Can OS Specialization give new life to old carbon in the cloud?

Han Dong
Boston University
Boston, USA
handong@bu.edu

Sanjay Arora
Red Hat Inc
New York, USA
saarora@redhat.com

Orran Krieger
Boston University
Boston, USA
okrieg@bu.edu

Jonathan Appavoo
Boston University
Boston, USA
jappavoo@bu.edu

ABSTRACT

Is there "fat" (overheads) in cloud computing infrastructure software that can be trimmed? Would doing so help ameliorate the need for frequent hardware refreshes and extend the life of existing hardware? In this paper, we demonstrate that, indeed, there is "fat" that can be trimmed by using specialized OS-based software stacks. Doing so can allow decade-old computers to be used for critical cloud infrastructure services, potentially yielding 3x improvements in efficiency compared to standard software stacks on newer hardware. The implications of these results raise the possibility of exploiting OS optimizations to reduce server hardware obsolescence. Further, it suggests the importance of addressing the key portability challenges of specialized OS stacks.

CCS CONCEPTS

• **Hardware** → **Power and energy**; • **Computer systems organization**;

KEYWORDS

sustainability, operating systems, measurement

ACM Reference Format:
Han Dong, Sanjay Arora, Orran Krieger, and Jonathan Appavoo. 2024. Can OS Specialization give new life to old carbon in the cloud?. In *The 17th ACM International Systems and Storage Conference (SYSTOR '24), September 23–24, 2024, Virtual, Israel.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3688351.3689158

1 INTRODUCTION

In this paper, we discuss the opportunity to breathe new life into old hardware by using specialized software for data center-scale applications. Doing so can potentially reduce cloud computing providers' embodied and operational carbon footprints. However, to realize this potential, we must address some challenges and open problems.

We present data suggesting that using specialized software stacks for dedicated services can enable performance improvements and energy savings by over 3X on older hardware. These improvements make it viable to use older hardware for performance critical services and help reduce hardware obsolescence by prolonging the life of purchase hardware within data centers. A data center operator can then squeeze more value out of their investments by judiciously using their modern higher-performance hardware.

1.1 Background

A hallmark of modern cloud computing are the fleets of servers that run core infrastructure services such as in-memory caches and databases [17]. While not glamorous, these services are critical in ensuring that hot data used on every web request resides in memory and thus can be efficiently accessed and computed on.

For example, serving a single page involves many internal transactions accessing static and dynamic data such as images, text, and generating personalized content like stats, feeds, and ads. To ensure a fast response, this initial request is split into many smaller parallel transactions to multiple distributed servers. Each transaction must then complete promptly to meet the provider's service-level agreement (SLA), an example is that 99% of requests are satisfied within X milliseconds or microseconds. To meet these stringent latency requirements, the aforementioned in-memory services are often scaled horizontally in dedicated nodes, and run on hundreds of clusters representing hundreds of thousands of cores and growing [46, 49].

Ultimately, to meet these SLA targets while keeping the development and maintenance burden of these web pages reasonable, developers rely on generic reusable software components, which in turn run on the generic, widely used operating systems (OSes), such as Linux, that they know and are familiar with. While it is easy and natural to deploy on Linux, there is also a broad understanding in the systems community that *specialization* improves overall *efficiency*. Prior research has demonstrated that when running

SYSTOR '24, September 23–24, 2024, Virtual, Israel

ACM ISBN 979-8-4007-1181-7/24/09.
https://doi.org/10.1145/3688351.3689158

a single primary application, a specialized OS tailored to the application's needs, can yield orders of magnitude better performance [4, 8, 10, 15, 22, 24–26, 29–34, 36, 38–40, 43, 48, 51], however, we find there has been limited works to explore their performance and power efficiency [37]/carbon tradeoffs.

Therefore in this paper, we present preliminary results from a novel comparative performance and carbon study of two distinct OSes, Linux and a specialized library OS stack, EbbRT [43], on different hardware generations. We demonstrate that a baremetal EbbRT deployed on older process node technology (Q3'14) can yield significant performance and energy wins over newer hardware (Q2'21) running Linux. Further, our results also suggest that with the ever-increasing demands for cloud services, specialized OS stacks can help abate the demand to replace/upgrade older hardware.

Obviously, the use of specialized OSes need not be restricted to older hardware given prior work demonstrating how newer hardware features are used for significant performance gains [10, 15, 16, 25, 32, 36, 51]. However, in this work we restrict EbbRT to run on older hardware in order to: 1) motivate its adoption to address server hardware obsolescence while improving both performance and power/carbon efficiency, 2) highlight the portability challenges to using library OSes such as EbbRT as it would've required significant engineering effort (i.e. developing and maintaining device drivers) to work with new hardware, 3) and help pave a way forward of how one can marry the portability of Linux with the advantages of these specialized stacks in §4.

The remainder of this paper is organized as follows. In §2, we present results from our study on three generations of Intel server-class nodes. We then discuss the results with respect to carbon emissions in §3. In §4, we conclude with a discussion of potential implications and future work that our results suggest.

2 STUDY

2.1 Software Stacks

Our study aims to expose the performance and energy consumption of generic versus specialized software stacks. Each stack is composed of an OS and a primary application. We use two applications that prior work has used to evaluate in-memory services: 1) memcached [8, 14, 16, 32, 38, 43] and a transactional database, silo [10, 36].

