The Case of Unsustainable CPU Affinity

Jiechen Zhao
University of Toronto
Toronto, Canada

Katie Lim
University of Washington
Seattle, USA

Thomas Anderson
University of Washington
Seattle, USA

Natalie Enright Jerger
University of Toronto
Toronto, Canada

ABSTRACT

CPU affinity reduces data copies and improves data locality and has become a prevalent technique for high-performance programs in datacenters. This paper explores the tension between CPU affinity and sustainability. In particular, affinity settings can lead to significant uneven aging of cores on a CPU. We observe that infrastructure threads, used in a wide spectrum of network, storage, and virtualization sub-systems, exercise their affinitized cores up to 23× more when compared to typical μs-scale application threads. In addition, we observe that the affinitized infrastructure threads generate regional heat hot spots and preclude CPUs from being used with the expected lifetime. Finally, we discuss design options to tackle the unbalanced core-aging problem to improve the overall sustainability of CPUs and call for more attention to sustainability-aware affinity and mitigation of such problems.

CCS CONCEPTS

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

KEYWORDS

sustainability, CPU, micro-architecture, reliability, datacenter software, datacenter infrastructure, operating systems

ACM Reference Format:
Jiechen Zhao, Katie Lim, Thomas Anderson, and Natalie Enright Jerger. 2023. The Case of Unsustainable CPU Affinity. In *2nd Workshop on Sustainable Computer Systems (HotCarbon '23), July 9, 2023, Boston, MA, USA.* ACM, Boston, MA, USA, 7 pages. https://doi.org/10.1145/3604930.3605706

1 INTRODUCTION

Today's datacenter applications require a response to a single user request from thousands of software services. Achieving high request rates under μs-scale tail latency is particularly important for requests with service times of only a couple of μs (e.g., MemCacheD [36] or RAMCloud [39]). However, recent studies have shown that CPUs are not well-designed for today's cloud applications [10, 18, 26, 46, 47]. The common bottlenecks for those μs-scale applications are CPU front-end stalls and data miss penalties, even when no resource interference exists.

CPU affinity is a system setting that is used to improve data locality and reduce context switches. The affinity is realized by the operating system (OS) that binds a thread and a physical core so that the thread always runs on that particular core. The affinity setting has been widely used to mitigate the aforementioned CPU efficiency problem. For example, modern latency-critical (LC) applications benefit from the affinity setting to achieve low end-to-end latency and high throughput [25, 32].

This paper focuses on a broader usage of the affinity setting. Within datacenters, besides application threads, many *infrastructure-managed threads* need CPU affinity for high performance as well. Although those threads do not execute useful business logic, their performance fundamentally determines the tail latency and throughput of LC applications because they closely coordinate with LC application threads on each request. These threads widely exist in low-latency network/storage stacks [2, 6, 11, 41, 44, 50], virtualization stacks [31, 40, 49], and many user- or OS-level modern schedulers [2, 24, 38].

In this work, we study the effects of CPU affinity in terms of sustainability: *what are the implications of CPU affinity on cores' lifetime?* Two main factors have been positioned as determinants of cores' lifespan. First, transistors gradually wear out by the continuous movement of charges, manifesting as slower transistors and, thus, slower critical paths [12]. Second, thermal variance across a chip die generates hot spots. The heat asymmetry causes gradual elevation of the threshold voltage and longer transistor switching delay [16, 48] on some cores.

Although prior work discusses some relevant aspects of CPU lifetimes, we find there are still the following two fundamental challenges in understanding, measuring, and anticipating cores' sustainability:

- There lacks a methodology to distinguish infrastructure threads and LC applications threads that require core-pinning. We term them core$_{Infra}$ and core$_{LCApp}$.[1] While the microarchitectural features of core$_{LCApp}$ have been extensively explored [10, 18, 26, 46, 47], there has been little exploration of common features across different core$_{Infra}$.
- Although a wide range of modeling has been proposed to estimate the lifetime of cores, they usually require knowledge such as transistor-level activity factors and manufacturing parameters [37, 48]. This information is either hard or too costly to

[1]Since we focus on CPU affinity, if not clarified, threads and cores are interchangeable in this paper.

collect over time. Metrics visible at runtime and manageable by cloud providers are still missing.

To this end, this work investigates the downside of CPU affinity across various core$_{LCApp}$ and core$_{Infra}$. Three key insights that guide our proposal:

- Even though instructions per cycle (IPC) is sometimes not an accurate metric for measuring performance [7, 30], IPC is a good metric to describe core-aging, which correlates with the switching activity of critical transistor paths [37]. On top of that, the instructions per second metric is useful to represent how worn-out a core has been during a time unit.
- Our experiments suggest that it is important to distinguish between core$_{LCApp}$ and core$_{Infra}$ for maintaining cores' sustainability in the case of CPU affinity. We demonstrate that core$_{Infra}$ delivers up to 8× higher IPC and 23× higher instructions per second than core$_{LCApp}$.
- We pin-point that core$_{Infra}$ creates heat hot spots, increases the chances of unexpected CPU failures and carries new thermal management challenges. Our experiments running a TCP echo benchmark on the Demikernel dataplane OS [50] show that core$_{Infra}$ generates up to 1.5× more heat than other core$_{LCApp}$.

