Towards Microsecond-Scale VM Core Provisioning Agility on Serverless Platforms

Yibo Yan
University of Southern California
Los Angeles, USA

Seo Jin Park
University of Southern California
Los Angeles, USA

ABSTRACT

Bursty, data-intensive serverless workloads, such as serverless big data analytics and DNN inference, amplify the fundamental tension for cloud providers between meeting strict tail-latency service-level objectives (SLOs) and maintaining cost-effectiveness. A key reason is that today's multi-tenant platforms cannot reallocate physical cores between hardware-isolated virtual machines (VMs) at the microsecond speeds these applications require. This capability is critical for instantly responding to demand spikes by shifting resources from low-priority to latency-sensitive tasks.

This position paper introduces HyperFlux, a hyper-reactive virtualization stack comprising three co-designed components: a virtual machine monitor (VMM), a lightweight guest OS, and a Linux kernel module as its core arbiter. HyperFlux achieves core reallocation and vertical core scaling (up/down) among hardware-isolated VMs within tens of microseconds. By coupling transparent, fast vertical core scaling with microsecond-level core reallocation, HyperFlux advances hardware-isolated (KVM) serverless provisioning agility and resource elasticity to the microsecond scale. We discuss the system design and aim to stimulate discussion on next-generation, cloud-native computing with extreme agility.

CCS CONCEPTS

• **Computer systems organization** → **Cloud computing**; • **Software and its engineering** → *Operating systems*; • **Security and privacy** → **Virtualization and security**.

KEYWORDS

serverless computing, vertical scaling, lightweight isolation, KVM, core (re)allocation

APSys '25, October 12–13, 2025, Seoul, Republic of Korea

ACM ISBN 979-8-4007-1572-3/25/10
https://doi.org/10.1145/3725783.3764396

ACM Reference Format:
Yibo Yan and Seo Jin Park. 2025. Towards Microsecond-Scale VM Core Provisioning Agility on Serverless Platforms . In *16th ACM SIGOPS Asia-Pacific Workshop on Systems (APSys '25), October 12–13, 2025, Seoul, Republic of Korea.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3725783.3764396

1 INTRODUCTION

Serverless computing abstracts resource management away from developers, allowing the platform operator to allocate computation resources on demand and to reclaim them when workload pressure subsides. This model promises both higher utilization for the cloud provider and lower cost for the tenant. In practice, however, cloud providers must maintain substantial head-room—by pre-warming a pool of microVMs or reserving idle cores—so that each request meets its tail-latency Service-Level Objective (SLO). Such head-room comes at a direct monetary cost, eroding the pay-as-you-go value proposition [28, 56, 61, 70].

A new generation of burst-style, data-intensive serverless workloads amplifies the tension. Serverless big data analytics and serverless DNN inference [2, 9, 11, 12] all exhibit high burstiness: the arrival rate can surge by orders of magnitude in a few milliseconds. They also require varying parallelism over requests: the ideal degree of parallelism changes dramatically from one request to another. For instance, consider aggregation operations on a serverless big data analytics platform. A small aggregation can complete on a few dozen cores, whereas a large join over tens of gigabytes could require an order of magnitude more cores. Here, the degree of parallelism is unknown until the query arrives at the system; thus, we cannot allocate cores in advance.

The combination of bursty request arrivals, request-specific parallelism, and real-time SLOs poses challenges for today's infrastructure in aligning resource allocation with demands. Over-provisioning for the worst-case demand of bursty, data-intensive workloads across all tenants in the cloud is economically untenable, as it leads to substantial resource waste. Yet, without over-provisioned resources to handle peak demand, some requests may require a degree of parallelism that exceeds the allocated computation capacity, resulting in SLO violations.

Prior studies have demonstrated that microsecond core reallocation mechanism can effectively maintain microsecond-scale tail latency without wasting cores for over-provisioning [27, 36, 49, 52]. However, none of them targets multi-tenant cloud environments, which heavily rely on VM-based isolation to provide inter-tenant security guarantees. Therefore, multi-tenant cloud serverless platforms demand a fast core reallocation mechanism that matches these new burst-style, data-intensive serverless workload's time scale, assigning or revoking hundreds of cores in a few tens of microseconds while preserving hardware-based isolation among tenants, thereby providing microsecond-scale VM core provisioning agility.

We present HyperFlux, a hyper-reactive virtualization substrate. HyperFlux is co-designed with a virtual machine monitor (VMM), a lightweight guest OS, and a Linux kernel module to (1) vertically scale up and down VMs by scheduling and de-scheduling their virtual CPUs (vCPUs) and (2) dynamically reallocate physical cores among VMs based on the demand—both within tens of microseconds and without requiring guest application modification and cooperation.

The remainder of this paper is organized as follows: §2 motivates microsecond-scale VM vertical elasticity and core reallocation; §3 details the design of HyperFlux and the rationale of our design choices; §4 discusses the potential of HyperFlux for the future cloud computing.

2 BACKGROUND AND MOTIVATION

Microsecond-scale VM core provisioning agility consists of two complementary capabilities: *vertical core elasticity*, which resizes a VM by adding or removing vCPUs, and *core reallocation*, which shifts physical cores across VMs. This section primarily focuses on motivating why both operations must be completed within tens of microseconds to support emerging latency-sensitive, bursty, and data-driven serverless workloads.

2.1 Vertical Core Elasticity

Burst-style, data-driven serverless workloads inherently exhibit both *inter-request* and *intra-request* parallelism dynamism. At the inter-request level, different requests may process vastly different volumes of data, leading to highly variable parallelism requirements to satisfy SLOs. At the intra-request level, parallelism needs fluctuate across stages within the same query—for example, aggregation, shuffle, and reduce often impose distinct and shifting degrees of parallelism that are difficult to anticipate in advance [17, 20, 30, 72]. This dual form of parallelism dynamism makes static core allocation ineffective. To consistently meet tail-latency SLOs under bursty arrivals, application VMs must support rapid vertical core scaling, acquiring additional cores on demand as parallelism requirements surge.

Equally important, a serverless platform must avoid resource waste by reclaiming unused cores when parallelism demand subsides. Thus, the infrastructure should not only scale VMs up under pressure but also scale them down once additional computation resources are no longer required, thereby achieving *vertical core elasticity*.

