vCPUs to FaaS runtime based on the requested memory size [95]. In an Aggregate VM running on FragVisor, we run the OpenLambda server configured to spawn functions upon reception of an HTTP request on each available vCPU – on different VM slices. Such functions run the same Python code, which (1) fetches a series of pictures as a compressed file from a database on the same network, (2) extracts the pictures, and (3) runs a face detection algorithm [34], thus, returning to the client the number of detected faces.

A client, running on another machine, triggers function execution. We vary the number of parallel requests to be equal to the number of vCPUs: 2, 3 and 4. Other than total time, we break down the server-side execution times into: pictures download, compressed file extraction, face detection.

Results for an Aggregate VM on FragVisor and a distributed VM on Giant VM are shown in Figure 11. They are normalized to overcommitting where 2, 3 and 4 vCPUs are consolidated on the same pCPU. When overcommitting, the speedup for the overall operation increases with the number of vCPUs because the overcommitted VM needs to run more processing on the same computing resources (1 core). Regarding extraction time, even if there are write operations to different pages, the first write to a new region on a remote node always causes DSM write-exclusive invalidation messages, contributing to the slowdown of the extraction as the number of vCPU increases. The face detection phase is considerably sped up (up to 3.3x for 4vCPU) with FragVisor. As it dominates the execution time of the entire operation, the overall performance outperforms the overcommitting cases by 1.9x to 3.26x from 2 to 4 vCPUs. Comparing to GiantVM, FragVisor is always faster on every phase, in particular the download one, up to 13x with 4vCPUs. This lead to a speedup of 2.17x (2vCPU) to 2.64x (4vCPU) for the whole operation. FragVisor proved to be faster not only for its kernel-space DSM, but also for IO device multiqueue and DSM-bypass.

7.3 Scheduling-driven Migration

While the previous experiments demonstrate virtualized memory and devices mobility, here we showcase the cost of vCPU mobility. FragVisor's vCPU migration feature is useful when resources free up on at least one host running part of

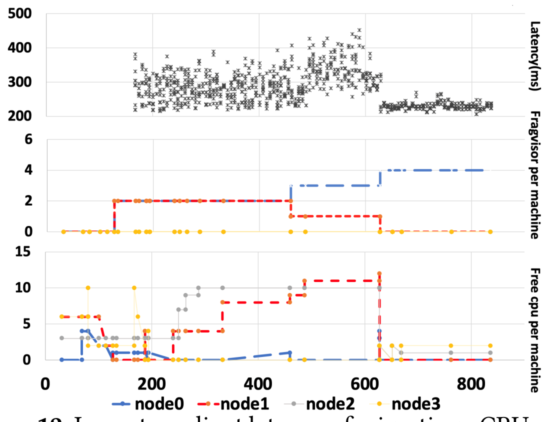

Figure 12. Impact on client latency of migrating vCPUs of an Aggregate VM controlled by a scheduler (top). Node locations of such vCPUs (middle). Free CPUs on nodes (bottom).

an Aggregate VM, allowing that VM to be consolidated on a smaller number of nodes for higher performance. Migration is also important for reliability when a failure can be predicted.

To demonstrate migration we set up a cluster of 4 servers, each with 12 CPUs available for VMs (other 4 left for management tasks). We extract VM sizes and VM execution times (scaled down by 100 to ease experiments) distributions from [41]. We generate bursts of 100 arrivals that we fed into our BFS-extended scheduler, which can start and migrate FragVisor's Aggregate VMs. Aggregate VMs run a web server on vCPU0 and the previously described PHP benchmark on the other vCPUs. A client on the same network sends 1000 requests to the server and measures the latency of each request. While requests are generated the vCPUs' of the Aggregate VM migrate.

We picked a trace of a 4 vCPUs VM, depicted in Figure 12. The top graph reports the perceived client latency, which is the lowest 215ms when vCPUs are consolidated on a single machine, but 299ms on average for this experiment. The middle graph reports how many vCPUs for each node the Aggregate VM is using. The bottom graph reports how many free vCPUs are available per machine, which shows the effectiveness of our simple algorithm that reduces resource fragmentation.

Finally, we recorded the inter-node migration overhead of a vCPU which is 86us on average – 38us to dump registers.