The organization of this paper is as follows. Sec. 2 introduces the prevalence of CPU affinity, which is especially required by infrastructure threads. Then, Sec. 3 characterizes microarchitectural features of core$_{Infra}$ and core$_{LCApp}$, and offers insights that help reason about the unsustainability caused by CPU affinity. Next, Sec. 4 provides design options on how to mitigate sustainability problems due to CPU affinity, including a new metric and designs from OS and system architecture perspectives. Finally, Sec. 5 and Sec. 6 discuss related works and conclude this paper.

2 THE PREVALENCE OF CPU AFFINITY

In the literature, application threads that require CPU affinity have been extensively explored [10, 18, 26, 46, 47]. In this paper, we primarily highlight the prevalence of CPU affinity for *infrastructure threads* in the following categories:

- Virtualization overheads can be significantly reduced by exclusively dedicating physical cores (i.e., sidecores [31, 40]) to its user-level virtualization layer [49] via virtual device emulation. The sidecore polls guest I/O operations via shared memory regions, so no VM exits are needed to submit device commands. Moreover, the user-level NVMe device driver [49] enables sidecores without user-kernel mode switches.
- As scheduling in the network stack favors a centralized queuing model for low tail latency [15, 24], system designers dedicate physical cores to balance loads and dispatch packets to application threads [24, 38].
- Low-latency network stacks get rid of high overhead interrupts and employ dedicated cores to poll device queues to improve I/O performance [2]. Some work pins network connections on the same core to improve data locality and reduce coherence misses [11, 42]. More recent OS designs take advantage of user-level busy-polling [11, 41, 44, 50] to guarantee microsecond-scale tail latency. End-to-end low-latency networking systems, such as MICA [32] and eRPC [25] dedicate polling cores to receive short requests. Note that low-latency storage stacks behave similarly and benefit from CPU affinity [6, 29, 31].

In addition, infrastructure administrators stick to this thread-to-core pinning model for *performance scalability*, which sacrifices more sustainability if not handled properly. More cores should be pinned to infrastructure threads to achieve performance scaling. Other examples of why CPU affinity is even more essential for scalable infrastructure management include:

- Distributing and balancing interrupt requests (IRQs) from a particular I/O device across multiple physical cores can improve interrupt handling throughput to catch up with device scaling [1, 3].
- Servers with multiple NICs or SSDs require the number of polling cores to scale proportionally [19, 29]. Virtualization further mandates 16 extra CPU cores to saturate 12 SSDs [29].
- Since one physical core limits scheduling throughput to under 5 million packets per second [24, 38], further throughput improvements come from dedicating multiple cores to the dispatch threads [24, 53].

3 UNDERSTANDING THE IMPACT OF CPU AFFINITY ON SUSTAINABILITY

In this section, we demonstrate our findings with respect to unsustainable CPU affinity. Our characterizations suggest that core$_{Infra}$ and core$_{LCApp}$ should be treated differently in terms of lifespan.

Observation Summary: Significant unevenness of IPC, CPU stalls, instructions per second, and temperature exist between core$_{Infra}$ and core$_{LCApp}$. Therefore, core$_{Infra}$ may potentially fail much sooner due to faster transistor wear-out, leading to more frequent hardware refreshes and higher embodied carbon emissions.

3.1 Differentiating Infrastructure Cores and Application Cores

We profile low-level features of cores that have pinned threads for performance benefits. Our experiments run on Intel Xeon Gold 6145 with 2 sockets, 20 cores per socket, and Mellanox MT27710 ConnectX-4 Lx NIC and Intel NVMe P4600 SSD. We use Linux Perf as the measurement tool. We use the pState driver with performance governor that scales each core's frequency from 1GHz to 3.7GHz depending on demand.

IPC comparison. Fig. 1 and Fig. 2 show a comparison of instructions per cycle (IPC) between core$_{Infra}$ and core$_{LCApp}$. Fig. 1 illustrates the common observation that LC applications usually run at low IPC between 0.4–1.0, which is corroborated by a plethora of prior work [9, 14, 26, 47].