2.1.1 ***OSes.*** We explore the impact of software stack specialization by comparing a commodity OS, Linux, commonly used by cloud providers and a research library OS, EbbRT, designed for event-driven network-oriented applications. These OSes represent two extremes in design: Linux is a general-purpose OS with tens of millions of lines of code supporting a wide range of hardware and applications, and EbbRT is a specialized OS with about 20K lines of code that supports a single application on a specific hardware platform.

Linux uses the default Ubuntu 22 images available on CloudLab [13], which runs the Linux 5.15 kernel. To ensure a fair comparison between the OSes, we optimize Linux by using the `performance` governor[1], fixing packet receive interrupts to specific cores, and enabling large-page support. We measure CPU package power and instruction counts using Linux *perf* [18], which employs Intel RAPL [9] for energy measurements, and has been independently validated by other researchers [11, 27, 50].

EbbRT is a library OS/unikernel that was developed almost a decade ago, whose advantages with respect to KVM virtualized performance (over 2X compared to Linux) was previously published [43]. Shortly after, we developed an EbbRT device driver for the Intel 10 GbE NIC from scratch [12], which permits EbbRT to run non-virtualized on bare-metal hardware with this older NIC. Like other specialized OSs [4, 8, 10, 30, 32, 34, 36, 38, 48, 51], EbbRT features a small optimized code base with custom event-driven interfaces and components designed to eliminate the overheads associated with a general-purpose OSes. Optimized applications are written to these interfaces, compiled with the kernel code, and executed within the single supervisor-privileged domain.

Unfortunately, these specialized OSes have seen little adoption due to the difficulty of maintaining and forward porting them to new hardware and software. To our knowledge, beyond UniKraft[28], there are few, if any, non-active research uni-kernels. Given our goal of evaluating, in the extreme, what benefits OS specialization can have concerning breathing new life into older hardware, we use bare-metal EbbRT, which aggressively uses multi-threading, large-pages, and has a version of both memcached and silo ported to it.

2.1.2 ***Applications.*** We use two standard applications to evaluate our OS stacks.

memcached: This widely used OS-demanding caching application benefits from specialization techniques like kernel bypass to improve performance [8, 14, 16, 32, 38, 43]. In our study, we use the Ubuntu-provided memcached-1.6.26 for the Linux stack, while EbbRT uses a re-implemented version tailored to its native interfaces, supporting the standard memcached binary protocol.

[1]We also experimented with Linux's dynamic `ondemand` governor, which showed reduced overall performance compared to `performance` mode. For example, in memcached on Linux-Node3, `ondemand` reached only 37% of the peak throughput of `performance`. However, at lighter RPS rates, `ondemand` used up to 43% less power than `performance`. Ultimately, EbbRT consumed the least power overall, 2.3 times less than `ondemand` for the same RPS rates.

Name	CPU	Node	Release	Threads	TDP (W)	Idle (W)	NIC	DDR4 RAM	SSD
Client	Intel Xeon Silver 4314	10 nm	Q2'21	32	135	45	Mellanox 40GbE	128GB	960 GB
Node1	Intel E5-2630 v3	22 nm	Q3'14	2 x 16	2 x 85	16	Intel 82599ES 10GbE	128GB	480 GB
Node2	Intel Xeon Silver 4114	14 nm	Q3'17	2 x 20	2 x 135	47	Intel X710 10GbE	192GB	480 GB
Node3	Intel Xeon Silver 4314	10 nm	Q2'21	2 x 32	2 x 135	89	Mellanox 40GbE	256GB	960 GB

Table 1: Different hardware used in experiments. We used Linux perf [18] to measure its idle power consumption. We have also validated the idle power number on LibOS for Node1. Note that Node1, Node2, Node3 all have 2 packages in one node.

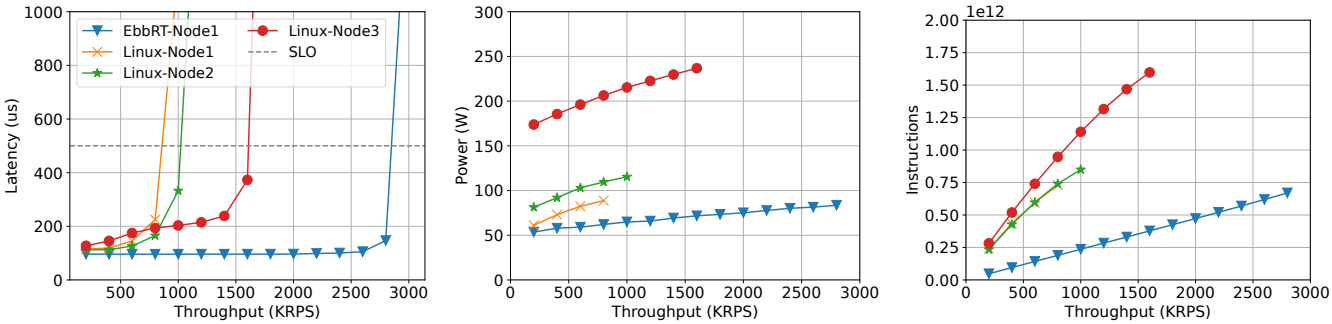


Figure 1: Memcached latency, power, and instruction count measures at different RPS rates.