8 Related Work

Resource Fragmentation in the Data Center. Harvest VMs [11] and Elastic VM [96] grow and shrink the amount of hardware resources based on what is/may become available on a single-machine, helping to tackle fragmentation. Differently, an Aggregate VM extends to multiple machines. Similarly to Spot VMs [10, 37, 62], Harvest VMs and Elastic VMs may be terminated at any time, making them practical only for a subset of applications. Instead, an Aggregate VM is never evicted, and always provide the requested resources – like a Primary VM, but we show the SLO depends on the workload.

Earlier, also StopGap [70] aims at solving fragmentation, introducing VMs that can be dynamically resized. It supports only certain types of applications, elastic multi-tier master-slave applications, while FragVisor supports all applications.

Distributed Virtual Machines. Distributed VMs have been proposed before: vNUMA [24], ScaleMP [7], TidalScale [90], and GiantVM [97]. Fundamentally different from FragVisor, such works target scale-up workloads: they aggregate all resources in a computer cluster to execute one or a few VMs with resource requirements greater than what provided by single machine. Conversely, FragVisor aggregates only a *subsets* of hosts' physical resources for the execution of VMs with applications' resources requirements that can be satisfied by a single machine. Moreover, while in related works a VM distribution is mostly static, in FragVisor the distributed state of a VM aims to be temporary for it to be consolidated upon as few hosts as possible once resources free up. Hence, the resource borrowing hypervisor focus on mobility features, e.g., vCPU migration, missing in vNUMA, ScaleMP, and GiantVM (but all provide vRAM migration via DSM). These do not support checkpointing either (for consolidation or fault tolerance). TidalScale provides vCPU migration, but for a different reason. No previous work seems to provide device mobility.

Finally, vNUMA [24] and ScaleMP [7] are *Type 1* hypervisors, while GiantVM [97] and TidalScale [90] are *Type 2* like FragVisor. Differently from [7, 24, 90], FragVisor is free and open-source, built atop the widespread KVM hypervisor. When compared to [97], FragVisor is mainly implemented in kernel-space, and provides Aggregate VMs as a first class VM.

Distributed OSes. Distributed OSes aggregates resources from different physical machines while still providing the same OS interface, and scaling beyond a single physical machine. Notable past works include MOSIX [14], Amoeba [64]. More recently, Helios [69], and Popcorn [15, 17, 73]. Contrary to FragVisor (and GiantVM), these works allow an application, rather than a VM, to use remote resources.

Distributed Shared Memory. Several of the mentioned works leverage DSM in order to give to the application a unified virtual address space. Similarly, FragVisor uses DSM for vRAM mobility and gives the VM a unified pseudo-physical address space. Much work studied DSM during the 1990s [12, 23, 79], with the overall objective of letting traditional/legacy shared memory applications scale over several physical machines. Recent work leverages modern interconnect technologies, i.e., RDMA, in DSM systems [22, 67, 86]. FragVisor introduces several new optimizations to make the nodes involved in DSM changing dynamically (mobility), vs statically defined.

Disaggregated Computing. FragVisor takes an approach different from existing works on hardware disaggregated computing [33, 42, 57, 58, 85, 89]. Indeed, our objective is the aggregation of scattered hardware resources in data centers, rather than the abstraction of future disaggregated hardware. In addition, contrary to these works, we target commodity servers and our solution is directly deployable today, at no capital cost. In the future, FragVisor can complement these approaches.

9 Conclusion

For certain workloads an Aggregate VM can be used to tackle resource fragmentation issues in data centers without reducing promised hardware resources or evicting VMs, i.e., with a SLO similar to Primary VMs. In general, the SLO of an Aggregate VM will depend on its memory-sharing characteristics. We introduced a new design, the resource-borrowing hypervisor, which leverages fragmented hardware resources available among different physical machines, transparently, and temporarily. This temporary nature is achieved through mobility of virtualized memory, CPUs, and devices. We implement FragVisor, based on Linux/KVM and demonstrate its benefits vs. a state-of-the-art distributed hypervisor, GiantVM, and overcommitment. and further demonstrates significant speedups versus overcommittment. FragVisor will be open sourced online.

