Are Edge MicroDCs Equipped to Tackle Memory Contention?

Long Tran
Rutgers University

River Bartz
University of Oregon

Ramakrishnan Durairajan
University of Oregon

Ulrich Kremer
Rutgers University

Sudarsun Kannan
Rutgers University

Abstract

Hazard monitoring systems rely on micro datacenters (MicroDCs) for local data processing and real-time response in resource- and energy-constrained environments. These MicroDCs often host diverse, multitenant applications—such as object detection and sensor data ingestion—that contend for shared memory. Through a case study of compute-intensive and I/O-intensive applications, we show that different applications use memory differently (e.g., heap vs. OS-managed page cache), leading to asymmetric performance degradation under memory pressure. Our findings highlight the limitations of existing OS-level resource management approaches and motivate the need for cross-layered coordination between applications and the operating system to treat all memory uses as first-class citizens and adapt to changing workload demands in MicroDCs.

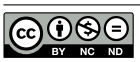
CCS Concepts

• **Software and its engineering** → **Main memory**; **Allocation / deallocation strategies**; **Operating systems**; • **Hardware** → *Energy metering*.

Keywords

Edge computing, Multitenancy, DRAM, Operating Systems, Machine Learning

ACM Reference Format:
Long Tran, River Bartz, Ramakrishnan Durairajan, Ulrich Kremer, and Sudarsun Kannan. 2025. Are Edge MicroDCs Equipped to Tackle Memory Contention?. In *17th ACM Workshop on Hot Topics in Storage and File Systems (HotStorage '25), July 10–11, 2025, Boston, MA, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3736548.3737827



HotStorage '25, July 10–11, 2025, Boston, MA, USA

ACM ISBN 979-8-4007-1947-9/2025/07
https://doi.org/10.1145/3736548.3737827

1 Introduction

Hazard monitoring systems are critical to the early detection and response to natural disasters, helping safeguard lives and infrastructure. These systems are composed of diverse environmental sensors—ranging from wildfire detection cameras to weather and seismic monitors—connected to distributed computing nodes called micro datacenters (MicroDCs). These MicroDCs are designed for near-sensor processing, enabling real-time analytics and low-latency during emergencies.

MicroDCs are typically small-scale compute clusters, deployed remotely and often constrained by limited power, cooling, and hardware availability. Their configurations range from CPU-only systems to those equipped with GPUs to support workloads like image recognition and data-intensive logging. These systems must run multiple applications concurrently ranging from ML inference for wildfire detection (e.g., YOLO [21]) to data ingestion and storage (e.g., RocksDB [3]) while minimizing latency and energy usage. As such, multitenancy is not a design choice but a requirement, driven by the need to support heterogeneous tasks on limited infrastructure.

Unlike cloud-scale data centers, MicroDCs cannot rely on resource overprovisioning or cross-node scheduling to balance workloads. They are often oversubscribed: resource demands from applications consistently exceed the available CPU, GPU, and memory capacity. This oversubscription, coupled with highly dynamic workloads (e.g., sudden spikes in video uploads during wildfire events), creates acute pressure on shared resources—particularly memory.

Memory is a critical shared resource across applications in hazard monitoring MicroDCs. It serves GPU-based applications like YOLO (for heap-allocated batch inference), CPU-bound applications like RocksDB (which rely heavily on page cache), and communication-intensive services that buffer data in memory before sending it to the cloud. These diverse and conflicting usage patterns make static partitioning approaches—such as those enforced by virtual machines or containers—ineffective. Static limits often lead to underutilization during periods of low activity and critical contention during bursts, degrading performance and increasing energy usage.

Consider current deployments such as the SAGE Continuum [4, 5], where MicroDCs support various tenants engaged in real-time hazard analysis. During calm periods, system activity remains moderate; however, when environmental events are detected, applications may rapidly enter high-throughput modes, requiring more compute and memory. Static memory allocation in these scenarios often fails to respond in time, resulting in application interference, degraded performance, and data loss at the sensor edge due to backpressure.

Prior efforts to improve resource sharing in traditional data centers (e.g., PARTIES [6], AWARE [20]) rely on continuous monitoring and resource availability, i.e., conditions that MicroDCs often cannot guarantee. Moreover, these systems treat memory regions differently (e.g., heap vs. cache vs. buffer), lacking coordination across OS layers and application phases.