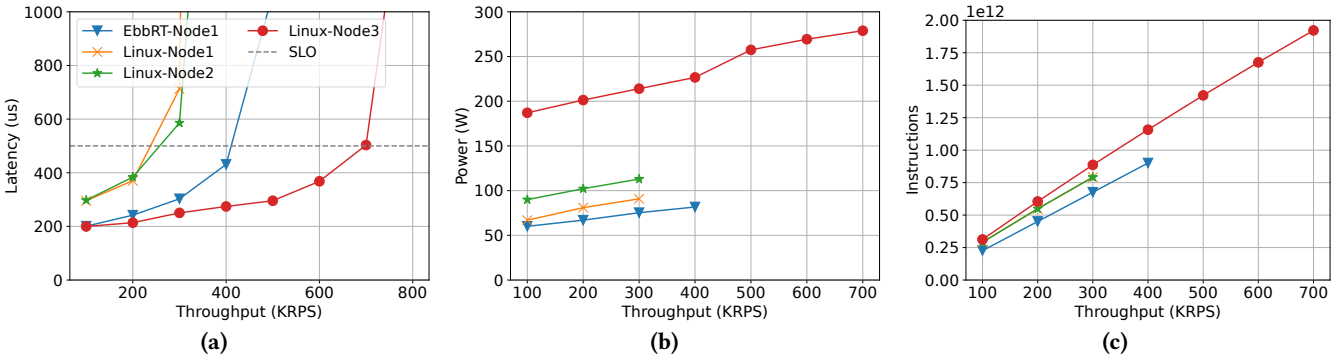


Figure 2: Silo latency, power, and instruction count measures at different RPS rates.

silo: Cloud services also use systems that support complex operations on stored data, such as in-memory relational databases (SQL and No-SQL) [3]. These databases impose significantly more CPU and memory load per operation than memcached. Silo [47] is designed to represent this type of application. We built, from source, a version of silo [35] with a web front-end for both OSes. Each incoming web request triggers a set of TPC-C transactions on its in-memory database component.

2.2 Hardware

Details of the hardware nodes used in the study are in Table 1. We set up a cluster of these nodes using CloudLab [13]. The core goal is to compare EbbRT booted on the oldest CPU technology, Node1 (2014), to Linux on all three CPU generations: Node1 (2014), Node2 (2017), and Node3 (2021).

2.3 Methodology

To generate load on the memcached and silo servers, we use the mutilate [21] tool in order to create load across a total of 1280 connections - this is coordinated by using four of the Client node type shown in table 1. The load is based on the representative ETC workload from Facebook [5]. We use a standard SLA (as used by others [8, 16, 32, 36, 43]) of 99% latency under 500 μs for both applications and we increase the amount of requests-per-second (RPS) at mean rates until the server node is saturated where it can no longer meet the SLA. We run each mean RPS rate for a total of 30 seconds in order to measure the server node's power, instruction count, and the reported 99% latency at the corresponding RPS rate. Experiments were repeated up to four times, with worst-case standard deviations of 2.49% from the mean.

2.4 Performance Results

Fig. 1 and fig. 2 illustrates the overall data from our study. For each application, we present performance as the load is increased (fig. 1a and fig. 2a), along with its measured power consumption (fig. 1b and fig. 2b,) and instruction counts (fig. 1c and fig. 2c)[2].

By enabling both OS stacks to use all available cores on a hardware node, we can observe how the stacks compare in terms of their ability to satisfy a particular load (e.g. throughput in KRPS). The memcached results indicate that `EbbRT-Node1` can satisfy requested loads with considerably fewer instructions: almost 5X reduction when compared to `Linux-Node1`. This reduction reveals both the "fat" (overheads) of the general-purpose OS and the ability of the specialized OS to trim it.

We note that given the OS-centric nature of memcached, the instruction counts in fig. 1c reveals more insight into the ability of EbbRT to squeeze more out of the hardware. While it is unsurprising that `Linux-Node2`, and `Linux-Node3` require more instructions given the higher thread count of their CPUs, `EbbRT-Node1` can still achieve a higher peak throughput (1.75X over `Linux-Node3`) while using the fewer threads available on `Node1`. Furthermore, contrasting the shape of the curves, one can see that EbbRT's instruction count scales linearly with the load. This indicates its ability to use the threads of the `Node1` more efficiently. In contrast, the non-linearity of the Linux curves shows poorer scalability and suggests multi-core overheads. Prior works in optimizing memcached [8, 16, 32, 36, 43] have also remarked on these scalability issues.

Given that silo is a more compute-intensive application and that both OS stacks are constructed with the same application source, it is harder to see the instruction count differences in fig. 2c. The raw data reveals that at a query rate of 100 KRPS on `Node1`, EbbRT requires 22% fewer instructions compared to Linux. While not as dramatic a difference, eliminating this overhead helped contribute to the specialized OS being able to squeeze out a 2X improvement in the peak throughput compared to `Linux-Node1` and `Linux-Node2`. Lastly, we note that the increased number of CPUs and faster clock speeds of `Node3` eventually enable Linux to achieve the highest peak throughput for silo in our study (1.75X better than EbbRT-Node1). The following sections examines the carbon implications of these trade-offs in more detail.