Achieving fast vertical core elasticity, however, presents considerable challenges. Uncoordinated removal of vCPUs from a guest VM can destabilize execution, causing scheduling anomalies and inducing incorrect behaviors. Specifically, if a vCPU is abruptly de-scheduled from the physical core without proper coordination with the guest kernel, the guest kernel may still treat it as active and continue scheduling tasks onto it. On the other hand, while CPU hotplug [8] offers a mechanism for coordinated vertical core scaling, its inherent high latency (ranging from hundreds of milliseconds to seconds) renders it unsuitable for burst-style serverless applications that have microsecond-level tail-latency SLOs.

Takeaway 1

To host bursty, data-driven serverless applications in a cost-effective manner, the serverless infrastructure must be able to scale VMs up within tens of microseconds to accommodate unpredictable parallelism demands, and scale them down promptly to reclaim resources once demand subsides.

2.2 Microsecond-Scale Core Reallocation

Beyond enabling vertical core elasticity within a VM, a serverless platform must also provision sufficient cores instantly to absorb sudden bursts of applications that demand microsecond tail-latency SLOs. Any delay in provisioning cores can inflate tail latency and potentially lead to SLO violations.

Given the importance of high resource utilization and service density [28, 56, 61, 70], serverless platforms typically colocate diverse workloads, mixing applications with strict tail-latency SLOs and others with looser latency requirements. To meet the strict SLOs of bursty, data-driven workloads, the serverless platform can harvest cores from latency-insensitive applications and reallocate them to latency-sensitive tasks.

A body of prior work has explored fast core (re)allocation on datacenter systems to protect latency-sensitive applications by promptly harvesting cores from latency-insensitive one [27, 36, 49, 52]. This approach avoids over-provisioning, reduces core waste, and improves resource utilization. However, these studies assume that applications run in a single trusted domain. Namely, applications running on these systems are not isolated by a hypervisor.

In contrast, multi-tenant cloud environments require hardware-enforced isolation, which introduces an additional layer of complexity and poses new challenges for fast core reallocation. To achieve microsecond-scale core reallocation across VMs, the system must rapidly and safely preempt cores from latency-insensitive application VMs and reassign cores to latency-sensitive ones by directly manipulating vCPU-to-core mappings.

Takeaway 2

Serverless platforms demand a microsecond-scale core reallocation mechanism that can swiftly shift cores to tail-latency-sensitive workloads, ensuring SLO compliance without sacrificing resource utilization.

2.3 Lightweight Guest OS for Performance

Booting a full Linux kernel inside virtual machines incurs prohibitive overhead on the critical path, making it unsuitable for fine-grained, low-latency serverless workloads [21, 37, 45, 64]. To mitigate this cost, recent systems [13, 26, 58, 67] adopt the concept of a *LibOS* (library OS [25, 50]), which provides a subset of OS interfaces as a library that linked directly with user applications. A LibOS offers substantial performance advantages as it is loaded in the guest application's address space to emulate most system calls and avoid most context switches and `VMExits`, while relying on a thin host interface for hardware resource multiplexing and management.

An alternative approach to reducing startup latency, exemplified by Hyperlight [14], is to compile essential guest OS functionalities directly into the application binary. This approach, conceptually similar to *unikernels* [39, 44], eliminates the need for a separate guest OS instantiation, thereby pushing cold-start latency even lower. The trade-off, however, is reduced compatibility: applications must be recompiled, and existing binaries cannot run unmodified.

3 HYPERFLUX: MICROSECOND-SCALE VM CORE PROVISIONING AGILITY

We introduce HyperFlux, a hyper-reactive, hardware-based virtualization stack that enables microsecond-scale core provisioning agility. Figure 1 illustrates the architecture of HyperFlux and highlights its three primary components in purple: FluxOS, Fluxion, and kFlux. FluxOS is a lightweight LibOS that serves as the guest OS. Fluxion acts as the virtual machine monitor (VMM), responsible for instantiating and managing VMs and coordinating their vCPUs, similar in role to Firecracker [18]. Finally, kFlux is a Linux kernel module running on the host that manages core preemption in response to changes in parallelism demand.

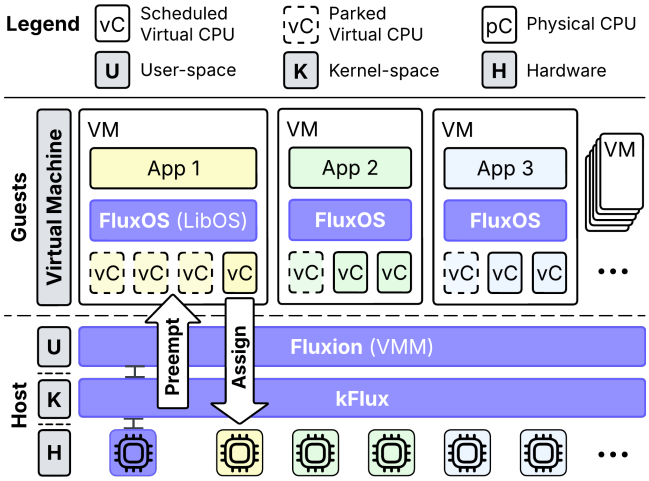

Figure 1: The HyperFlux architecture. Three primary components in purple: FluxOS serves as the guest OS for user applications; Fluxion serves as the virtual machine monitor (VMM), coordinating with kFlux to perform fast core reallocation.

Unlike traditional VMM (*e.g.,* Firecracker), Fluxion actively coordinates with kFlux to perform safe vCPU-to-core assignment and achieve microsecond-scale core reallocation.

3.1 Threat Model

HyperFlux assumes that all three components—FluxOS, Fluxion, and kFlux—are trusted. In contrast, user applications are not trusted. The entire guest VM is managed by the serverless service provider via HyperFlux. Memory isolation is enforced by hardware via KVM, along with other hypervisor-based security guarantees.

FluxOS is a preemptive OS and it can forcibly preempt threads of user applications. In particular, coordination between Fluxion and FluxOS cannot be blocked by a misbehaving or buggy application that indefinitely holds a lock.

3.2 Lightweight Isolation

HyperFlux loads FluxOS, a lightweight LibOS, to avoid the cost of loading a full Linux kernel. FluxOS directly handles system calls from user applications, enabling HyperFlux to host existing, unmodified software. FluxOS provides a custom ELF loader that resolves the system call jump addresses to FluxOS implementations when loading a binary.