In this position paper, we argue that efficient and adaptive memory sharing in MicroDCs requires a principled, cross-layer design that integrates application-level awareness with OS-level memory policies. We propose **PAMS**, a *Performance and Energy Adaptive Multi-Tenant Resource Management Framework*, that goes beyond static knobs and isolation boundaries. PAMS introduces three core design principles: (1) equal treatment of diverse memory types depending on application phase and system state, (2) application-aware adaptation to react to workload-specific triggers such as sensor surges during wildfire detection, and (3) intelligent resource allocation using predictive models and energy-aware scheduling.

To motivate our design, we study two representative applications (YOLO for wildfire detection and RocksDB for environmental data logging) co-running under multitenancy in a MicroDC. We show that traditional container-based partitioning leads to performance and energy inefficiencies. Our results highlight the nonlinear tradeoffs between memory allocation, application performance, and energy consumption—underscoring the need for dynamic, context-aware memory sharing strategies tailored to the constraints of MicroDCs.

2 Background and Related Work

2.1 Resource Sharing in MicroDCs

In MicroDCs deployed for hazard monitoring, applications are often connected through a network of diverse sensors to a shared local infrastructure. These applications, running concurrently, demand efficient multitenancy support across shared hardware components, especially memory, compute, storage, and I/O subsystems. As MicroDCs are tasked with executing real-time detection, data logging, compression, communication, and decision-making, the need to dynamically manage these resources becomes critical.

In this position paper, we focus on *memory* as a first-class resource for multitenancy due to its broad and essential role across multiple application domains. Memory is a shared and critical component for GPU-based applications such as YOLO (for wildfire detection), CPU-bound applications like NoSQL databases, and communication-intensive applications that interact frequently with remote cloud services or other MicroDCs. These workloads exhibit differing usage patterns (heap-intensive computation, cache-driven I/O access, and buffer-based communication), all contending for limited physical memory.

While compute isolation is generally well-managed, memory isolation remains a significant challenge. Containerization, although widely adopted, does not suffice in MicroDCs due to the diverse and fluctuating memory demands of colocated applications. Static memory partitioning fails to capture dynamic behavior, leading to underutilization or contention at runtime.

Further complicating memory management is the system-wide role of shared OS-level memory, including page caches, driver allocations, and kernel data structures. These memory regions, although managed by the OS, are often deprioritized under pressure, negatively impacting application performance.

A natural solution is to expand memory capacity, such as by incrementally adding dual in-line memory modules (DIMMs) over time, particularly when weighed against the potential cost of failing to mitigate a wildfire. However, we argue that this approach is not scalable for MicroDC deployments. These systems typically operate with tight energy budgets, often relying on solar power or intermittent grid access. Increasing physical memory not only raises operational energy consumption but also contributes to the embodied carbon footprint—both of which run counter to the sustainability objectives of hazard-monitoring infrastructure [13, 15]. Additionally, cost constraints in deploying large-scale MicroDC networks make provisioning excess memory impractical.

Thus, this paper argues for a dynamic, adaptive memory-sharing mechanism that treats all types of memory use—heap, cache, buffer—as equally important depending on workload context. Efficient memory sharing is not only essential for maintaining performance but also for achieving energy and carbon efficiency in MicroDCs. This calls for principled system support that can reason across layers -from the intent of the application to the OS's memory policies, while responding to workload dynamics and environmental constraints.

2.2 Related Work

A huge body of prior work have proposed techniques to improve multitenant resource management in traditional data centers and containerized environments. However, few directly address the constraints and dynamics faced by MicroDCs.

Multitenant Resource Management in Data Centers: Systems such as PARTIES [6] and Heracles [16] target performance isolation in large-scale servers by leveraging offline profiling and online, feedback-based control. While effective in stable, high-capacity environments, they assume predictable resource availability and do not address energy or carbon constraints. These systems also lack consideration in tolerating multiple latency-critical (LC) tasks with best-effort (BE) tasks, such as high throughput tasks, simultaneously. Heracles only enables co-running 1 LC task with many BE tasks. Meanwhile, PARTIES only focuses on allowing multiple LC tasks. These systems focus on completing individual tasks in large-scale servers where microservices are highly independent of each other. On the other hand, MicroDCs workloads are dynamic, requiring both types of LC task and BE tasks to co-exist and to achieve the goal. In addition, there are many approaches for automated resource management based on reinforcement learning (RL) [14, 18–20, 22, 23, 25]. For example, AWARE [20] focuses on adaptive memory management in containerized clusters using memory access profiling and online RL. These systems rely on extensive monitoring and resource availability, which is often impractical in resource-constrained MicroDCs.