3 CARBON ANALYSIS

In the prior section, we observed that a specialized software stack can support critical work with fewer instructions, resulting in greater efficiency, higher performance, and lower energy consumption when run on older hardware (a 22nm-2014 CPU, DRAM, and NIC conforming to a networking standard defined in 2002 [1]). This data also exposes a form of software-driven hardware obsolescence that increases e-waste and carbon. To extrapolate on this observation, we conduct an analysis below which considers both the embodied and operational carbon of our study.

[2]Note for figures (b) and (c), we filter out RPS values that violate the SLA objective.

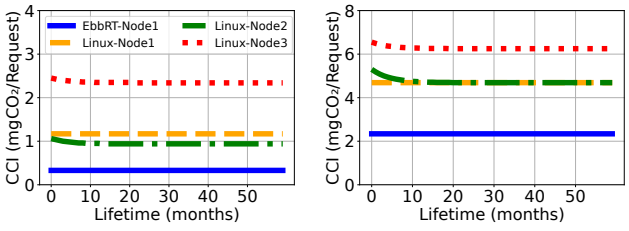
	CO2 (kg)			
	CPU	DRAM	SSD	Total
Node1	14.2	86.9	17.3	118.4
Node2	13.1	122.5	17.3	152.9
Node3	15.1	172.5	34.3	221.9

Table 2: Embodied carbon calculations.

3.1 Computational Carbon Intensity (CCI)

For this analysis, we use the Computational Carbon Intensity (CCI) metric [45] as it helps calculate the lifetime CO_2 released per unit of work. Further, CCI captures both embodied and operational carbon while rewarding the reuse of existing hardware. Roughly, CCI is defined as: $\frac{\sum_{lifetime}(C_M+C_C+C_N)}{\sum_{lifetime} ops}$

C_M is the embodied carbon and we use information provided by ACT [19], to calculate it for the three servers used in our study in table 2[3]. C_C is operational carbon and it is calculated as $CI_{grid} * E$ where CI_{grid} is the carbon intensity of the electrical grid and E is energy consumption of the server. We calculated a CI_{grid} of 277.3 gCO_2/kWh using a New England resource mix [20]. C_N is networking carbon and is calculated as $CI_{grid} * f_{net} * EI_{net}$, where f_{net} is the server's NIC speed and EI_{net} is energy intensity of networking; we use a published value of 2.7 μJoules/Byte [6]. We consider the lifetime ops of each server as the peak rates shown in fig. 1a, fig. 2a sustained across multiple years. CCI does not consider multiple lifetimes (i.e. repairs/maintenance) as it can be difficult to get this visibility in old hardware.



(a) memcached **(b) silo**

Figure 3: CCI for both workloads. Lower is better.

[3]We expect the embodied carbon trend to continue to increase as a modern server, such as those deployed in the Aurora supercomputing cluster (10nm CPU, deployed 2023) [2], contains an embodied carbon of 732 kg.

3.2 Analysis

Fig. 3 shows the computed CCI for memcached and silo across both OSes. Similar to [45], we assume C_M (embodied carbon) of the oldest server, `Node1`, is 0. Further, a refurbished server such as `Node1` [42] can also be purchased for 4X cheaper compared to `Node3` [44].

In both memcached and silo, `EbbRT-Node1` achieved the lowest CCI and is typically 2-6X lower compared to Linux. We find that the peak RPS-per-Watt of `EbbRT-Node1` contributed greatly to its lower CCI as well as the carbon savings from the use of older hardware components (i.e. CPU, NIC, etc). Further, by sticking with `EbbRT-Node1`, a provider not only abates the need to upgrade a memcached server until the request peak rate increases above 2800 KRPS (1.75X beyond `Linux-Node3`), but it also can exploit the low CCI to process each request more efficiently. For silo, though `EbbRT-Node1` cannot reach the peak RPS of `Linux-Node3`, its CCI efficiency indicates a provider could choose to scale horizontally by purchasing from the refurbished market another `Node1` hardware instead.

It has been noted that the load on web service's fluctuate significantly due to its diurnal patterns [5, 23]; with datacenter service's often running at low utilization to respond to bursty behavior [7]. From a carbon perspective, this can significantly change the trade-offs concerning hardware refresh that specialized OSes can help unlock. `Node1` (2014) has both a lower idle power consumption and thermal design power (TDP)[4] when compared to `Node2` (2017) and `Node3` (2021). For memcached at peak load, `Linux-Node3` induced only a 1.33X increase in power when compared to `Linux-Node1`, at lighter loads (200 KRPS) we find this actually increased to over 2.84X. In contrast, using `EbbRT-Node1` yields a reduction of 12% power compared to `Linux-Node1` at light loads. Therefore over a server's lifetime, we find that the combination of a specialized stack running on older hardware serves as better units for horizontal scaling while reducing both embodied and operational carbon costs.

4 FUTURE DIRECTIONS

Our study illustrates how optimizing the OS can help address the three **R**'s of sustainability for cloud providers [19]: 1) **R**educe the demand for new hardware systems, 2) **R**euse older systems for a broader set of roles, and 3) have a path to utilizing **R**ecycled hardware for modern use cases. However, there are still some open questions and challenges that must be addressed.