To favor a minimalist design and reduce latency incurred during ELF loading and system call address resolving, FluxOS does not implement the complete set of Linux system calls. A recent study [26] reports that implementing 126 Linux system calls can effectively support a variety of applications written in C/C++ and Rust, as well as several language runtimes, including Python, Go, Node.js, and Java.

Furthermore, inspired by Virtines [64], HyperFlux reuses existing KVM structures (*i.e.,* virtual contexts) to

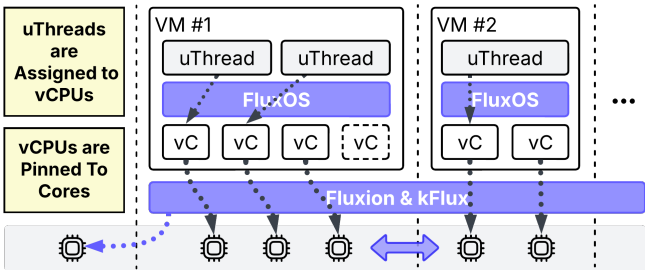

Figure 2: HyperFlux performs vertical core scaling by (un)pinning vCPUs on cores, thereby adjusting the parallelism capacity for `uThreads` inside the VM.

bypass the expensive VM initialization incurred during the `KVM_CREATE_VM` ioctl. Prior work such as Dandelion [38] shows that reusing KVM structures can potentially reduce the VM startup latency to the sub-millisecond level.

3.3 Parallelism Management

Threading. As shown in Figure 2, FluxOS provides a high-performance, user-level threading library to the guest application. Each vCPU has its own task queue and a scheduler that polls the queue to schedule tasks, *a.k.a.*, user-level threads (uThread). FluxOS employs work stealing to balance lightweight uThreads across vCPUs—a strategy widely recognized as effective for minimizing tail latency in prior studies [27, 31, 51, 66].

Interface. For compatibility, FluxOS provides the standard `pthread` (POSIX Threads) interface to guest applications by intercepting and re-routing `pthread` calls to FluxOS's threading library during ELF loading. We also intend to explore new APIs that can help demanding applications to more explicitly and efficiently express their parallelism demands.

vCPU Management. Each vCPU runs inside a dedicated kernel thread (kThread) on the host. In Figure 2, Fluxion and kFlux activate a vCPU by pinning its corresponding kThread to a physical core, thereby increasing the number of active vCPUs available to a VM. Similarly, they de-schedule these kThreads to revoke vCPUs from VMs, thereby decreasing the VM's compute capacity.

Internal Concurrency. To fully benefit from vertical core scaling, we note that user applications must expose internal concurrency, *i.e.*, spawning new threads for parallel, independent tasks. For example, single-threaded, event-loop-based applications such as Redis [16] cannot benefit from vertical core scaling without modifications. Specifically, one uThread can only be scheduled and executed on one vCPU at a time; therefore, adding additional cores does not effectively increase the compute capacity. Nevertheless, we argue that with the evolution of cloud-native programming, the cloud software landscape is expected to evolve and adapt to new infrastructure paradigms accordingly [19, 23, 34, 60, 68].

3.4 Vertical Core Scaling

HyperFlux instantiates VMs ahead of time, each with a specified maximum number of vCPUs it can utilize. These VMs are then kept in a *dormant* state, where no vCPUs are assigned to physical cores and, consequently, no computation resources are consumed. To scale up and accommodate bursty tasks, HyperFlux dynamically activates dormant VMs by assigning their vCPUs to physical cores. This mechanism provides additional computation capacity and parallelism to user applications through vertical scaling.

Ditto [71] experimented with application-assisted vCPU vertical scaling, reporting a 14 μs latency for vCPU addition and removal, indicating the feasibility of microsecond-level scaling. Building on this insight, HyperFlux advances further by providing application-transparent vertical core scaling through the coordination of FluxOS and Fluxion.

Downscaling with Safe Parking. HyperFlux vertically scale down VMs by de-scheduling their vCPUs from physical cores. However, blindly de-scheduling vCPUs can disrupt execution. For example, if a task running on a vCPU is holding a lock and that vCPU is abruptly de-scheduled, other tasks needing to acquire the lock cannot progress properly.

To avoid such hazards, HyperFlux coordinates Fluxion and FluxOS to safely park vCPUs before de-scheduling. Specifically, Fluxion first notifies FluxOS of the to-be-parked vCPU. FluxOS preempts the currently running task (if any) on that vCPU and saves its execution context. Subsequently, FluxOS suspends the vCPU so that no new tasks can be scheduled onto it. Finally, FluxOS signals Fluxion that the vCPU is safe to de-schedule, and Fluxion coordinates with kFlux to de-schedule the vCPU from the physical core. Fluxion puts the de-scheduled vCPU into the dormant state and yields the core to other VMs.

While a regular Linux guest kernel also support vCPU hotplug via a similar sequence, the operation typically completes at second-scale latencies [8, 46]. In contrast, the tight integration of lightweight FluxOS with Fluxion reduces this latency to tens of microseconds.

Progressiveness after Parking. The parked vCPU may have a queue of pending tasks; FluxOS ensures progressiveness by allowing active vCPUs to steal tasks from parked ones. However, solely relying on work stealing can cause load imbalance and increased tail latencies for tasks originally queued on the parked vCPU. We plan to explore proactive task migration to evaluate whether the additional complexity is justified by potential performance benefits.

If tasks on parked vCPUs hold locks, active vCPUs will eventually steal and execute them, thereby preserving progressiveness. Nevertheless, parking vCPUs under such conditions can temporarily increase latency, as execution must wait until an active vCPU picks up the task and other tasks

that need to acquire the lock are delayed. In practice, HyperFlux downscales a VM primarily when higher-priority VMs require additional resources or when the downscaled VM no longer needs as much computation capacity. In such cases, the modest latency increase is an acceptable trade-off.

Voluntary Yield. When a vCPU exhausts its local task queue and cannot find stealable work from other vCPUs, FluxOS voluntarily yields the core back to Fluxion. Yielded cores can be reassigned to other VMs, avoiding resource waste and preventing core stranding.