In Fig. 2, we profile threads that are used in infrastructure tasks and that usually require CPU affinity for performance benefits. We characterize a spectrum of infrastructure tasks that are pinned on physical cores. `Poll1` and `Poll2` represent I/O submission/completion queue busy-polling in DPDK [2] and SPDK [6], respectively. `Router` shows an L3 network router program that forwards received packets to another by accessing 5-tuple hash objects and matching a flow table at runtime. `Interrupt` and `Virtualization`

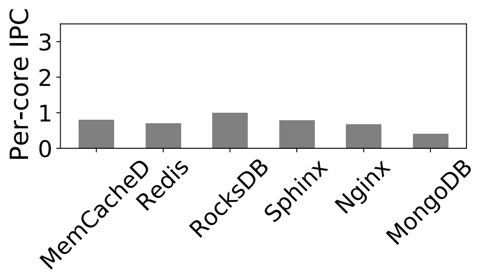

Figure 1: Application core IPC profiling.

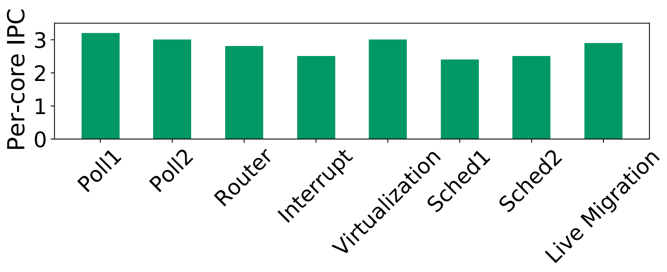

Figure 2: Infrastructure core IPC profiling.

characterizes IPCs of IRQ cores [14] and cores that are dedicated to performing virtual-to-physical address remapping in the Sidecore approach [29], respectively. `Sched1` and `Sched2` represent cores that are dedicated to scheduling (managing queues and dispatching requests to other application threads) in Shinjuku [24] and Demikernel [50], respectively. `Live Migration` shows a QEMU/KVM managed VM live migration task [45].

As Fig. 2 demonstrates, cores$_{Infra}$ exhibit 2.8-8× higher IPC than cores$_{LCApp}$, and this high-IPC characteristic is common across most infrastructure threads we evaluate. The much higher IPC implies that core$_{Infra}$ may wear out faster than core$_{LCApp}$ due to their inherent instruction-level differences. CPU affinity amplifies such differences and causes unbalanced cores' lifetime. Next, we perform detailed analyses to answer the following:

- What is the difference at the microarchitectural-level?
- What metrics would help measure sustainability?

CPU stalls: bad for performance, but good for sustainability. This work profiles CPU stalls happening in the FrontEnd, BackEnd, and branch-related pipeline bubbles. The FrontEnd is responsible for fetching instructions and decoding them into low-level micro-ops (μOps). The BackEnd includes μOp allocation and execution, including stalls caused by data accesses. Branch stalls are accounted for when a μOp get canceled before retiring due to mispredicted branches.

Fig. 7 shows that only 20–27% CPU cycles are effectively utilized for retiring instructions on cores$_{LCApp}$. This aligns with the fact that those applications have <1 IPC (shown Fig. 1) which accounts for only ~20% of the theoretical execution bandwidth of a Skylake CPU core, whose theoretical peak IPC is 5.0. On the other hand, busy-polling core$_{Infra}$ retires instructions during 89% of its execution cycles.

CPU stalls are the main reason for low effective CPU utilization of cores$_{LCApp}$. Fig. 3, Fig. 4 and Fig. 6 illustrate that MemCacheD, Redis and RocksDB suffer from 25–45% FrontEnd stalls, 26–40% BackEnd stalls and 4–6% Branch stalls, respectively. However, Poll has few stalls, which explains its much higher IPC when compared with core$_{LCApp}$.

Stall breakdown analysis. These stalls can be broken down further. Our experiments in Fig. 5 show that L1 MPKIs are as high as 17, while production-level applications are more complex and can exceed 50 [47]. Recently, applications have become more complex with deeper layering abstractions. For example, instruction footprints have grown to be 100× larger than the size of an L1 instruction cache [10], which creates long stalls and interference in the cache hierarchy. Therefore, CPUs waste cycles in the FrontEnd waiting for codes required by those applications. In contrast, Poll has only 0.01 MPKI, as shown in Fig. 5. As for BackEnd stalls, the waiting times due to L2, LLC, main memory or SSD accesses are all possible contributors. Unlike those cores$_{LCApp}$, core$_{Infra}$ running polling threads barely suffer from L2 or lower-level memory data misses, thus its BackEnd stalls are negligible.

The Branch stalls of Poll only account for 0.01% of CPU cycles. The polling thread sits in a tight loop checking RX queues, which spins in a `While` loop whose branch is almost always taken. Infrastructure threads usually exhibit the following features: (1) independence from other application or kernel threads, e.g., not much synchronization, (2) lightweight code block and simplicity, and (3) repetitiveness in terms of the work they do.

Therefore, unlike cores$_{LCApp}$ that go through complicated branching routines and rely on huge instruction and data sets, cores$_{Infra}$ do not suffer from many stalls.

Instructions per second deviations. Due to the significant difference in IPCs, the rate of aging among core$_{Infra}$ and core$_{LCApp}$ may be drastically different, and the inherent characteristics of infrastructure threads can exacerbate this aging gap.