Memory Management and Co-scheduling: OS-level policies for page cache and heap prioritization have been studied extensively. For example, MemVerge [2] and ElastiCOS [9] offer strategies for dynamic memory movement or disaggregation, while Dyno [1] and TMO [24] tackle memory prioritization and task migration in multitenant setups. However, these systems often do not provide mechanisms to consider different types of memory usage (heap, cache, buffer) as equal citizens, nor are they designed with energy efficiency and cross-layer cooperation in mind.

MicroDC and Edge Resource Management: Several recent works have explored resource allocation in edge and MicroDC environments. Projects such as Seer [12] and EdgeBench [8] explore dynamic scheduling and performance modeling at the edge. While these systems highlight the variability of workloads and constraints in edge computing, they do not propose cross-layer mechanisms that reason across application states and OS-level memory pressure.

Energy and Sustainability-Aware Systems: Work on energy-aware scheduling (e.g., Pegasus [11]) and carbon-aware infrastructure (e.g., ZeroCarbonCloud [7], CO2-aware scheduling) provides critical background for designing sustainable systems. However, these solutions focus on macro-level power capping or inter-DC scheduling and do not address fine-grained memory sharing in power-constrained MicroDC deployments.

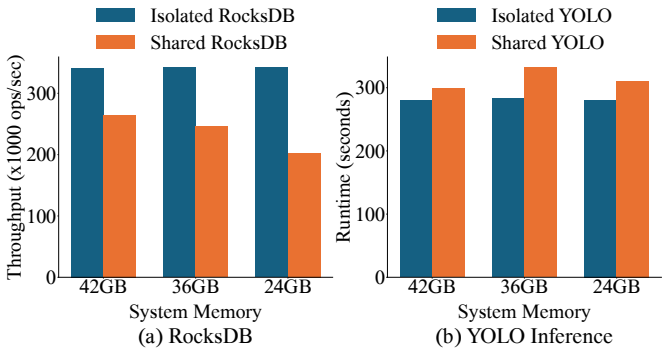


Figure 1: System resource sharing across (a) RocksDB and (b) YOLO ML inference performance on baremetal system. *Isolated* indicates running applications without resource sharing; *Shared* denotes sharing CPU and memory. The x-axis incrementally reduces memory.

Our proposed design, **PAMS**, complements these efforts by proposing a framework that unifies diverse memory abstractions, integrates application-aware triggers, and optimizes resource allocation using predictive models, all tailored to the constraints and variability of MicroDCs used in hazard monitoring.

2.3 Case Study and Analysis

Our case study evaluates the impact of multitenancy in MicroDC environments using two representative applications: YOLO [21], a widely-used ML-based computer vision framework, and RocksDB, a reliable NoSQL key-value store. These applications reflect real-world deployments in wildfire hazard monitoring MicroDCs. YOLO, built on PyTorch [17], performs real-time inference over wildfire camera data using a convolutional neural network (CNN). Multiple YOLO processes run in parallel, each handling image batches that fit into available GPU memory. RocksDB manages large volumes of sensor data, including wildfire imagery and logs from weather stations, wildlife monitors, and temperature sensors.

Our MicroDC experimental setup consists of a single-socket system with 48GB DRAM, a 16-core 2.1GHz Intel(R) Xeon(R) Silver 4110 processor, a 512GB NVMe SSD, and a Nvidia Quadro P5000 16GB GPU. The system runs Linux 5.15 and suppports both training and inference pipelines for YOLO. We explore three primary objectives: (1) analyzing YOLO's inference and RocksDB's performance under resource contention, (2) evaluating training-time multitenancy effects, and (3) quantifying energy and carbon implications.

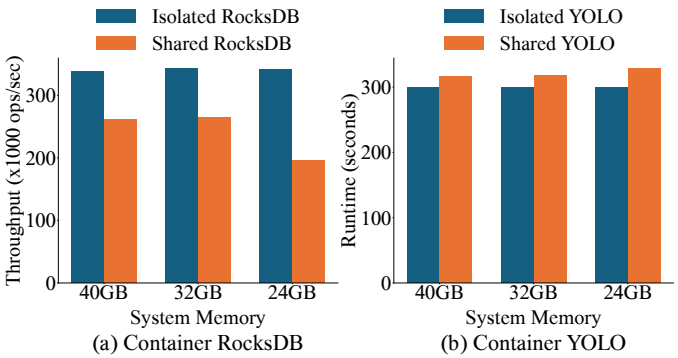


Figure 2: Containerized applications (a) RocksDB and (b) YOLO ML inference performance. *Isolated* shows running applications without resource sharing; *Shared* shows sharing CPU and memory; x-axis incrementally reduces memory.