Upscaling. Fluxion schedules the corresponding vCPUs onto physical cores and signals them to resume normal scheduling via FluxOS. If the newly scheduled vCPUs have no queued tasks, their schedulers engage in work stealing to acquire tasks from other vCPUs.

3.5 Core Reallocation

Inspired by Caladan [27], HyperFlux dedicates a physical core to busy-polling for monitoring and enabling microsecond-scale core reallocation using Fluxion and kFlux. Unlike Caladan, however, HyperFlux supports core reallocation across VM boundaries, a capability essential in multi-tenant environments.

Monitoring and Reaction. HyperFlux supports both *proactive* and *reactive* core reallocation. In proactive reallocation, when a burst-style request with strict SLOs arrives, the platform can instruct Fluxion to (re)allocate sufficient cores to the corresponding VM before the request generates a large number of parallel tasks. Proactive reallocation alone, however, is not sufficient: computation jobs often consist of multiple stages, and the concurrency level in intermediate stages is difficult to predict in advance [17, 20, 30, 72].

Therefore, HyperFlux also performs reactive reallocation by monitoring runtime metrics and adjusting cores accordingly. For example, kFlux can track the average task (uThread) and network packet queuing delay of a VM via FluxOS. If the average queuing delay exceeds a predefined threshold, Fluxion reallocates additional cores to the VM, provided it has higher priority. Prior systems such as Shenango [49] and Caladan [27] have examined the challenging problem of identifying effective metrics and reacting within tens of microseconds in datacenter environments. HyperFlux builds on these insights to enable inter-VM core reallocation in multi-tenant settings.

Core Preemption and Assignment. To reallocate physical cores among VMs, HyperFlux performs core preemption and assignment in a tight loop. For preemption, HyperFlux triggers the safe parking mechanism on target vCPUs so that these vCPUs can be de-scheduled safely. Once reclaimed, these cores are reassigned to VMs requiring additional computation capacity. Both Shenango and Caladan [27, 49] have demonstrated that such operations, while seemingly complex, can be completed within tens of microseconds.

3.6 Low-Overhead Host-Guest Communication

HyperFlux is designed around the core principle of tight, efficient coordination between the guest OS (FluxOS) and the VMM (Fluxion). To achieve microsecond-level efficient coordination, HyperFlux relies heavily on hypercalls and a shared-memory control plane.

Hypercalls in FluxOS serve two key purposes: 1) eliminating unnecessary `VMExits` during Linux system call handling, and 2) enabling low-overhead voluntary core yielding and safe parking. In addition, Fluxion establishes a shared memory region with each guest VM running FluxOS. Fast core reallocation requires aggressive polling of VM runtime states. Each VM must promptly populate aggregated runtime metrics into the shared memory so that kFlux can make accurate reallocation decisions at microsecond timescales.

However, efficiently coordinating and managing these metrics between guest and host at scale remains an open question for exploration. Furthermore, scaling the shared-memory control plane to thousands of VMs demands careful design to ensure both efficiency and robustness.

3.7 Language Runtime and Dependency Initialization Latency

Prior studies [24, 35, 41, 42, 53] revealed that loading language runtimes (*e.g.,* JVM or Python) and dependencies (*e.g.,* third-party libraries) incurs non-negligible overhead in the critical path of cold start. To mitigate this, HyperFlux follows the conventional approach of pre-warming and pooling, maintaining a pool of VMs with pre-initialized language runtimes and dependencies. However, aggressive pre-warming and pooling can induce substantial memory overhead, which is uneconomical for serverless platforms. Fortunately, HyperFlux's vertical scaling capabilities can alleviate the need for such aggressive pre-warming and pooling (detailed in §4.3).

4 DISCUSSIONS

4.1 Tiered Cold-Start Mitigation Strategy

Mitigating cold-start delays has been a crucial research focus in serverless computing. Prior work has largely emphasized minimizing sandbox start-up time [18, 48, 59] or avoiding cold starts on the critical path [21, 24, 41, 65]. We contend that neither approach alone is sufficient, particularly for emerging bursty, data-driven serverless workloads.

HyperFlux strikes this balance through a *tiered cold-start mitigation* strategy, illustrated in Figure 3, with latency projections based on prior work. Tiers are ordered from top

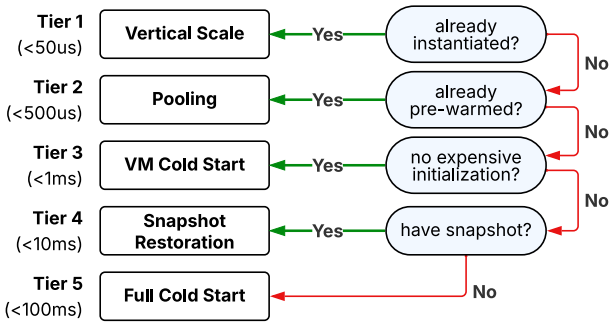

Figure 3: HyperFlux's tiered strategy for mitigating cold-start latencies during upscaling. Higher tiers offer lower scaling latency.

to bottom by increasing latency, and HyperFlux degrades gracefully to a lower tier only when the requirements of a higher tier cannot be met. The *VM Cold Start* tier (tier 3) applies when a job requires a VM but does not involve heavyweight language runtime or dependency initialization; thanks to HyperFlux 's lightweight VM design, latency in this tier remains below 1 ms. If a job demands costly runtime initialization and no applicable snapshot is available, the platform falls back to a *Full Cold Start*—a path prohibitively slow for microsecond-scale tasks.

The overarching goal of HyperFlux is to keep provisioning paths within the top three tiers whenever possible, thereby ensuring sub-millisecond provisioning latency.

4.2 TEE-Ready Substrate

HyperFlux aligns well with the cloud's shift toward Trusted Execution Environments (TEEs). Hardware-backed confidential VMs (CVMs), *e.g.,* AMD SEV-SNP [1, 4, 10] and AWS Nitro [3], have become the dominant TEE primitive in the cloud because they graft attested memory-encryption onto the mature VM management stack, shielding tenant code even from a compromised host OS. Since HyperFlux is implemented natively atop KVM, it can readily benefit from CVM protections.