Take branch instructions in Fig. 8 as an example; nearly all programs need branch-related microarchitectural components such as branch predictors and ALUs. We find that polling exercises up to 23× more branch instructions per second. This implies that, besides higher IPC, the density of branch instructions in polling is also higher than those in core$_{LCApp}$. This is true because the code block of each polling `While` loop is relatively simple and independent.

Worse, the number of branch instructions per second can further increase with higher core frequency. Up to 875 million instructions are executed per second when core$_{Infra}$ is running at 3.67GHz. We will explore instruction types other than branches in future work.

3.2 State-of-the-Art OS Characterization

Sec. 3.1 shows microarchitectural characteristics, i.e., IPC, L1 MPKI, CPU stalls and instructions per second, on the core that runs Poll microbenchmark. In this section, we measure those metrics of core$_{Infra}$ running a state-of-the-art OS, Demikernel [50]. We evaluate core$_{Infra}$ that executes TCP and UDP stacks, which involve polling, scheduling and dispatching packets. Table 1 shows that this end-to-end system setting corroborates the observations we summarized in Sec. 3.1. IPCs of core$_{Infra}$ are 2-6.275× higher than

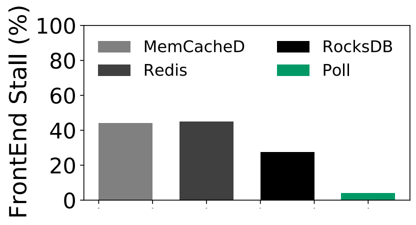

Figure 3: FrondEnd CPU stalls.

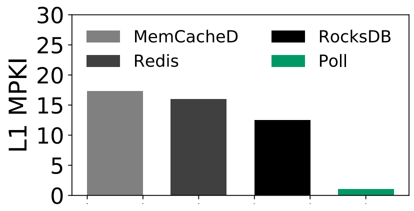

Figure 5: L1 miss rate.

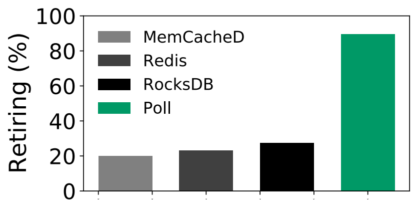

Figure 7: Effective CPU utilization.

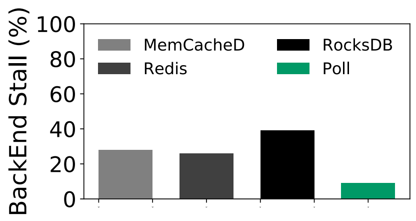

Figure 4: BackEnd CPU stalls.

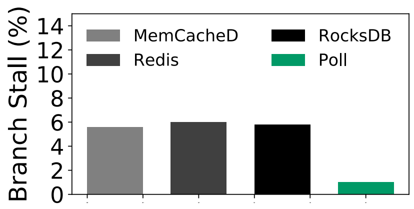

Figure 6: Branch CPU stalls

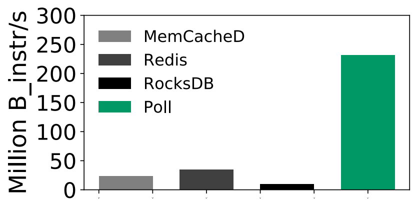

Figure 8: Branch instructions per second on one core@1GHz.

Table 1: Microarchitectural characterizations on DPDK-based Demikernel network stacks [50] (TCP and UDP) vs. Poll benchmark (Sec. 3.1). All cores@3.67GHz).

Metric	TCP	UDP	Poll
IPC	2.51	2.03	3.2
L1 MPKI	1.28	5.32	0.01
Branch Instructions	1.1 G/s	1.2 G/s	0.9 G/s
FrontEnd Stall	8.2%	16.1%	3.6%
BackEnd Stall	23.6%	29.3%	9.5%
Branch Stall	1.7%	4.3%	0.3%
Retiring	66.5%	50.3%	86.7%

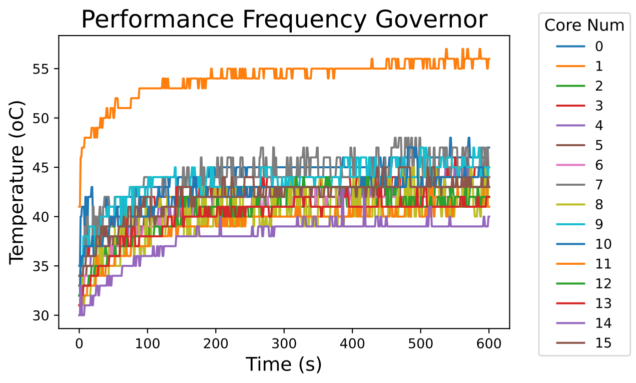

Figure 9: Per-core temperature measurement.

applications shown in Fig. 1. L1 MPKI of the core$_{Infra}$ are much higher than Poll evaluated in Fig. 5, but still much lower than those of applications. The number of branch instructions per second consistently remains at an intense level, no less than 1.26× higher than Poll core$_{Infra}$ @3.67Hz.