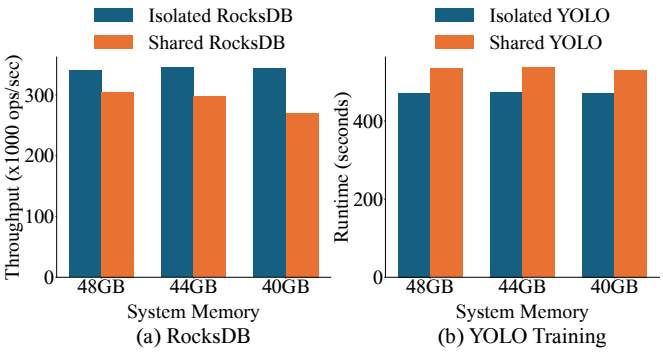


Figure 3: Analysis of system resource sharing across (a) RocksDB and (b) YOLO ML training performance. *Isolated* indicates running applications without resource sharing; *Shared* denotes sharing CPU and memory. The x-axis incrementally reduces memory.

(1) Multitenancy Performance for Inference: Figure 1 shows the inference performance of YOLO and RocksDB under shared memory conditions on a bare-metal system. As memory availability drops, RocksDB's throughput degrades sharply (over 40% at 24GB) due to its reliance on OS-level page cache, which is deprioritized under pressure. In contrast, YOLO, with heap-reserved memory, sees less than 10% degradation. These results highlight the need to treat I/O-intensive applications and their memory use (such as page caches) as first-class resources in memory-constrained MicroDCs.

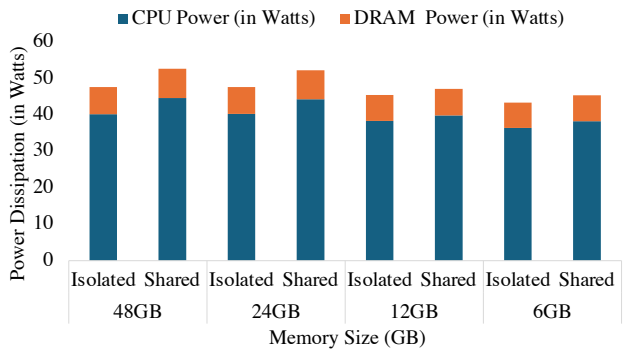
(2) Containerized Inference Under Static Partitioning: Figure 2 compares the performance of containerized YOLO and RocksDB applications under isolated and shared execution across different memory capacities. In the isolated setup, each application runs independently within its own container, while in the shared configuration, both containers run concurrently, sharing CPU and memory. Based on profiling, we statically determined that YOLO requires 8GB of memory to maintain its best isolated performance. Accordingly, we assign 8GB to the YOLO container and allocate the remaining memory to RocksDB. As system memory is reduced from 40GB to 24GB, YOLO inference latency increases modestly by just over 10% while RocksDB throughput suffers a more substantial drop of nearly 40% under shared conditions. These results show that static partitioning (e.g., even with container-based isolation) cannot adequately prevent memory contention, particularly for I/O-intensive applications like RocksDB.

(3) Multitenancy Performance for Training: Figure 3 compares the training performance of YOLO and RocksDB under shared and isolated execution as system memory is reduced from 48GB to 40GB. RocksDB's throughput degrades by up to 20% under shared conditions, while YOLO's training runtime increases by approximately 15%. Unlike inference, the performance degradation is more closely aligned across both applications. This is because YOLO training involves reading large batches of images per iteration, making it partially reliant on the OS-level I/O cache. Consequently, both applications compete for cache-backed memory resources, which disproportionately impacts RocksDB due to its fully I/O-bound nature.