References

[1] Php benchmark script webpage, 2020. http://www.php-benchmark-script.com/.

[2] Lemp stack resources and q&a, 2021. http://lemp.io/.

[3] Apache HTTP Server Benchmarking Tool, Jan 2020. http://httpd.apache.org/docs/2.4/programs/ab.html.

[4] MySQL Wbsite, Jan 2020. https://www.mysql.com/.

[5] NGINX Website, Jan 2020. https://nginx.org/en/.

[6] PHP Website, Jan 2020. https://www.php.net/.

[7] ScaleMP vSMP Website, Jan 2020. https://www.scalemp.com/.

[8] Keith Adams and Ole Agesen. A comparison of software and hardware techniques for x86 virtualization. *ACM Sigplan Notices*, 41(11):2–13, 2006.

[9] Amazon. AWS Lambda Website, 2020. https://aws.amazon.com/lambda.

[10] Amazon. Ec2 spot instances, 2021. https://aws.amazon.com/ec2/spot-instances/.

[11] Pradeep Ambati, Íñigo Goiri, Felipe Frujeri, Alper Gun, Ke Wang, Brian Dolan, Brian Corell, Sekhar Pasupuleti, Thomas Moscibroda, Sameh Elnikety, et al. Providing slos for resource-harvesting vms in cloud platforms. In *14th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 20)*, pages 735–751, 2020.

[12] Cristiana Amza, Alan L Cox, Sandhya Dwarkadas, Pete Keleher, Honghui Lu, Ramakrishnan Rajamony, Weimin Yu, and Willy Zwaenepoel. Treadmarks: Shared memory computing on networks of workstations. *Computer*, 29(2):18–28, 1996.

[13] HTTP Archive. Report: Page Weight, Jan 2020. https://httparchive.org/reports/page-weight.

[14] A. Barak and O. La'adan. The MOSIX multicomputer operating system for high performance cluster computing. *Journal of Future Generation Computer Systems*, 13:361–372, March 1998.

[15] Antonio Barbalace, Robert Lyerly, Christopher Jelesnianski, Anthony Carno, Ho-Ren Chuang, Vincent Legout, and Binoy Ravindran. Breaking the boundaries in heterogeneous-isa datacenters. In *Proceedings of the 22nd ASPLOS*, Xi'an, China, April 2017.

[16] Antonio Barbalace, Binoy Ravindran, and David Katz. Popcorn: a replicated-kernel os based on linux. Ottawa Linux Symposium, 2014.

[17] Antonio Barbalace, Marina Sadini, Saif Ansary, Christopher Jelesnianski, Akshay Ravichandran, Cagil Kendir, Alastair Murray, and Binoy Ravindran. Popcorn: Bridging the programmability gap in heterogeneous-ISA platforms. In *Proceedings of the 10th EuroSys*, Bordeaux, France, April 2015.

[18] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the art of virtualization. *OSR*, 37(5):164–177, 2003.

[19] Nick Barrow-Williams, Christian Fensch, and Simon Moore. A communication characterisation of splash-2 and parsec. In *2009 IEEE International Symposium on Workload Characterization (IISWC)*, pages 86–97, 2009.

[20] Andrew Baumann, Paul Barham, Pierre-Evariste Dagand, Tim Harris, Rebecca Isaacs, Simon Peter, Timothy Roscoe, Adrian Schüpbach, and Akhilesh Singhania. The multikernel: a new os architecture for scalable multicore systems. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, pages 29–44, 2009.

[21] Ricardo Bianchini. Improving datacenter efficiency. In *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '17, New York, NY, USA, 2017. ACM.

[22] Qingchao Cai, Wentian Guo, Hao Zhang, Divyakant Agrawal, Gang Chen, Beng Chin Ooi, Kian-Lee Tan, Yong Meng Teo, and Sheng Wang. Efficient distributed memory management with rdma and caching. *Proceedings of the VLDB Endowment*, 11(11):1604–1617, 2018.

[23] John B Carter, John K Bennett, and Willy Zwaenepoel. Implementation and performance of munin. *ACM SIGOPS Operating Systems Review*, 25(5):152–164, 1991.