3.3 Heat Hot Spots

In addition to the unevenness of IPC and instruction per second, we find that core$_{Infra}$ also incurs heat hot spots on a CPU die. We ran a TCP echo benchmark on Demikernel using its DPDK network stack over 10 minutes. We measured per-core temperature through Intel RAPL tool, whose results are shown in Fig. 9. The temperature of Core1 where Demikernel was pinned for network processing reaches 53-58$^{\circ}C$, which generates more heat than any other core.

4 DESIGN OPTIONS

This work proposes a few design options for sustainability-aware CPU affinity, including a new metric, OS designs, and system architecture designs.

4.1 A New Metric: WTTF

Worst Time to Failure Modeling. Instead of predicting the lifetime of each core through modeling based on thermal/transistor states, we estimate the worst time to failure (WTTF). Each instruction does not necessarily wear out the same path or accumulate the same number of transitions on the same path. However, in the worst case, as instructions accumulate and whose number reaches a threshold, the core can be considered as a core that is not 100% reliable. Such unreliable cores can potentially deliver wrong results and cannot be fully trusted if running critical code blocks [22]. We use Equ. 1 to calculate WTTF at runtime.

$$WTTF = \sum_{n=1}^{\infty} IPC_{tn} \times freq_{tn} \times t_n \tag{1}$$

Where $IPC_{tn} \times freq_{tn}$ gives the number of instructions running on a core per second; within a time window t_n when the core is running at a frequency of $freq_{tn}$ and the pinned thread on the core's microarchitecture exhibits IPC_{tn}, the total number of instructions, in the worst case, have worn out the weakest path of this core IPC_{tn} $\times$ $freq_{tn} \times t_n$ times.

Instead of figuring out a concrete threshold that represents WTTF, which is difficult to estimate accurately, we focus on balancing the *relative* WTTF between cores.

4.2 OS Perspectives

Sustainability-aware OS thread scheduling. OS-level solutions would help balance WTTFs of cores$_{Infra}$ and cores$_{LCApp}$.

There are two options for OS scheduler designs, based on either user-defined or automated solution. First, we can enable a transparent interface that augments the flexibility of existing interface implementing thread binding, e.g., Linux `cpuset`. At runtime, the new sustainability-aware interface allows dynamic user-defined binding between threads within a specified `cpuset` and cores that age at uneven pace. Second, the OS can migrate and swap affinitized threads from one core that has significantly aged or thermally loaded to another core. Since we expect the age-oriented scheduling happens infrequently, e.g., monthly or yearly, there is a considerable design space which allows us to trade scheduling overheads for consistency, robustness and complete live management during migration or swapping.

Affinity-aware thermal management. The heat asymmetry of cores$_{Infra}$ and cores$_{LCApp}$ due to CPU affinity requires new thermal management policies and mechanisms. For controlling hot spots, one can either reduce the heat production of cores$_{Infra}$, or balance heat production across cores$_{Infra}$ and cores$_{LCApp}$. The former can benefit from existing mechanisms such as Dynamic Voltage and Frequency Scaling (DVFS). We leave it for future work to understand the trade-off between energy efficiency, lifetime, and tail latency [8] when managing cores$_{Infra}$ through DVFS. For instance, at low load, lowering the frequency of cores$_{Infra}$ is beneficial to save energy and reduce heat dissipation, if tail latency is negligibly affected. The latter can leverage thermal-aware mechanisms such as thread migrations and execution throttling [13, 16].

4.3 System Architecture Perspectives

Sustainability-oriented isolation. Alternatively, it is also possible to pin those threads with highly skewed aging speeds in different server processors, and group threads with similar aging speeds in the same physical server. This deployment architecture fundamentally eliminates the need of tracking each core's aging within each server and simplifies maintenance. One option is to isolate core$_{Infra}$ on PCIe-attached DPU [5] or IPU [4] which acts as an embedded processor and is independent of core$_{LCApp}$ located on CPUs. This isolation opens up new motivations for isolating infrastructure and application threads other than reasons such as interference removal. The other option is to isolate core$_{Infra}$ and core$_{LCApp}$ on different CPU servers. For instance, sidecores used for high-performance virtualization can be placed in a remote server [28]. Further, core$_{Infra}$ CPU servers can use cheaper CPUs with lower cost since core$_{Infra}$ wears out much faster.