In contrast, existing container-based resource orchestrators and provisioners are poorly suited for latency-sensitive serverless computing with TEEs. Most confidential container (CC) solutions nest the container runtime inside a CVM [5–7], introducing an additional control plane and extra `VMExit` overheads. HyperFlux avoids this indirection and additional layers of overhead, preserving the microsecond-scale latency budget that these workloads require.

CVMs do retain a known weakness—cold-start latencies on the order of hundreds of milliseconds to seconds—primarily due to the cost of attesting memory contents [32], rather than vCPU topology. Fortunately, Ditto [71] shows that securely adding or removing vCPUs in a CVM can be completed in only ≈14 μs. HyperFlux inherits this capability out-of-the-box: a pool of dormant, pre-attested CVMs can be kept resident, while Fluxion's microsecond vertical-scaling path lends or reclaims encrypted vCPUs on demand. This design masks cold-start costs, delivering TEE guarantees without sacrificing provision agility.

4.3 Memory Overhead with Vertical Scaling

HyperFlux's vertical scaling incurs significantly less memory overhead than conventional pre-warming and pooling techniques [29, 37, 57]. Traditional pre-warming mechanisms typically pre-provision multiple microVMs for a user application or function so they can be harvested on demand without incurring a cold start. However, pre-warmed microVMs share a large portion of identical memory content, *e.g.,* the guest OS, language runtime, and loaded libraries. The memory duplication leads to high memory waste at scale [40]. In contrast, each HyperFlux VM primarily scales vertically to absorb increased load, with all computations running inside the same instance and sharing the same memory content. This design alleviates the inefficiencies of memory duplication and reduces reliance on sophisticated techniques such as memory deduplication [54] and sandbox sharing [41].

5 RELATED WORK

Scaling Strategies. Vertical [33, 63] and horizontal [22, 43] core scaling have been explored at coarser time scales (at or beyond the millisecond scale) with container-based provisioning, whereas HyperFlux focuses on VM-based core provisioning at the microsecond scale. Nikolos *et al.* [47] discussed hybrid scaling for VM-based core provisioning, but their vertical scaling relies on statically over-provisioning VMs beforehand to accommodate more computation capacity demands. HyperFlux, in contrast, provides true vertical elasticity that can reclaim and redistribute physical cores among VMs, thereby preventing the excessive reservation of unused CPU resources.

Memory Elasticity. Conventional VM memory elasticity is typically achieved via memory ballooning or hotplug techniques [15, 55, 62]. However, these mechanisms are generally too slow to be viable for serverless platforms, given their stringent latency budgets. A recent study [69] proposes a fast memory reclamation mechanism tailored for serverless platforms. HyperFlux primarily addresses CPU elasticity rather than memory elasticity. Nevertheless, memory elasticity is an orthogonal concern, and we anticipate that achieving VM-based memory elasticity at the microsecond scale represents another important challenge for future serverless computing.

ACKNOWLEDGMENTS

We thank our anonymous reviewers and members of the NSL group at USC for their valuable discussions and feedback. This work was partially supported by Amazon.

REFERENCES

[1] AMD Secure Encrypted Virtualization (SEV). https://www.amd.com/en/developer/sev.html

[2] AWS EMR Serverless. https://aws.amazon.com/emr/serverless.

[3] AWS Nitro System. https://aws.amazon.com/ec2/nitro/

[4] Azure Confidential VM. https://learn.microsoft.com/en-us/azure/confidential-computing/virtual-machine-options

[5] Confidential Containers. https://confidentialcontainers.org/

[6] Confidential Containers on Azure Container Instances. https://learn.microsoft.com/en-us/azure/container-instances/container-instances-confidential-overview

[7] Confidential Containers with Azure Red Hat OpenShift. https://learn.microsoft.com/en-us/azure/openshift/confidential-containers-overview

[8] CPU Hotplug in the Kernel — The Linux Kernel Documentation. https://docs.kernel.org/core-api/cpu_hotplug.html

[9] Databricks: Run your Databricks job with serverless compute for workflows. https://docs.databricks.com/en/workflows/jobs/run-serverless-jobs.html.

[10] GCP: Confidential VM Overview. https://cloud.google.com/confidential-computing/confidential-vm/docs/confidential-vm-overview

[11] GCP: Serverless GPUs on Google Cloud Run. https://cloud.google.com/run/docs/configuring/services/gpu. Accessed: January 2025.

[12] Google Dataproc Serverless Spark. https://cloud.google.com/dataproc-serverless/docs.

[13] gVisor. https://gvisor.dev/

[14] HyperLight: A Lightweight Hypervisor. Microsoft. https://github.com/hyperlight-dev/hyperlight

[15] Memory Hot(Un)Plug — The Linux Kernel Documentation. https://docs.kernel.org/admin-guide/mm/memory-hotplug.html

[16] Redis. https://redis.io/

[17] Databricks: Adaptive Query Execution. https://www.databricks.com/blog/2020/05/29/adaptive-query-execution-speeding-up-spark-sql-at-runtime.html

[18] Alexandru Agache, Marc Brooker, Alexandra Iordache, Anthony Liguori, Rolf Neugebauer, Phil Piwonka, and Diana-Maria Popa. Firecracker: Lightweight Virtualization for Serverless Applications. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*. 419–434. https://www.usenix.org/conference/nsdi20/presentation/agache

[19] Gustavo Alonso, Ana Klimovic, Tom Kuchler, and Michael Wawrzoniak. Rethinking Serverless Computing: From the Programming Model to the Platform Design. In *Proceedings of the Joint Proceedings of Workshops at the 49th International Conference on Very Large Data Bases VLDB 2023 (CEUR Workshop Proceedings, Vol. 3462)*. CEUR-WS.org, 10 p. doi:10.3929/ETHZ-B-000652749

[20] Eric Boutin, Jaliya Ekanayake, Wei Lin, Bing Shi, Jingren Zhou, Zhengping Qian, Ming Wu, and Lidong Zhou. Apollo: Scalable and Coordinated Scheduling for Cloud-Scale Computing. In *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*. 285–300. https://www.usenix.org/conference/osdi14/technical-sessions/presentation/boutin

[21] James Cadden, Thomas Unger, Yara Awad, Han Dong, Orran Krieger, and Jonathan Appavoo. SEUSS: Skip Redundant Paths to Make Serverless Fast. In *Proceedings of the Fifteenth European Conference on Computer Systems (EuroSys '20)*. Association for Computing Machinery, New York, NY, USA, 1–15. doi:10.1145/3342195.3392698