[24] Matthew Chapman and Gernot Heiser. vNUMA: A virtual shared-memory multiprocessor. In *USENIX Annual Technical Conference*, pages 349–362, 2009.

[25] Christopher Clark, Keir Fraser, Steven Hand, Jacob Gorm Hansen, Eric Jul, Christian Limpach, Ian Pratt, and Andrew Warfield. Live migration of virtual machines. In *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation-Volume 2*, pages 273–286, 2005.

[26] Eli Cortez, Anand Bonde, Alexandre Muzio, Mark Russinovich, Marcus Fontoura, and Ricardo Bianchini. Resource central: Understanding and predicting workloads for improved resource management in large cloud platforms. In *Proceedings of the 26th Symposium on Operating Systems Principles*, pages 153–167, 2017.

[27] Christina Delimitrou and Christos Kozyrakis. Quasar: resource-efficient and QoS-aware cluster management. *ACM SIGPLAN Notices*, 49(4):127–144, 2014.

[28] Google Developers. Improve Server Response Time, Dec 2018. https://developers.google.com/speed/docs/insights/Server#overview.

[29] Yaozu Dong, Xiaowei Yang, Jianhui Li, Guangdeng Liao, Kun Tian, and Haibing Guan. High performance network virtualization with SR-IOV. *Journal of Parallel and Distributed Computing*, 72(11):1471–1480, 2012.

[30] Aleksandar Dragojević, Dushyanth Narayanan, Miguel Castro, and Orion Hodson. FaRM: Fast remote memory. In *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*, pages 401–414, 2014.

[31] Aleksandar Dragojević, Dushyanth Narayanan, Miguel Castro, and Orion Hodson. Farm: Fast remote memory. In *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*, pages 401–414, Seattle, WA, April 2014. USENIX Association.

[32] Eolas. Eolas website, 2020. https://www.eolas.fr/.

[33] Peter X. Gao, Akshay Narayan, Sagar Karandikar, Joao Carreira, Sangjin Han, Rachit Agarwal, Sylvia Ratnasamy, and Scott Shenker. Network Requirements for Resource Disaggregation. In *Proceedings of the 12nd OSDI*, Savannah, GA, November 2016.

[34] Adam Geitgey. Open-source face detection algorithm, 2020. https://github.com/ageitgey/face_recognition.

[35] Ali Ghodsi, Matei Zaharia, Benjamin Hindman, Andy Konwinski, Scott Shenker, and Ion Stoica. Dominant resource fairness: Fair allocation of multiple resource types. In *NSDI*, volume 11, pages 24–24, 2011.

[36] Google. Google Cloud Functions, 2020. https://cloud.google.com/functions.

[37] Google. Spot vms, 2021. https://cloud.google.com/spot-vms/.

[38] Robert Grandl, Ganesh Ananthanarayanan, Srikanth Kandula, Sriram Rao, and Aditya Akella. Multi-resource packing for cluster schedulers. *ACM SIGCOMM Computer Communication Review*, 44(4):455–466, 2014.

[39] Juncheng Gu, Youngmoon Lee, Yiwen Zhang, Mosharaf Chowdhury, and Kang G Shin. Efficient memory disaggregation with infiniswap. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, pages 649–667, 2017.

[40] Juncheng Gu, Youngmoon Lee, Yiwen Zhang, Mosharaf Chowdhury, and Kang G. Shin. Efficient memory disaggregation with infiniswap. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, pages 649–667, Boston, MA, March 2017. USENIX Association.

[41] Ori Hadary, Luke Marshall, Ishai Menache, Abhisek Pan, Esaias E Greeff, David Dion, Star Dorminey, Shailesh Joshi, Yang Chen, Mark Russinovich, et al. Protean:{VM} allocation service at scale. In *14th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 20)*, pages 845–861, 2020.

[42] Sangjin Han, Norbert Egi, Aurojit Panda, Sylvia Ratnasamy, Guangyu Shi, and Scott Shenker. Network Support for Resource Disaggregation in Next-Generation Datacenters. In *Proceedings of the 14th HotNets*, Philadelphia, PA, November 2015.