Lifetime-aware co-location. Due to CPU stalls, core$_{LCApp}$ has much longer lifetimes than core$_{Infra}$. Filling the portion of core$_{LCApp}$ left unused by LC tasks with batch processing applications is another way to balance the lifetime between core$_{LCApp}$ and core$_{Infra}$. This approach has been employed in some proposals for high CPU utilization while maintaining predictable tail latency [20, 34, 35, 52]. For example, Microsoft Bing co-locates LC and batch jobs on over 90,000 servers [23], and the median machine in compute clusters at Google hosts 8 applications [51].

5 RELATED WORK

Some of the prior work focuses on the lifetime of cores$_{LCApp}$. Donald et al. [16] emphasize heat balancing on a CPU dies and address the thermal asymmetry among cores that run application threads. ExtraTime [37] provides an aging analysis framework to use transistor-level variables to predict aging effects. Facelift [48] offers aging-aware scheduling among applications and cores to configure the lifetime of a multicore CPU processor. On the other hand, our paper positions the different characteristics between core$_{LCApp}$ and core$_{Infra}$ and is orthogonal to core$_{LCApp}$ lifetime extensions.

Early proposals also advocate leveraging IPC as the metric to improve energy efficiency [21, 33]. Instead, we characterize IPCs between core$_{LCApp}$ and core$_{Infra}$, and use it as an indication to estimate lifetime. Furthermore, we also introduce instructions per second, accompanied with IPCs, to estimate cores' sustainability.

Polling threads that require CPU affinity have been explored by some researchers considering tail latency and energy efficiency [17, 21, 43]. They propose amortizing the polling-induced performance and energy overhead by tuning core frequency [2, 43]. This paper extends the focus of CPU affinity beyond polling. We find the common microarchitectural features across different infrastructure threads that demand CPU affinity. In addition, our work sheds light on CPU core lifetime in the context of CPU affinity.

6 CONCLUSIONS

Based on our characterizations, core$_{Infra}$ are aging much faster than other cores due to sustainability-unaware CPU affinity. Our work advocates distinguishing affinitized core$_{Infra}$ from other cores to avoid uneven core-aging and heat hot spots, which lead to more frequent replacement and higher embodied carbon emissions. In addition, we call for alternative approaches beyond traditional per-core frequency or voltage tuning [27, 34, 43, 48] to narrow the aging speed gap. Design options that we offer include OS designs such as instruction intensity aware balancing, heat balancing, thread isolation and co-location.

ACKNOWLEDGMENTS

We would like to thank our anonymous reviewers for their valuable comments and helpful feedback. This work was supported by the Natural Science and Engineering Research Council of Canada, National Science Foundation (NSF) grant CNS-210458, NSF Graduate Research Fellowships Program, a Canada Research Chair, the Canadian Foundation for Innovation, VMware and Cisco Systems.

REFERENCES

[1] [n. d.]. ADATA PCIe Gen 5.0 4 SSDs, Project Nighthawk. https://www.adata.com/en/news/960.

[2] [n. d.]. Data Plane Development Kit. https://www.dpdk.org.

[3] [n. d.]. Ethernet Alliance. https://ethernetalliance.org/technology/2020-roadmap/.

[4] [n. d.]. Intel Infrastructure Processing Unit (IPU) and SmartNICs. https://www.intel.com/content/www/us/en/products/network-io/smartnic.html.

[5] [n. d.]. NVidia BlueField 3 DPU. https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/documents/datasheet-nvidia-bluefield-3-dpu.pdf.

[6] [n. d.]. Storage performance development kit (SPDK). http://www.spdk.io/.

[7] Alaa R. Alameldeen and David A. Wood. 2006. IPC considered harmful for multiprocessor workloads. *IEEE Micro* 26, 4 (2006), 8–17.

[8] Thomas Anderson, Adam Belay, Mosharaf Chowdhury, Asaf Cidon, and Irene Zhang. 2022. Treehouse: A case for carbon-aware datacenter software. In *Proceedings of the 2nd Workshop on Sustainable Computer Systems (HotCarbon).*

[9] Grant Ayers, Jung Ho Ahn, Christos Kozyrakis, and Parthasarathy Ranganathan. 2018. Memory hierarchy for web search. In *Proceedings of the 24th IEEE International Symposium on High Performance Computer Architecture (HPCA).* 643–656.

[10] Grant Ayers, Nayana Prasad Nagendra, David I. August, Hyoun Kyu Cho, Svilen Kanev, Christos Kozyrakis, Trivikram Krishnamurthy, Heiner Litz, Tipp Moseley, and Parthasarathy Ranganathan. 2019. Asmdb: understanding and mitigating front-end stalls in warehouse-scale computers. In *Proceedings of the 46th IEEE/ACM International Symposium on Computer Architecture (ISCA).* 462–473.

[11] 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 ACM Symposium on Operating Systems Design and Implementation (OSDI).* 49–65.

[12] Kerry Bernstein, David J. Frank, Anne E. Gattiker, Wilfried Haensch, Brian L. Ji, Sani R. Nassif, Edward J. Nowak, Dale J. Pearson, and Norman J. Rohrer. 2006. High-performance CMOS variability in the 65-nm regime and beyond. *IBM journal of research and development* 50, 4.5 (2006), 433–449.