[22] Liao Chen, Shutian Luo, Chenyu Lin, Zizhao Mo, Huanle Xu, Kejiang Ye, and Chengzhong Xu. Derm: SLA-aware Resource Management for Highly Dynamic Microservices. In *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*. 424–436. doi:10.1109/ISCA59077.2024.00039

[23] Alvin Cheung, Natacha Crooks, Joseph M. Hellerstein, and Mae Milano. New Directions in Cloud Programming. In *Proceedings of the 11th Conference on Innovative Data Systems Research*. www.cidrdb.org. http://cidrdb.org/cidr2021/papers/cidr2021_paper16.pdf

[24] Dong Du, Tianyi Yu, Yubin Xia, Binyu Zang, Guanglu Yan, Chenggang Qin, Qixuan Wu, and Haibo Chen. Catalyzer: Sub-millisecond Startup for Serverless Computing with Initialization-less Booting. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '20)*. Association for Computing Machinery, New York, NY, USA, 467–481. doi:10.1145/3373376.3378512

[25] D. R. Engler, M. F. Kaashoek, and J. O'Toole. Exokernel: An Operating System Architecture for Application-Level Resource Management. *ACM SIGOPS Operating Systems Review* 29, 5 (Dec. 1995), 251–266. doi:10.1145/224057.224076

[26] Joshua Fried, Gohar Irfan Chaudhry, Enrique Saurez, Esha Choukse, Inigo Goiri, Sameh Elnikety, Rodrigo Fonseca, and Adam Belay. Making Kernel Bypass Practical for the Cloud with Junction. In *Proceedings of the 21st USENIX Symposium on Networked Systems Design and Implementation*. USENIX Association, 55–73. https://www.usenix.org/conference/nsdi24/presentation/fried

[27] Joshua Fried, Zhenyuan Ruan, Amy Ousterhout, and Adam Belay. Caladan: Mitigating Interference at Microsecond Timescales. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. 281–297. https://www.usenix.org/conference/osdi20/presentation/fried

[28] Alexander Fuerst, Stanko Novaković, Íñigo Goiri, Gohar Irfan Chaudhry, Prateek Sharma, Kapil Arya, Kevin Broas, Eugene Bak, Mehmet Iyigun, and Ricardo Bianchini. Memory-Harvesting VMs in Cloud Platforms. In *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '22)*. Association for Computing Machinery, New York, NY, USA, 583–594. doi:10.1145/3503222.3507725

[29] Alexander Fuerst and Prateek Sharma. FaasCache: Keeping Serverless Computing Alive with Greedy-Dual Caching. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '21)*. Association for Computing Machinery, New York, NY, USA, 386–400. doi:10.1145/3445814.3446757

[30] Robert Grandl, Mosharaf Chowdhury, Aditya Akella, and Ganesh Ananthanarayanan. Altruistic Scheduling in Multi-Resource Clusters. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. 65–80. https://www.usenix.org/conference/osdi16/technical-sessions/presentation/grandl_altruistic

[31] Md E. Haque, Yong hun Eom, Yuxiong He, Sameh Elnikety, Ricardo Bianchini, and Kathryn S. McKinley. Few-to-Many: Incremental Parallelism for Reducing Tail Latency in Interactive Services. In *Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '15)*. Association for Computing Machinery, New York, NY, USA, 161–175. doi:10.1145/2694344.2694384

[32] Benjamin Holmes, Jason Waterman, and Dan Williams. SEVeriFast: Minimizing the Root of Trust for Fast Startup of SEV microVMs. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2 (ASPLOS '24, Vol. 2)*. Association for Computing Machinery, New York, NY, USA, 1045–1060. doi:10.1145/3620665.3640424

[33] Xiaofeng Hou, Chao Li, Jiacheng Liu, Lu Zhang, Shaolei Ren, Jingwen Leng, Quan Chen, and Minyi Guo. AlphaR: Learning-Powered Resource Management for Irregular, Dynamic Microservice Graph. In

2021 IEEE International Parallel and Distributed Processing Symposium (IPDPS). 797–806. doi:10.1109/IPDPS49936.2021.00089

[34] Serhii Ivanenko, Carlos Segarra, and Rodrigo Bruno. Mosaic: Optimizing Cloud Resource Efficiency with Lazily-Packaged Application Modules. In *Proceedings of the 3rd Workshop on SErverless Systems, Applications and MEthodologies*. ACM, Rotterdam Netherlands, 21–29. doi:10.1145/3721465.3721864

[35] Artjom Joosen, Ahmed Hassan, Martin Asenov, Rajkarn Singh, Luke Darlow, Jianfeng Wang, Qiwen Deng, and Adam Barker. Serverless Cold Starts and Where to Find Them. In *Proceedings of the Twentieth European Conference on Computer Systems (EuroSys '25)*. Association for Computing Machinery, New York, NY, USA, 938–953. doi:10.1145/3689031.3696073

[36] Kostis Kaffes, Timothy Chong, Jack Tigar Humphries, Adam Belay, David Mazières, and Christos Kozyrakis. Shinjuku: Preemptive Scheduling for μsecond-Scale Tail Latency. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*. 345–360. https://www.usenix.org/conference/nsdi19/presentation/kaffes

[37] Ricardo Koller and Dan Williams. Will Serverless End the Dominance of Linux in the Cloud?. In *Proceedings of the 16th Workshop on Hot Topics in Operating Systems (HotOS '17)*. Association for Computing Machinery, New York, NY, USA, 169–173. doi:10.1145/3102980.3103008

[38] Tom Kuchler, Pinghe Li, Yazhuo Zhang, Lazar Cvetković, Boris Goranov, Tobias Stocker, Leon Thomm, Simone Kalbermatter, Tim Notter, Andrea Lattuada, and Ana Klimovic. Unlocking True Elasticity for the Cloud-Native Era with Dandelion. arXiv:2505.01603 [cs.DC] https://arxiv.org/abs/2505.01603