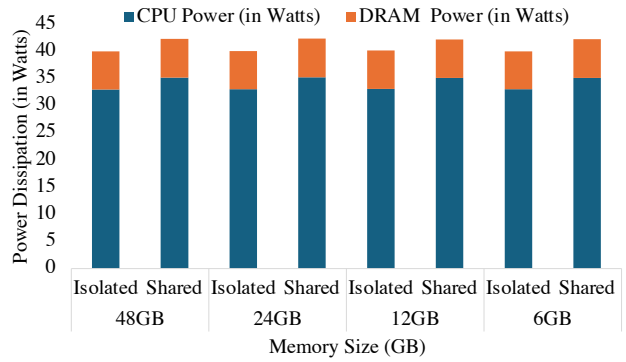
(4) Energy and Carbon Impact of CPU and Memory Sharing: Figure 4 presents power and carbon analysis when YOLO and RocksDB co-run, focusing on CPU and DRAM. We measure power using Intel RAPL counters and calculate carbon emissions using CPU and DRAM intensity factors from [10]. GPU energy and carbon are excluded, as our analysis centers on host-level resource sharing between the two applications.

Panels (a) and (b) show power breakdowns when RocksDB and YOLO are co-executed. Despite co-running, power increases are modest (<6%) due to overlapping idle/active cycles and baseline system activity from daemons. Panel (c) reports total carbon emissions. For the isolated case, we measure YOLO and RocksDB independently and sum their emissions; for shared runs, we measure joint execution. Notably, multitenancy reduces total carbon emissions compared to isolated runs—by nearly 50% in some configurations. This reduction comes from better utilization of static power and amortization of embodied system costs, reinforcing the sustainability part of efficient resource sharing in MicroDCs.

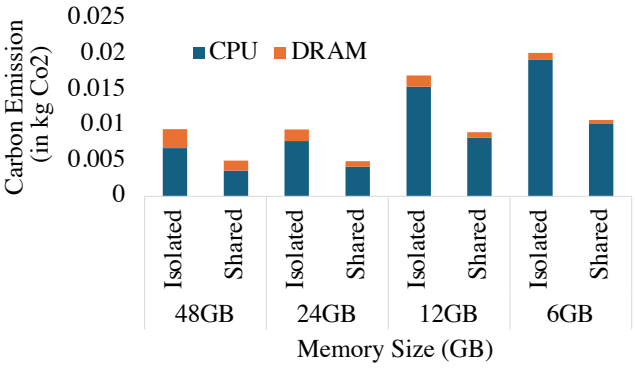
In summary, the results underscore the importance of treating cache-backed I/O and heap memory as first-class, application-aware resources when managing memory under multitenancy.



(a) RocksDB power co-run with YOLO .



(b) YOLO power co-run with RocksDB.



(c) Total Carbon Emission.

Figure 4: Power and carbon analysis from resource sharing between YOLO ML training and RocksDB. For the carbon analysis in 4c, the isolated baseline shows the sum of the carbon footprints of RocksDB and YOLO. We use the CPU and DRAM carbon footprint values from [10].

3 Proposed Design

To address the challenges of multitenant resource management in MicroDCs for hazard monitoring systems, we propose **PAMS**, a Performance and Energy Adaptive Multitenant Resource Management Framework. The primary objective of PAMS is to establish a principled, cross-layered design that bridges application-level characteristics with operating system-level resource management to optimize performance, energy efficiency, and sustainability under constrained and dynamically changing conditions.

Unified Treatment of Memory Type: MicroDCs host diverse applications that rely on different types of memory: RocksDB benefits from page cache to optimize I/O throughput, YOLO uses heap memory for continuous inference, and communication stacks depend on kernel and network buffers. However, current OS memory managers treat page cache as a second-class citizen compared to heap memory, leading to unpredictable eviction and performance loss for I/O-intensive tasks.

PAMS introduces a unified memory classification model that dynamically promotes different memory regions (e.g., page cache, heap, network buffers) as “first-class" citizens based on active workload characteristics. Implementation involves extending OS-level memory tracking (e.g., using cgroup-level memory pressure and page fault statistics) and associating priority labels with virtual memory regions. These priorities inform eviction and reclamation policies during memory contention, ensuring that memory critical to active workloads is retained.