As mentioned, although there have been significant effort for over a decade to build specialized OSes; there has been little real-world deployments of these systems due to their portability limitations. To reap the potential benefits that our study suggests, the challenges of porting specialized OSes deserve more attention. From literature, we find a key optimization approach taken by these systems is to bypass traditional kernel paths and co-optimize directly with the hardware (i.e. NICs) itself; either through in-house implementations [15, 34, 43] or the use of third-party libraries such as DPDK [8, 10, 24, 25, 32, 36, 51]. Unfortunately, this limits its portability as these systems typically require a rewrite or modification of the original application to take advantage of systems optimizations. Ideally, one would like to marry the portability and richness of Linux with specialized OS stacks.

Junction [15] is an example of recent library OS work that makes kernel-bypass practical by adding compatibility to run unmodified Linux applications. However, it relies on advanced hardware features of new data center CPUs and NICs (as well as a custom mlnx5 driver). Another approach is to take incremental steps instead where any unmodified application, running on Linux, can be the starting point and incremental application and OS co-optimization can be induced to move its performance and power profile towards specialized OSes. Expanding on this, UniKernel Linux (UKL) [41], introduces a set of Linux kernel configurations that permits primary application to be compiled into the kernel. When the primary application is launched, it executes at the same hardware privilege as the kernel. As such the application can directly call internal kernel routines to optimize its behavior. This raises the interesting possibility of developing a novel hybrid OS runtime.

We can consider a port of the scalable components of one of the many available specialized OSes [4, 8, 10, 22, 25, 26, 29, 31–34, 36, 38, 43, 48, 51] that is linked with the primary application but also designed, and implemented, to judiciously use internal Linux routines to inherit Linux's hardware compatibility. When the primary application is started, it bootstraps the specialized OS components. It relies on these for core functionality such as high performance scalable memory allocation and event scheduling. However, rather than including its own NIC driver, the specialized OS exploits the standardized NAPI interfaces to use the NIC drivers present in Linux. A similar approach can be take for other devices such as GPUs, SSD, and other hardware. In this approach an application can incrementally exploit the optimized code paths of the specialized OS and the specialized OS can also exploit Linux's battle-tested codebase and large hardware compatibility list to support a broad range of use-cases.

[4]TDP is a Intel provided metric that represents the average power the processor dissipates under some predefined high-complexity workload.

ACKNOWLEDGMENTS

We would like to thank our shepherd, David Irwin, for his help in preparing the final version. This work is supported through the Red Hat Collaboratory, Optimizing Kernel Paths for Performance and Energy Hardware, 2024-01-RH09.

REFERENCES

[1] [n.d.]. 10 Gigabit Ethernet. https://en.wikipedia.org/wiki/10_Gigabit_Ethernet.

[2] [n.d.]. Aurora System Overview. https://docs.alcf.anl.gov/aurora/hardware-overview/machine-overview/.

[3] [n.d.]. List of in-memory databases. https://en.wikipedia.org/wiki/List_of_in-memory_databases.

[4] Antti Kantee, Justin Cormack. [n.d.]. Rump Kernels: No OS? No Problem! https://www.usenix.org/publications/login/october-2014-vol-39-no-5.

[5] Atikoglu, Berk and Xu, Yuehai and Frachtenberg, Eitan and Jiang, Song and Paleczny, Mike. 2012. Workload Analysis of a Large-scale Key-value Store. In *Proceedings of the 12th ACM SIGMETRICS/PERFORMANCE Joint International Conference on Measurement and Modeling of Computer Systems* (London, England, UK) *(SIGMETRICS '12)*. ACM, New York, NY, USA, 53–64. https://doi.org/10.1145/2254756.2254766

[6] Jayant Baliga, Robert W. A. Ayre, Kerry Hinton, and Rodney S. Tucker. 2011. Green Cloud Computing: Balancing Energy in Processing, Storage, and Transport. *Proc. IEEE* 99, 1 (2011), 149–167. https://doi.org/10.1109/JPROC.2010.2060451

[7] Luiz Andre Barroso and Urs Hoelzle. 2009. *The Datacenter As a Computer: An Introduction to the Design of Warehouse-Scale Machines* (1st ed.). Morgan and Claypool Publishers.

[8] Adam Belay, George Prekas, Ana Klimovic, Samuel Grossman, Christos Kozyrakis, and Edouard Bugnion. 2014. IX: A Protected Dataplane Operating System for High Throughput and Low Latency. In *Proceedings of the 11th USENIX Conference on Operating Systems Design and Implementation* (Broomfield, CO) *(OSDI'14)*. USENIX Association, USA, 49–65.