[39] Simon Kuenzer, Vlad-Andrei Bădoiu, Hugo Lefeuvre, Sharan Santhanam, Alexander Jung, Gaulthier Gain, Cyril Soldani, Costin Lupu, Ştefan Teodorescu, Costi Răducanu, Cristian Banu, Laurent Mathy, Răzvan Deaconescu, Costin Raiciu, and Felipe Huici. Unikraft: Fast, Specialized Unikernels the Easy Way. In *Proceedings of the Sixteenth European Conference on Computer Systems (EuroSys '21)*. Association for Computing Machinery, New York, NY, USA, 376–394. doi:10.1145/3447786.3456248

[40] Zijun Li, Jiagan Cheng, Quan Chen, Eryu Guan, Zizheng Bian, Yi Tao, Bin Zha, Qiang Wang, Weidong Han, and Minyi Guo. {RunD}: A Lightweight Secure Container Runtime for High-density Deployment and High-concurrency Startup in Serverless Computing. In *2022 USENIX Annual Technical Conference (USENIX ATC 22)*. 53–68. https://www.usenix.org/conference/atc22/presentation/li-zijun-rund

[41] Zijun Li, Linsong Guo, Quan Chen, Jiagan Cheng, Chuhao Xu, Deze Zeng, Zhuo Song, Tao Ma, Yong Yang, Chao Li, and Minyi Guo. Help Rather Than Recycle: Alleviating Cold Startup in Serverless Computing Through Inter-Function Container Sharing. In *2022 USENIX Annual Technical Conference (USENIX ATC 22)*. 69–84. https://www.usenix.org/conference/atc22/presentation/li-zijun-help

[42] Xuanzhe Liu, Jinfeng Wen, Zhenpeng Chen, Ding Li, Junkai Chen, Yi Liu, Haoyu Wang, and Xin Jin. *FaaSLight* : General Application-level Cold-start Latency Optimization for Function-as-a-Service in Serverless Computing. *ACM Transactions on Software Engineering and Methodology* 32, 5 (Sept. 2023), 1–29. doi:10.1145/3585007

[43] Shutian Luo, Huanle Xu, Kejiang Ye, Guoyao Xu, Liping Zhang, Jian He, Guodong Yang, and Chengzhong Xu. Erms: Efficient Resource Management for Shared Microservices with SLA Guarantees. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1 (ASPLOS 2023)*. Association for Computing Machinery, New York, NY, USA, 62–77. doi:10.1145/3567955.3567964

[44] Anil Madhavapeddy and David J. Scott. Unikernels: Rise of the Virtual Library Operating System. *Queue* 11, 11 (Nov. 2013), 30–44. doi:10.1145/2557963.2566628

[45] Filipe Manco, Costin Lupu, Florian Schmidt, Jose Mendes, Simon Kuenzer, Sumit Sati, Kenichi Yasukata, Costin Raiciu, and Felipe Huici. My VM Is Lighter (and Safer) than Your Container. In *Proceedings of the 26th Symposium on Operating Systems Principles (SOSP '17)*. Association for Computing Machinery, New York, NY, USA, 218–233. doi:10.1145/3132747.3132763

[46] Tianxiang Miao and Haibo Chen. FlexCore: Dynamic Virtual Machine Scheduling Using VCPU Ballooning. *Tsinghua Science and Technology* 20, 1 (Feb. 2015), 7–16. doi:10.1109/TST.2015.7040515

[47] Orestis Lagkas Nikolos, Chloe Alverti, Stratos Psomadakis, Georgios Goumas, and Nectarios Koziris. Scaling Serverless Functions: Horizontal or Vertical? Both!. In *Proceedings of the 3rd Workshop on SErverless Systems, Applications and MEthodologies*. ACM, Rotterdam Netherlands, 30–32. doi:10.1145/3721465.3721865

[48] Edward Oakes, Leon Yang, Dennis Zhou, Kevin Houck, Tyler Harter, Andrea Arpaci-Dusseau, and Remzi Arpaci-Dusseau. SOCK: Rapid Task Provisioning with Serverless-Optimized Containers. In *2018 USENIX Annual Technical Conference (USENIX ATC 18)*. 57–70. https://www.usenix.org/conference/atc18/presentation/oakes

[49] Amy Ousterhout, Joshua Fried, Jonathan Behrens, Adam Belay, and Hari Balakrishnan. Shenango: Achieving High CPU Efficiency for Latency-sensitive Datacenter Workloads. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*. 361–378. https://www.usenix.org/conference/nsdi19/presentation/ousterhout

[50] Donald E. Porter, Silas Boyd-Wickizer, Jon Howell, Reuben Olinsky, and Galen C. Hunt. Rethinking the Library OS from the Top Down. *ACM SIGPLAN Notices* 46, 3 (March 2011), 291–304. doi:10.1145/1961296.1950399

[51] George Prekas, Marios Kogias, and Edouard Bugnion. ZygOS: Achieving Low Tail Latency for Microsecond-scale Networked Tasks. In *Proceedings of the 26th Symposium on Operating Systems Principles (SOSP '17)*. Association for Computing Machinery, New York, NY, USA, 325–341. doi:10.1145/3132747.3132780

[52] Henry Qin, Qian Li, Jacqueline Speiser, Peter Kraft, and John Ousterhout. Arachne: Core-Aware Thread Management. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. 145–160. https://www.usenix.org/conference/osdi18/presentation/qin

[53] Alireza Sahraei, Soteris Demetriou, Amirali Sobhgol, Haoran Zhang, Abhigna Nagaraja, Neeraj Pathak, Girish Joshi, Carla Souza, Bo Huang, Wyatt Cook, Andrii Golovei, Pradeep Venkat, Andrew Mcfague, Dimitrios Skarlatos, Vipul Patel, Ravinder Thind, Ernesto Gonzalez, Yun Jin, and Chunqiang Tang. XFaaS: Hyperscale and Low Cost Serverless Functions at Meta. In *Proceedings of the 29th Symposium on Operating Systems Principles (SOSP '23)*. Association for Computing Machinery, New York, NY, USA, 231–246. doi:10.1145/3600006.3613155

[54] Divyanshu Saxena, Tao Ji, Arjun Singhvi, Junaid Khalid, and Aditya Akella. Memory Deduplication for Serverless Computing with Medes. In *Proceedings of the Seventeenth European Conference on Computer Systems (EuroSys '22)*. Association for Computing Machinery, New York, NY, USA, 714–729. doi:10.1145/3492321.3524272