[43] Scott Hendrickson, Stephen Sturdevant, Tyler Harter, Venkateshwaran Venkataramani, Andrea C Arpaci-Dusseau, and Remzi H Arpaci-Dusseau. Serverless computation with openlambda. In *8th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 16)*, 2016.

[44] Benjamin Hindman, Andy Konwinski, Matei Zaharia, Ali Ghodsi, Anthony D Joseph, Randy Katz, Scott Shenker, and Ion Stoica. Mesos: A platform for {Fine-Grained} resource sharing in the data center. In *8th USENIX Symposium on Networked Systems Design and Implementation (NSDI 11)*, 2011.

[45] IDC. Global ethernet switch and router markets deliver mixed results in q2 2020, according to idc. Online: https://www.idc.com/getdoc.jsp?containerId=prUS46830820, accessed 01-01-2021.

[46] Calin Iorgulescu, Reza Azimi, Youngjin Kwon, Sameh Elnikety, Manoj Syamala, Vivek Narasayya, Herodotos Herodotou, Paulo Tomita, Alex Chen, Jack Zhang, and Junhua Wang. Perfiso: Performance isolation for commercial latency-sensitive services. In *2018 USENIX Annual Technical Conference (USENIX ATC 18)*, pages 519–532, Boston, MA, July 2018. USENIX Association.

[47] Eric Jonas, Johann Schleier-Smith, Vikram Sreekanti, Chia-Che Tsai, Anurag Khandelwal, Qifan Pu, Vaishaal Shankar, Joao Carreira, Karl Krauth, Neeraja Yadwadkar, et al. Cloud programming simplified: A berkeley view on serverless computing. *arXiv preprint arXiv:1902.03383*, 2019.

[48] Mohamed Karaoui, Brice Teguia, Bernabe Batchakui, and Alain Tchana. Analysis of a modern distributed hypervisor: what we learn from our experiments. The 11th Workshop on Systems for Post-Moore Architectures, 2022.

[49] David Katz, Antonio Barbalace, Saif Ansary, Akshay Ravichandran, and Binoy Ravindran. Thread migration in a replicated-kernel os. In *2015 IEEE 35th International Conference on Distributed Computing Systems*, pages 278–287. IEEE, 2015.

[50] Kerrighed. Kerrighed: linux clusters made easy. http://www.kerrighed.org/wiki/index.php, September 2010.

[51] Sang-Hoon Kim, Ho-Ren Chuang, Robert Lyerly, Pierre Olivier, Changwoo Min, and Binoy Ravindran. Dex: Scaling applications beyond machine boundaries. In *2020 IEEE 40th International Conference on Distributed Computing Systems (ICDCS)*, pages 864–876. IEEE, 2020.

[52] kubernetes Documentation. Kubernetes scheduler (kube-scheduler), 2022. https://kubernetes.io/docs/concepts/scheduling-eviction/kube-scheduler/.

[53] KVM Contributors. Main page — KVM website. https://www.linux-kvm.org/index.php?title=Main_Page&oldid=173792, 2016. [Online; accessed 2-August-2017].

[54] Ang Li, Shuaiwen Leon Song, Jieyang Chen, Jiajia Li, Xu Liu, Nathan R. Tallent, and Kevin J. Barker. Evaluating modern gpu interconnect: Pcie, nvlink, nv-sli, nvswitch and gpudirect. *IEEE Transactions on Parallel and Distributed Systems*, 31(1):94–110, 2020.

[55] Kai Li and Paul Hudak. Memory coherence in shared virtual memory systems. *ACM Transactions on Computer Systems (TOCS)*, 7(4):321–359, 1989.

[56] Kai Li and Paul Hudak. Memory coherence in shared virtual memory systems. *TOCS*, 7(4):321–359, 1989.

[57] Kevin Lim, Jichuan Chang, Trevor Muge, Parthasarathy Ranganathan, Steven K. Reinhardt, and Thomas F. Wenisch. Disaggregated memory for expansion and sharing in blade servers. In *Proceedings of the 35th ISCA*, Austin, TX, USA, 2008.

[58] Kevin Lim, Yoshio Turner, Jose Renato Santos, Alvin AuYoung, Jichuan Chang, Parhasarathy Ranganathan, and Thomas F. Wenisch. System-level implications of disaggregated memory. In *Proceedings of the 18th HPCA*, New Orleans, Louisiana, USA, February 2012.