[9] Howard David, Eugene Gorbatov, Ulf R. Hanebutte, Rahul Khanna, and Christian Le. 2010. RAPL: Memory Power Estimation and Capping. In *Proceedings of the 16th ACM/IEEE International Symposium on Low Power Electronics and Design* (Austin, Texas, USA) *(ISLPED '10)*. Association for Computing Machinery, New York, NY, USA, 189–194. https://doi.org/10.1145/1840845.1840883

[10] Henri Maxime Demoulin, Joshua Fried, Isaac Pedisich, Marios Kogias, Boon Thau Loo, Linh Thi Xuan Phan, and Irene Zhang. 2021. When Idling is Ideal: Optimizing Tail-Latency for Heavy-Tailed Datacenter Workloads with Perséphone. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles* (Virtual Event, Germany) *(SOSP '21)*. Association for Computing Machinery, New York, NY, USA, 621–637. https://doi.org/10.1145/3477132.3483571

[11] Spencer Desrochers, Chad Paradis, and Vincent M. Weaver. 2016. A Validation of DRAM RAPL Power Measurements. In *Proceedings of the Second International Symposium on Memory Systems* (Alexandria, VA, USA) *(MEMSYS '16)*. Association for Computing Machinery, New York, NY, USA, 455–470. https://doi.org/10.1145/2989081.2989088

[12] Han Dong, Sanjay Arora, Yara Awad, Tommy Unger, Orran Krieger, and Jonathan Appavoo. 2021. Slowing Down for Performance and Energy: An OS-Centric Study in Network Driven Workloads. arXiv:cs.OS/2112.07010

[13] Dmitry Duplyakin, Robert Ricci, Aleksander Maricq, Gary Wong, Jonathon Duerig, Eric Eide, Leigh Stoller, Mike Hibler, David Johnson, Kirk Webb, Aditya Akella, Kuangching Wang, Glenn Ricart, Larry Landweber, Chip Elliott, Michael Zink, Emmanuel Cecchet, Snigdhaswin Kar, and Prabodh Mishra. 2019. The Design and Operation of CloudLab. In *Proceedings of the USENIX Annual Technical Conference (ATC)*. 1–14. https://www.flux.utah.edu/paper/duplyakin-atc19

[14] Brad Fitzpatrick. 2004. Distributed Caching with Memcached. *Linux Journal* 2004, 124 (Aug. 2004), 5. http://dl.acm.org/citation.cfm?id=1012889.1012894

[15] Joshua Fried, Gohar Irfan Chaudhry, Enrique Saurez, Esha Choukse, Inigo Goiri, Sameh Elnikety, Rodrigo Fonseca, and Adam Belay. 2024. Making Kernel Bypass Practical for the Cloud with Junction. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*. USENIX Association, Santa Clara, CA, 55–73. https://www.usenix.org/conference/nsdi24/presentation/fried

[16] Joshua Fried, Zhenyuan Ruan, Amy Ousterhout, and Adam Belay. 2020. Caladan: mitigating interference at microsecond timescales *(OSDI'20)*. USENIX Association, USA, Article 16, 17 pages.

[17] Yu Gan, Yanqi Zhang, Dailun Cheng, Ankitha Shetty, Priyal Rathi, Nayan Katarki, Ariana Bruno, Justin Hu, Brian Ritchken, Brendon Jackson, Kelvin Hu, Meghna Pancholi, Yuan He, Brett Clancy, Chris Colen, Fukang Wen, Catherine Leung, Siyuan Wang, Leon Zaruvinsky, Mateo Espinosa, Rick Lin, Zhongling Liu, Jake Padilla, and Christina Delimitrou. 2019. An Open-Source Benchmark Suite for Microservices and Their Hardware-Software Implications for Cloud & Edge Systems. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems* (Providence, RI, USA) *(ASPLOS '19)*. Association for Computing Machinery, New York, NY, USA, 3–18. https://doi.org/10.1145/3297858.3304013

[18] Brendan Gregg. [n.d.]. perf Examples. https://www.brendangregg.com/perf.html.

[19] Udit Gupta, Mariam Elgamal, Gage Hills, Gu-Yeon Wei, Hsien-Hsin S. Lee, David Brooks, and Carole-Jean Wu. 2022. ACT: designing sustainable computer systems with an architectural carbon modeling tool. In *Proceedings of the 49th Annual International Symposium on Computer Architecture* (New York, New York) *(ISCA '22)*. Association for Computing Machinery, New York, NY, USA, 784–799. https://doi.org/10.1145/3470496.3527408

[20] ISO New England. [n.d.]. https://www.iso-ne.com/. Accessed on 08/20/2024.

[21] J. Leverich. [n.d.]. Mutilate: high performance memcached load generator. https://github.com/leverich/mutilate.

[22] EunYoung Jeong, Shinae Wood, Muhammad Jamshed, Haewon Jeong, Sunghwan Ihm, Dongsu Han, and KyoungSoo Park. 2014. mTCP: a Highly Scalable User-level TCP Stack for Multicore Systems. In *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*. USENIX Association, Seattle, WA, 489–502. https://www.usenix.org/conference/nsdi14/technical-sessions/presentation/jeong

[23] Rashmi Vinayak Juncheng Yang, Yao Yue. 2020. A large scale analysis of hundreds of in-memory cache clusters at Twitter. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. USENIX Association. https://www.usenix.org/conference/osdi20/presentation/yang

[24] Kostis Kaffes, Timothy Chong, Jack Tigar Humphries, Adam Belay, David Mazières, and Christos Kozyrakis. 2019. Shinjuku: Preemptive Scheduling for microsecond-scale Tail Latency. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*. USENIX Association, Boston, MA, 345–360. https://www.usenix.org/conference/nsdi19/presentation/kaffes