[55] Joel Schopp and Keir Fraser. Resizing Memory With Balloons and Hotplug. In *Proceedings of the Linux Symposium*, Vol. 2. Ottawa, Ontario, Canada, 313. https://kernel.org/doc/mirror/ols2006v2.pdf#page=313

[56] Malte Schwarzkopf, Andy Konwinski, Michael Abd-El-Malek, and John Wilkes. Omega: Flexible, Scalable Schedulers for Large Compute Clusters. In *Proceedings of the 8th ACM European Conference on Computer Systems (EuroSys '13)*. Association for Computing Machinery, New York, NY, USA, 351–364. doi:10.1145/2465351.2465386

[57] Mohammad Shahrad, Rodrigo Fonseca, Íñigo Goiri, Gohar Chaudhry, Paul Batum, Jason Cooke, Eduardo Laureano, Colby Tresness, Mark Russinovich, and Ricardo Bianchini. Serverless in the Wild: Characterizing and Optimizing the Serverless Workload at a Large Cloud

Provider. In *2020 USENIX Annual Technical Conference (USENIX ATC 20)*. 205–218. https://www.usenix.org/conference/atc20/presentation/shahrad

[58] Jiacheng Shi, Jinyu Gu, Yubin Xia, and Haibo Chen. Serverless Functions Made Confidential and Efficient with Split Containers. In *34th USENIX Security Symposium (USENIX Security 25)*. 1091–1110. https://www.usenix.org/conference/usenixsecurity25/presentation/shi-jiacheng

[59] Simon Shillaker and Peter Pietzuch. Faasm: Lightweight Isolation for Efficient Stateful Serverless Computing. In *2020 USENIX Annual Technical Conference (USENIX ATC 20)*. 419–433. https://www.usenix.org/conference/atc20/presentation/shillaker

[60] Ariel Szekely, Adam Belay, Robert Morris, and M. Frans Kaashoek. Unifying Serverless and Microservice Workloads with SigmaOS. In *Proceedings of the ACM SIGOPS 30th Symposium on Operating Systems Principles (SOSP '24)*. Association for Computing Machinery, New York, NY, USA, 385–402. doi:10.1145/3694715.3695947

[61] Abhishek Verma, Luis Pedrosa, Madhukar Korupolu, David Oppenheimer, Eric Tune, and John Wilkes. Large-Scale Cluster Management at Google with Borg. In *Proceedings of the Tenth European Conference on Computer Systems (EuroSys '15)*. Association for Computing Machinery, New York, NY, USA, 1–17. doi:10.1145/2741948.2741964

[62] Carl A. Waldspurger. Memory Resource Management in VMware ESX Server. *SIGOPS Oper. Syst. Rev.* 36, SI (Dec. 2003), 181–194. doi:10.1145/844128.844146

[63] Zibo Wang, Pinghe Li, Chieh-Jan Mike Liang, Feng Wu, and Francis Y. Yan. Autothrottle: A Practical Bi-Level Approach to Resource Management for SLO-Targeted Microservices. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*. 149–165. https://www.usenix.org/conference/nsdi24/presentation/wang-zibo

[64] Nicholas C. Wanninger, Joshua J. Bowden, Kirtankumar Shetty, Ayush Garg, and Kyle C. Hale. Isolating Functions at the Hardware Limit with Virtines. In *Proceedings of the Seventeenth European Conference on Computer Systems (EuroSys '22)*. ACM, Rennes France, 644–662. doi:10.1145/3492321.3519553

[65] Xingda Wei, Fangming Lu, Tianxia Wang, Jinyu Gu, Yuhan Yang, Rong Chen, and Haibo Chen. No Provisioned Concurrency: Fast RDMA-codesigned Remote Fork for Serverless Computing. In *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23)*. 497–517. https://www.usenix.org/conference/osdi23/presentation/wei-rdma

[66] Xi Yang, Stephen M. Blackburn, and Kathryn S. McKinley. Elfen Scheduling: Fine-Grain Principled Borrowing from Latency-Critical Workloads Using Simultaneous Multithreading. In *2016 USENIX Annual Technical Conference (USENIX ATC 16)*. 309–322. https://www.usenix.org/conference/atc16/technical-sessions/presentation/yang

[67] Jianing You, Kang Chen, Laiping Zhao, Yiming Li, Yichi Chen, Yuxuan Du, Yanjie Wang, Luhang Wen, Keyang Hu, and Keqiu Li. AlloyStack: A Library Operating System for Serverless Workflow Applications. In *Proceedings of the Twentieth European Conference on Computer Systems (EuroSys '25)*. Association for Computing Machinery, New York, NY, USA, 921–937. doi:10.1145/3689031.3717490

[68] Minchen Yu, Tingjia Cao, Wei Wang, and Ruichuan Chen. Following the Data, Not the Function: Rethinking Function Orchestration in Serverless Computing. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. 1489–1504. https://www.usenix.org/conference/nsdi23/presentation/yu

[69] Xinmin Zhang, Qiang He, Hao Fan, and Song Wu. Faascale: Scaling MicroVM Vertically for Serverless Computing with Memory Elasticity. In *Proceedings of the ACM Symposium on Cloud Computing (SoCC '24)*. ACM, Redmond WA USA, 196–212. doi:10.1145/3698038.3698512

[70] Xiao Zhang, Eric Tune, Robert Hagmann, Rohit Jnagal, Vrigo Gokhale, and John Wilkes. CPI^2: CPU Performance Isolation for Shared Compute Clusters. In *Proceedings of the 8th ACM European Conference on Computer Systems*. ACM, Prague Czech Republic, 379–391. doi:10.1145/2465351.2465388

[71] Shixuan Zhao, Mengyuan Li, Mengjia Yan, and Zhiqiang Lin. Ditto: Elastic Confidential VMs with Secure and Dynamic CPU Scaling. doi:10.48550/arXiv.2409.15542

[72] Zhuangzhuang Zhou, Yanqi Zhang, and Christina Delimitrou. AQUATOPE: QoS-and-Uncertainty-Aware Resource Management for Multi-stage Serverless Workflows. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1 (ASPLOS 2023)*. Association for Computing Machinery, New York, NY, USA, 1–14. doi:10.1145/3567955.3567960