Application-Level Triggers: MicroDC workloads, especially those for hazard monitoring, exhibit time-varying criticality. For instance, YOLO continuously processes imagery but requires urgent resource boosts when wildfire-like patterns are detected. Similarly, databases may operate in low-activity logging mode but must handle bursts of data ingestion or queries during an emergency. PAMS aims to integrate hooks and telemetry at the application level to detect such transitions. Triggers include internal signals (e.g., classification confidence in YOLO, ingestion rate in RocksDB) and system events (e.g., user annotations or external alerts). Applications communicate their state transitions to the OS through a shared trigger interface (e.g., via extended cgroups or sysfs signals). The OS scheduler responds with reconfiguration, for example, increasing CPU shares, increasing cache retention, tuning swappiness, or migrating tasks to higher-performance cores.

Predictive and Adaptive Resource Allocation: In addition to the cross-layered application/runtime and the OS design, we aim to develop an intelligent resource allocation engine that blends predictive modeling with multi-objective optimization. Using historical performance logs and current telemetry, PAMS forecasts future resource demand (e.g., spikes in CPU usage during wildfire detection) and proactively adjusts allocations. It solves a multi-objective problem, balancing performance metrics (e.g., latency, throughput) against energy and carbon constraints. The allocation engine built atop a lightweight machine learning model would

augment with an online reinforcement learning (RL) loop. RL agents are trained to adjust knobs (e.g., memory shares, CPU limits, I/O bandwidth quotas) in response to application and system state. This dynamic feedback loop allows the system to adapt continuously to environmental variability and workload dynamics.

4 Summary and Future Work

This position paper identifies the challenges of multitenant memory sharing in resource- and energy-constrained MicroDCs used for hazard monitoring. Through a case study, we show that diverse memory usage patterns lead to unequal performance degradation under contention, which traditional isolation mechanisms fail to address. Our findings motivate a cross-layered approach where the OS and applications coordinate to manage memory dynamically. Our on-going/future work will focus on lightweight monitoring, energy-aware coordination, and adaptive policies that respond to workload shifts. We will also explore whether PAMS can be implemented in eBPF, Kubelet controller, or user-space daemons without significant changes to the host OS for wider deployment.

Acknowledgments

We thank the anonymous reviewers for their constructive feedback. This work is supported by the National Science Foundation through CNS-1910593, CNS-2231724, CNS-2145813, OAC-2319944, CAREER-2443438, and the Internet Society Foundation (ISOC) grants. This work was carried out on the CloudLab platform. A portion of this work also utilized the experimental platform from Rutgers Panic Lab, funded by NSF II-EN grant 1730043, and the NSF Chameleon Cloud.

References

[1] DYNO. https://dyno.gg/.

[2] MemVerge. https://memverge.com/.

[3] RocksDB. http://rocksdb.org/.

[4] ANL. SAGE continuum project, 2024.

[5] Peter Beckman, Rajesh Sankaran, Charles Catlett, Nicola Ferrier, Robert Jacob, and Michael Papka. Waggle: An open sensor platform for edge computing. pages 1–3, 10 2016.

[6] Shuang Chen, Christina Delimitrou, and José F. Martínez. Parties: Qos-aware resource partitioning for multiple interactive services. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '19, page 107–120, New York, NY, USA, 2019. Association for Computing Machinery.

[7] Andrew A. Chien, Rich Wolski, and Fan Yang. Zero-carbon cloud: A volatile resource for high-performance computing. In *2015 IEEE International Conference on Computer and Information Technology; Ubiquitous Computing and Communications; Dependable, Autonomic and Secure Computing; Pervasive Intelligence and Computing*, pages 1997–2001, 2015.

[8] Anirban Das, Stacy Patterson, and Mike Wittie. Edgebench: Benchmarking edge computing platforms. In *2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion)*, pages 175–180. IEEE, 2018.

[9] Amit Gupta, Ehab Ababneh, Richard Han, and Eric Keller. Towards elastic operating systems. In *14th Workshop on Hot Topics in Operating Systems (HotOS XIV)*, 2013.

[10] Udit Gupta, Mariam Elgamal, Gage Hills, Gu-Yeon Wei, Hsien-Hsin S. Lee, David Brooks, and Carole-Jean Wu. Act: Designing sustainable computer systems with an architectural carbon modeling tool. In *Proceedings of the 49th Annual International Symposium on Computer Architecture*, ISCA '22, page 784–799, New York, NY, USA, 2022. Association for Computing Machinery.

[11] Vishakha Gupta, Karsten Schwan, Niraj Tolia, Vanish Talwar, and Parthasarathy Ranganathan. Pegasus: coordinated scheduling for virtualized accelerator-based systems. In *Proceedings of the 2011 USENIX Conference on USENIX Annual Technical Conference*, USENIXATC'11, page 3, USA, 2011. USENIX Association.