[13] David Brooks and Margaret Martonosi. 2001. Dynamic thermal management for high-performance microprocessors. In *Proceedings of the 7th IEEE International Symposium on High-Performance Computer Architecture (HPCA).* 171–182.

[14] Shuang Chen, Christina Delimitrou, and Jose F. Martinez. 2019. PARTIES: QoS-Aware Resource Partitioning for Multiple Interactive Services. In *Proceedings of the 24th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS).* 107–120.

[15] Alexandros Daglis, Mark Sutherland, and Babak Falsafi. 2019. RPCValet: NI-Driven Tail-Aware Balancing of μs-Scale RPCs. In *Proceedings of the 24th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS).* 35–48.

[16] James Donald and Margaret Martonosi. 2006. Techniques for multicore thermal management: Classification and new exploration. *ACM SIGARCH computer architecture news* 34, 2 (2006), 78–88.

[17] Eric Dumazet. 2017. Busy polling: Past, present, future. In *Netdev Conference*, Vol. 2.

[18] Michael Ferdman, Almutaz Adileh, Onur Kocberber, Stavros Volos, Mohammad Alisafaee, Djordje Jevdjic, Cansu Kaynak, Adrian Daniel Popescu, Anastasia Ailamaki, and Babak Falsafi. 2012. Clearing the clouds: a study of emerging scale-out workloads on modern hardware. *Acm sigplan notices* 47, 4 (2012), 37–48.

[19] Daniel Firestone, Andrew Putnam, Sambhrama Mundkur, Derek Chiou, Alireza Dabagh, Mike Andrewartha, Hari Angepat, Vivek Bhanu, Adrian Caulfield, Eric Chung, Harish Kumar Chandrappa, Somesh Chaturmohta, Matt Humphrey, Jack Lavier, Norman Lam, Fengfen Liu, Kalin Ovtcharov, Jitu Padhye, Gautham Popuri, Shachar Raindel, Tejas Sapre, Mark Shaw, Gabriel Silva, Madhan Sivakumar, Nisheeth Srivastava, Anshuman Verma, Qasim Zuhair, Deepak Bansal, Doug Burger, Kushagra Vaid, David A. Maltz, and Albert Greenberg. 2018. Azure accelerated networking: SmartNICs in the public cloud. In *Proceedings of the 15th USENIX Symposium on Networked Systems Design and Implementation (NSDI).* 51–66.

[20] Joshua Fried, Zhenyuan Ruan, Amy Ousterhout, and Adam Belay. 2020. Caladan: Mitigating interference at microsecond timescales. In *Proceedings of the 14th USENIX Symposium on Operating Systems Design and Implementation (OSDI).* 281–297.

[21] Hossein Golestani, Amirhossein Mirhosseini, and Thomas F. Wenisch. 2019. Software data planes: You can't always spin to win. In *Proceedings of the ACM Symposium on Cloud Computing (SoCC).* 337–350.

[22] Peter H. Hochschild, Paul Turner, Jeffrey C. Mogul, Rama Govindaraju, Parthasarathy Ranganathan, David E. Culler, and Amin Vahdat. 2021. Cores that don't count. In *Proceedings of the ACM Workshop on Hot Topics in Operating Systems (HotOS).* 9–16.

[23] Călin Iorgulescu, Reza Azimi, Youngjin Kwon, Sameh Elnikety, Manoj Syamala, Vivek Narasayya, Herodotos Herodotou, Paulo Tomita, Alex Chen, Jack Zhang, and Junhua Wang. 2018. Perfiso: Performance isolation for commercial latency-sensitive services. In *Proceedings of the 2018 USENIX Annual Technical Conference (ATC).* 519–532.

[24] Kostis Kaffes, Timothy Chong, Jack Tigar Humphries, Adam Belay, David Mazières, and Christos Kozyrakis. 2019. Shinjuku: Preemptive scheduling for μsecond-scale tail latency. In *Proceedings of the 16th USENIX Symposium on Networked Systems Design and Implementation (NSDI).* 345–360.

[25] Anuj Kalia, Michael Kaminsky, and David Andersen. 2019. Datacenter RPCs can be general and fast. In *Proceedings of the 16th USENIX Symposium on Networked Systems Design and Implementation (NSDI).* 1–16.

[26] Svilen Kanev, Juan Pablo Darago, Kim Hazelwood, Parthasarathy Ranganathan, Tipp Moseley, Gu-Yeon Wei, and David Brooks. 2015. Profiling a warehouse-scale computer. In *Proceedings of the 42nd IEEE/ACM International Symposium on Computer Architecture (ISCA).* 158–169.