[25] Anuj Kalia, Michael Kaminsky, and David Andersen. 2019. Datacenter RPCs can be General and Fast. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*. USENIX Association, Boston, MA, 1–16. https://www.usenix.org/conference/nsdi19/presentation/kalia

[26] Antoine Kaufmann, SImon Peter, Naveen Kr. Sharma, Thomas Anderson, and Arvind Krishnamurthy. 2016. High Performance Packet Processing with FlexNIC. In *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems* (Atlanta, Georgia, USA) *(ASPLOS '16)*. Association for Computing Machinery, New York, NY, USA, 67–81.

https://doi.org/10.1145/2872362.2872367

[27] Kashif Nizam Khan, Mikael Hirki, Tapio Niemi, Jukka K. Nurminen, and Zhonghong Ou. 2018. RAPL in Action: Experiences in Using RAPL for Power Measurements. *ACM Trans. Model. Perform. Eval. Comput. Syst.* 3, 2, Article 9 (March 2018), 26 pages. https://doi.org/10.1145/3177754

[28] 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. 2021. Unikraft: fast, specialized unikernels the easy way. In *Proceedings of the Sixteenth European Conference on Computer Systems* (Online Event, United Kingdom) *(EuroSys '21)*. Association for Computing Machinery, New York, NY, USA, 376–394. https://doi.org/10.1145/3447786.3456248

[29] Hyeontaek Lim, Dongsu Han, David G. Andersen, and Michael Kaminsky. 2014. MICA: A Holistic Approach to Fast In-Memory Key-Value Storage. In *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*. USENIX Association, Seattle, WA, 429–444. https://www.usenix.org/conference/nsdi14/technical-sessions/presentation/lim

[30] Anil Madhavapeddy, Richard Mortier, Charalampos Rotsos, David Scott, Balraj Singh, Thomas Gazagnaire, Steven Smith, Steven Hand, and Jon Crowcroft. 2013. Unikernels: Library Operating Systems for the Cloud. In *Proceedings of the Eighteenth International Conference on Architectural Support for Programming Languages and Operating Systems* (Houston, Texas, USA) *(ASPLOS '13)*. ACM, New York, NY, USA, 461–472. https://doi.org/10.1145/2451116.2451167

[31] Ilias Marinos, Robert N.M. Watson, and Mark Handley. 2014. Network Stack Specialization for Performance. In *Proceedings of the 2014 ACM Conference on SIGCOMM* (Chicago, Illinois, USA) *(SIGCOMM '14)*. ACM, New York, NY, USA, 175–186. https://doi.org/10.1145/2619239.2626311

[32] Amy Ousterhout, Joshua Fried, Jonathan Behrens, Adam Belay, and Hari Balakrishnan. 2019. Shenango: Achieving High CPU Efficiency for Latency-Sensitive Datacenter Workloads. In *Proceedings of the 16th USENIX Conference on Networked Systems Design and Implementation* (Boston, MA, USA) *(NSDI'19)*. USENIX Association, USA, 361–377.

[33] Aleksey Pesterev, Jacob Strauss, Nickolai Zeldovich, and Robert T. Morris. 2012. Improving Network Connection Locality on Multicore Systems. In *Proceedings of the 7th ACM European Conference on Computer Systems* (Bern, Switzerland) *(EuroSys '12)*. Association for Computing Machinery, New York, NY, USA, 337–350. https://doi.org/10.1145/2168836.2168870

[34] Simon Peter, Jialin Li, Irene Zhang, Dan R. K. Ports, Doug Woos, Arvind Krishnamurthy, Thomas Anderson, and Timothy Roscoe. 2015. Arrakis: The Operating System Is the Control Plane. *ACM Trans. Comput. Syst.* 33, 4, Article 11 (Nov. 2015), 30 pages. https://doi.org/10.1145/2812806

[35] George Prekas. 2017. https://github.com/ix-project/servers/tree/master.

[36] George Prekas, Marios Kogias, and Edouard Bugnion. 2017. ZygOS: Achieving Low Tail Latency for Microsecond-Scale Networked Tasks. In *Proceedings of the 26th Symposium on Operating Systems Principles* (Shanghai, China) *(SOSP '17)*. Association for Computing Machinery, New York, NY, USA, 325–341. https://doi.org/10.1145/3132747.3132780

[37] George Prekas, Mia Primorac, Adam Belay, Christos Kozyrakis, and Edouard Bugnion. 2015. Energy Proportionality and Workload Consolidation for Latency-Critical Applications. In *Proceedings of the Sixth ACM Symposium on Cloud Computing* (Kohala Coast, Hawaii) *(SoCC '15)*. Association for Computing Machinery, New York, NY, USA, 342–355. https://doi.org/10.1145/2806777.2806848

[38] Henry Qin, Qian Li, Jacqueline Speiser, Peter Kraft, and John Ousterhout. 2018. Arachne: Core-Aware Thread Management. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. USENIX Association, Carlsbad, CA, 145–160. https://www.usenix.org/conference/osdi18/presentation/qin