[59] David Lo, Liqun Cheng, Rama Govindaraju, Parthasarathy Ranganathan, and Christos Kozyrakis. Heracles: Improving resource efficiency at scale. In *2015 ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA)*, pages 450–462, 2015.

[60] David Meisner, Brian T. Gold, and Thomas F. Wenisch. The powernap server architecture. *ACM Trans. Comput. Syst.*, 29(1), February 2011.

[61] Microsoft. Microsoft Azure Functions, 2020. https://azure.microsoft.com/en-us/services/functions.

[62] Microsoft. Azure spot virtual machines, 2021. https://azure.microsoft.com/en-gb/services/virtual-machines/spot.

[63] Microsoft Azure. Microsoft azure public dataset repository, 2020. https://github.com/Azure/AzurePublicDataset.

[64] Sape J. Mullender, Guido Van Rossum, AS Tananbaum, Robbert Van Renesse, and Hans Van Staveren. Amoeba: A distributed operating system for the 1990s. *Computer*, 23(5):44–53, 1990.

[65] NASA Advanced Supercomputing Division. NAS parallel benchmarks. https://tinyurl.com/y47k95cc.

[66] Gil Neiger, Amy Santoni, Felix Leung, Dion Rodgers, and Rich Uhlig. Intel virtualization technology: Hardware support for efficient processor virtualization. *Intel Technology Journal*, 10(3), 2006.

[67] Jacob Nelson, Brandon Holt, Brandon Myers, Preston Briggs, Luis Ceze, Simon Kahan, and Mark Oskin. Latency-tolerant software distributed shared memory. In *Proceedings of the 2015 ATC*, pages 291–305, Santa Clara, CA, July 2015.

[68] Yahoo Developer Network. Best Practices for Speeding Up Your Web Site, Dec 2006. https://developer.yahoo.com/performance/rules.html.

[69] Edmund B Nightingale, Orion Hodson, Ross McIlroy, Chris Hawblitzel, and Galen Hunt. Helios: heterogeneous multiprocessing with satellite kernels. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, pages 221–234, 2009.

[70] Vlad Nitu, Boris Teabe, Leon Fopa, Alain Tchana, and Daniel Hagimont. Stopgap: Elastic vms to enhance server consolidation. *Software: Practice and Experience*, 47(11):1501–1519, 2017.

[71] Vlad Nitu, Boris Teabe, Alain Tchana, Canturk Isci, and Daniel Hagimont. Welcome to zombieland: Practical and energy-efficient memory disaggregation in a datacenter. In *Proceedings of the Thirteenth EuroSys Conference*, pages 1–12, 2018.

[72] NVIDIA. Gpudirect rdma, direct communication between nvidia gpus, 2021. https://developer.nvidia.com/gpudirect.

[73] Pierre Olivier, Sang-Hoon Kim, and Binoy Ravindran. OS support for thread migration and distribution in the fully heterogeneous datacenter. In *Proceedings of the 16th HOTOS (HotOS XVI)*, pages 174–179, Whistler, BC, Canada, February 2017. ACM.

[74] Pierre Olivier, Binoy Ravindran, and Antonio Barbalace. The multihype: Virtualizing heterogeneous-isa architectures. In *9th Workshop on Systems for Multi-core and Heterogeneous Architectures (SFMA)*, 2019.

[75] RedHat OpenShift. VHost-net, Jan 2020. http://www.linux-kvm.org/page/UsingVhost.

[76] Loïc Perennou, Mar Callau-Zori, Sylvain Lefebvre, and Raka Chiky. Workload characterization for a non-hyperscale public cloud platform. In *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*, pages 409–413. IEEE, 2019.

[77] Ian Pratt, Andrew Warfield, P Barham, and R Neugebauer. Xen and the art of virtualization. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles*, pages 113–1118, 2003.

[78] Princeton University. The PARSEC benchmark suite. http://parsec.cs.princeton.edu.

[79] Jelica Protic, Milo Tomasevic, and Veljko Milutinovic. Distributed shared memory: Concepts and systems. *IEEE Parallel & Distributed Technology: Systems & Applications*, 4(2):63–71, 1996.