[12] Shaoyuan Huang, Zheng Wang, Zhongtian Zhang, Heng Zhang, Xiaofei Wang, and Wenyu Wang. Seer: Proactive revenue-aware scheduling for live streaming services in crowdsourced cloud-edge platforms. In *IEEE INFOCOM 2024 - IEEE Conference on Computer Communications*, pages 1801–1810, 2024.

[13] Sudarsun Kannan and Ulrich Kremer. Towards application centric carbon emission management. In *Proceedings of the 2nd Workshop on Sustainable Computer Systems*, HotCarbon '23, New York, NY, USA, 2023. Association for Computing Machinery.

[14] Sara Kardani-Moghaddam, Rajkumar Buyya, and Kotagiri Ramamohanarao. Adrl: A hybrid anomaly-aware deep reinforcement learning-based resource scaling in clouds. *IEEE Transactions on Parallel and Distributed Systems*, 32(3):514–526, 2021.

[15] Babar Khalid, Nolan Rudolph, Ramakrishnan Durairajan, and Sudarsun Kannan. Micromon: a monitoring framework for tackling distributed heterogeneity. In *Proceedings of the 12th USENIX Conference on Hot Topics in Storage and File Systems*, HotStorage '20, USA, 2020. USENIX Association.

[16] David Lo, Liqun Cheng, Rama Govindaraju, Parthasarathy Ranganathan, and Christos Kozyrakis. Heracles: improving resource efficiency at scale. In *Proceedings of the 42nd Annual International Symposium on Computer Architecture*, ISCA '15, page 450–462, New York, NY, USA, 2015. Association for Computing Machinery.

[17] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library, 2019.

[18] Haoran Qiu, Subho S. Banerjee, Saurabh Jha, Zbigniew T. Kalbarczyk, and Ravishankar K. Iyer. FIRM: An intelligent fine-grained resource management framework for SLO-Oriented microservices. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pages 805–825. USENIX Association, November 2020.

[19] Haoran Qiu, Weichao Mao, Archit Patke, Chen Wang, Hubertus Franke, Zbigniew T. Kalbarczyk, Tamer Başar, and Ravishankar K. Iyer. Simppo: a scalable and incremental online learning framework for serverless resource management. In *Proceedings of the 13th Symposium on Cloud Computing*, SoCC '22, page 306–322, New York, NY, USA, 2022. Association for Computing Machinery.

[20] Haoran Qiu, Weichao Mao, Chen Wang, Hubertus Franke, Alaa Youssef, Zbigniew T. Kalbarczyk, Tamer Başar, and Ravishankar K. Iyer. AWARE: Automate workload autoscaling with reinforcement learning in production cloud systems. In *2023 USENIX Annual Technical Conference (USENIX ATC 23)*, pages 387–402, Boston, MA, July 2023. USENIX Association.

[21] Joseph Redmon, Santosh Kumar Divvala, Ross B. Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. *CoRR*, abs/1506.02640, 2015.

[22] Fabiana Rossi, Matteo Nardelli, and Valeria Cardellini. Horizontal and vertical scaling of container-based applications using reinforcement learning. In *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*, pages 329–338, 2019.

[23] Ziliang Wang, Shiyi Zhu, Jianguo Li, Wei Jiang, K. K. Ramakrishnan, Yangfei Zheng, Meng Yan, Xiaohong Zhang, and Alex X. Liu. Deepscaling: microservices autoscaling for stable cpu utilization in large scale cloud systems. In *Proceedings of the 13th Symposium on Cloud Computing*, SoCC '22, page 16–30, New York, NY, USA, 2022. Association for Computing Machinery.

[24] Johannes Weiner, Niket Agarwal, Dan Schatzberg, Leon Yang, Hao Wang, Blaise Sanouillet, Bikash Sharma, Tejun Heo, Mayank Jain, Chunqiang Tang, and Dimitrios Skarlatos. Tmo: transparent memory offloading in datacenters. In *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '22, page 609–621, New York, NY, USA, 2022. Association for Computing Machinery.

[25] Zhe Yang, Phuong Nguyen, Haiming Jin, and Klara Nahrstedt. MIRAS: Model-based reinforcement learning for microservice resource allocation over scientific workflows. In *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*, pages 122–132, 2019.