[39] Rajesh Nishtala and Hans Fugal and Steven Grimm and Marc Kwiatkowski and Herman Lee and Harry C. Li and Ryan McElroy and Mike Paleczny and Daniel Peek and Paul Saab and David Stafford and Tony Tung and Venkateshwaran Venkataramani. 2013. Scaling Memcache at Facebook. In *Presented as part of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*. USENIX, Lombard, IL, 385–398. https://www.usenix.org/conference/nsdi13/technical-sessions/presentation/nishtala

[40] Ali Raza, Parul Sohal, James Cadden, Jonathan Appavoo, Ulrich Drepper, Richard Jones, Orran Krieger, Renato Mancuso, and Larry Woodman. 2019. Unikernels: The Next Stage of Linux's Dominance. In *Proceedings of the Workshop on Hot Topics in Operating Systems* (Bertinoro, Italy) *(HotOS '19)*. Association for Computing Machinery, New York, NY, USA, 7–13. https://doi.org/10.1145/3317550.3321445

[41] Ali Raza, Thomas Unger, Matthew Boyd, Eric B Munson, Parul Sohal, Ulrich Drepper, Richard Jones, Daniel Bristot De Oliveira, Larry Woodman, Renato Mancuso, Jonathan Appavoo, and Orran Krieger. 2023. Unikernel Linux (UKL). In *Proceedings of the Eighteenth European Conference on Computer Systems* (Rome, Italy) *(EuroSys '23)*. Association for Computing Machinery, New York, NY, USA, 590–605. https://doi.org/10.1145/3552326.3587458

[42] Restored Cisco C220 M4 1U Rack Server. [n.d.]. https://www.walmart.com/ip/Restored-Cisco-C220-M4-1U-Rack-Server-2-x-Intel-Xeon-E5-2630-v4-Deca-core-10-Core-2-20-GHz-64-GB-Installed-DDR4-SDRAM-Serial-ATA-600-Controller-0-1-1/5107050837. Accessed on 08/19/2024.

[43] Dan Schatzberg, James Cadden, Han Dong, Orran Krieger, and Jonathan Appavoo. 2016. EbbRT: A Framework for Building Per-Application Library Operating Systems. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. USENIX Association, GA, 671–688. https://www.usenix.org/conference/osdi16/technical-sessions/presentation/schatzberg

[44] Supermicro 2U Ultra SuperServer (SYS-220U-TNR). [n.d.]. https://store.supermicro.com/us_en/supermicro-2u-ultra-superserver-sys-220u-tnr.html. Accessed on 08/19/2024.

[45] Jennifer Switzer, Gabriel Marcano, Ryan Kastner, and Pat Pannuto. 2023. Junkyard Computing: Repurposing Discarded Smartphones to Minimize Carbon. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2* (Vancouver, BC, Canada) *(ASPLOS 2023)*. Association for Computing Machinery, New York, NY, USA, 400–412. https://doi.org/10.1145/3575693.3575710

[46] Chunqiang Tang, Kenny Yu, Kaushik Veeraraghavan, Jonathan Kaldor, Scott Michelson, Thawan Kooburat, Aravind Anbudurai, Matthew Clark, Kabir Gogia, Long Cheng, Ben Christensen, Alex Gartrell, Maxim Khutornenko, Sachin Kulkarni, Marcin Pawlowski, Tuomas Pelkonen, Andre Rodrigues, Rounak Tibrewal, Vaishnavi Venkatesan, and Peter Zhang. 2020. Twine: A Unified Cluster Management System for Shared Infrastructure. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. USENIX Association, 787–803. https://www.usenix.org/conference/osdi20/presentation/tang

[47] Stephen Tu, Wenting Zheng, Eddie Kohler, Barbara Liskov, and Samuel Madden. 2013. Speedy Transactions in Multicore In-Memory Databases. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles* (Farminton, Pennsylvania) *(SOSP '13)*. Association for Computing Machinery, New York, NY, USA, 18–32. https://doi.org/10.1145/2517349.2522713

[48] Matt Welsh, David Culler, and Eric Brewer. 2001. SEDA: An Architecture for Well-Conditioned, Scalable Internet Services. *SIGOPS Oper.*

Syst. Rev. 35, 5 (Oct. 2001), 230–243. https://doi.org/10.1145/502059.502057

[49] Juncheng Yang, Yao Yue, and K. V. Rashmi. 2020. A large scale analysis of hundreds of in-memory cache clusters at Twitter. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. USENIX Association, 191–208. https://www.usenix.org/conference/osdi20/presentation/yang

[50] Huazhe Zhang and H Hoffman. 2015. A Quantitative Evaluation of the RAPL Power Control System. *Feedback Computing* (2015).

[51] Irene Zhang, Amanda Raybuck, Pratyush Patel, Kirk Olynyk, Jacob Nelson, Omar S. Navarro Leija, Ashlie Martinez, Jing Liu, Anna Kornfeld Simpson, Sujay Jayakar, Pedro Henrique Penna, Max Demoulin, Piali Choudhury, and Anirudh Badam. 2021. The Demikernel Datapath OS Architecture for Microsecond-scale Datacenter Systems. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles* (Virtual Event, Germany) *(SOSP '21)*. Association for Computing Machinery, New York, NY, USA, 195–211. https://doi.org/10.1145/3477132.3483569