[80] Md Golam Rabbani, Rafael Pereira Esteves, Maxim Podlesny, Gwendal Simon, Lisandro Zambenedetti Granville, and Raouf Boutaba. On tackling virtual data center embedding problem. In *2013 IFIP/IEEE International Symposium on Integrated Network Management (IM 2013)*, pages 177–184. IEEE, 2013.

[81] Charles Reiss, Alexey Tumanov, Gregory R Ganger, Randy H Katz, and Michael A Kozuch. Heterogeneity and dynamicity of clouds at scale: Google trace analysis. In *Proceedings of the Third ACM Symposium on*

Cloud Computing, pages 1–13, 2012.

[82] Rusty Russell. virtio: towards a de-facto standard for virtual i/o devices. *ACM SIGOPS Operating Systems Review*, 42(5):95–103, 2008.

[83] Einar Rustad. Numascale numaconnect. White paper, online: https://www.numascale.com/numa_pdfs/numaconnect-white-paper.pdf, accessed 2017-08-08.

[84] Marina Sadini, Antonio Barbalace, Binoy Ravindran, and Francesco Quaglia. A page coherency protocol for popcorn replicated-kernel operating system. In *Proceedings of the 2013 Many-Core Architecture Research Community Symposium (MARC)*, October 2013.

[85] Yizhou Shan, Yutong Huang, Yilun Chen, and Yiying Zhang. LegoOS: A disseminated, distributed OS for hardware resource disaggregation. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, pages 69–87, 2018.

[86] Yizhou Shan, Shin-Yeh Tsai, and Yiying Zhang. Distributed shared persistent memory. In *Proceedings of the 2017 Symposium on Cloud Computing*, pages 323–337, 2017.

[87] Bikash Sharma, Victor Chudnovsky, Joseph L Hellerstein, Rasekh Rifaat, and Chita R Das. Modeling and synthesizing task placement constraints in google compute clusters. In *Proceedings of the 2nd ACM Symposium on Cloud Computing*, pages 1–14, 2011.

[88] A. K. Somani and N. H. Vaidya. Understanding fault tolerance and reliability. *Computer*, 30(04):45–50, apr 1997.

[89] Alain Tchana and Renaud Lachaize. Rebooting virtualization research (again). In *Proceedings of the 10th ACM SIGOPS Asia-Pacific Workshop on Systems*, pages 99–106, Hangzhou, China, August 2019.

[90] TidalScale. Tidalscale website, 2020. https://www.tidalscale.com/.

[91] TidalScale. Achieving painless reliability – an alternate view. https://www.tidalscale.com/achieving-painless-reliability-an-alternate-view/, August 2021.

[92] UEFI Forum. Advanced configuration and powerinterface (acpi) specification, 2019. https://github.com/Ahttps://uefi.org/sites/default/files/resources/ACPI_6_3_May16.pdfzure/AzurePublicDataset.

[93] W3Techs. Usage statistics and market share of wordpress. https://w3techs.com/technologies/details/cm-wordpress/all/all, 2021.

[94] Jason Wang. Multiqueue - kvm website. https://www.linux-kvm.org/page/Multiqueue, 2016.

[95] Liang Wang, Mengyuan Li, Yinqian Zhang, Thomas Ristenpart, and Michael Swift. Peeking behind the curtains of serverless platforms. In *Proceedings of the 2018 USENIX Conference on Usenix Annual Technical Conference*, USENIX ATC '18, page 133–145, 2018.

[96] Yawen Wang, Kapil Arya, Marios Kogias, Manohar Vanga, Aditya Bhandari, Neeraja J. Yadwadkar, Siddhartha Sen, Sameh Elnikety, Christos Kozyrakis, and Ricardo Bianchini. *SmartHarvest: Harvesting Idle CPUs Safely and Efficiently in the Cloud*, page 1–16. 2021.

[97] Jin Zhang, Zhuocheng Ding, Yubin Chen, Xingguo Jia, Boshi Yu, Zhengwei Qi, and Haibing Guan. Giantvm: A type-ii hypervisor implementing many-to-one virtualization. In *VEE'20*, 2020.