[27] Harshad Kasture, Davide B. Bartolini, Nathan Beckmann, and Daniel Sanchez. 2015. Rubik: Fast analytical power management for latency-critical systems. In *Proceedings of the 48th IEEE/ACM International Symposium on Microarchitecture (MICRO).* 598–610.

[28] Yossi Kuperman, Eyal Moscovici, Joel Nider, Razya Ladelsky, Abel Gordon, and Dan Tsafrir. 2016. Paravirtual remote i/o. In *Proceedings of the 21st ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS).* 49–65.

[29] Dongup Kwon, Junehyuk Boo, Dongryeong Kim, and Jangwoo Kim. 2020. FVM: FPGA-assisted Virtual Device Emulation for Fast, Scalable, and Flexible Storage Virtualization. In *Proceedings of the 14th USENIX Symposium on Operating Systems Design and Implementation (OSDI).* 955–971.

[30] Kevin M. Lepak, Harold W. Cain, and Mikko H. Lipasti. 2003. Redeeming ipc as a performance metric for multithreaded programs. In *Proceedings of the 12th ACM International Conference on Parallel Architectures and Compilation Techniques (PACT).* 232–243.

[31] Huaicheng Li, Mingzhe Hao, Stanko Novakovic, Vaibhav Gogte, Sriram Govindan, Dan R.K. Ports, Irene Zhang, Ricardo Bianchini, Haryadi S. Gunawi, and Anirudh Badam. 2020. Leapio: Efficient and portable virtual nvme storage on arm socs. In *Proceedings of the 25th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS).* 591–605.

[32] Hyeontaek Lim, Dongsu Han, David G. Andersen, and Michael Kaminsky. 2014. MICA: A Holistic Approach to Fast In-Memory Key-Value Storage. In *Proceedings of the 11th USENIX Symposium on Networked Systems Design and Implementation (NSDI).* 429–444.

[33] Ming Liu, Simon Peter, Arvind Krishnamurthy, and Phitchaya Mangpo Phothilimthana. 2019. E3: energy-efficient microservices on SmartNIC-accelerated servers. In *Proceedings of the 2019 USENIX Annual Technical Conference (ATC).* 363–378.

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

[35] Artemiy Margaritov, Siddharth Gupta, Rekai Gonzalez-Alberquilla, and Boris Grot. 2019. Stretch: Balancing qos and throughput for colocated server workloads on smt cores. In *Proceedings of the 25th IEEE International Symposium on High Performance Computer Architecture (HPCA).* 15–27.

[36] Rajesh Nishtala, Hans Fugal, Steven Grimm, Marc Kwiatkowski, Herman Lee, Harry C. Li, Ryan McElroy, Mike Paleczny, Daniel Peek, Paul Saab, Tony Stafford, David Tung, and Venkateshwaran Venkataramani. 2013. Scaling memcache at facebook. In *Proceedings of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI).* 385–398.

[37] Fabian Oboril and Mehdi B. Tahoori. 2012. Extratime: Modeling and analysis of wearout due to transistor aging at microarchitecture-level. In *Proceedings of the IEEE/IFIP International Conference on Dependable Systems and Networks (DSN).* 1–12.

[38] 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 Symposium on Networked Systems Design and Implementation (NSDI).* 361–378.

[39] John Ousterhout, Arjun Gopalan, Ashish Gupta, Ankita Kejriwal, Collin Lee, Behnam Montazeri, Diego Ongaro, Seo Jin Park, Henry Qin, Mendel Rosenblum, Stephen Rumble, Ryan Stutsman, and Stephen Yang. 2015. The RAMCloud storage system. *ACM Transactions on Computer Systems (TOCS)* 33, 3 (2015), 1–55.

[40] Bo Peng, Haozhong Zhang, Jianguo Yao, Yaozu Dong, Yu Xu, and Haibing Guan. 2018. MDev-NVMe: A NVMe Storage Virtualization Solution with Mediated Pass-Through. In *2018 USENIX Annual Technical Conference (ATC).* 665–676.

[41] 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 Transactions on Computer Systems (TOCS)* 33, 4

(2015), 1–30.

[42] George Prekas, Marios Kogias, and Edouard Bugnion. 2017. Zygos: Achieving low tail latency for microsecond-scale networked tasks. In *Proceedings of the 26th ACM Symposium on Operating Systems Principles (SOSP)*. 325–341.

[43] 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 6th ACM Symposium on cloud Computing (SoCC)*. 342–355.

[44] Henry Qin, Qian Li, Jacqueline Speiser, Peter Kraft, and John Ousterhout. 2018. Arachne: core-aware thread management. In *Proceedings of the 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*. 145–160.

[45] Adam Ruprecht, Danny Jones, Dmitry Shiraev, Greg Harmon, Maya Spivak, Michael Krebs, Miche Baker-Harvey, and Tyler Sanderson. 2018. VM Live Migration At Scale. In *Proceedings of the 14th ACM S International Conference on Virtual Execution Environments (VEE)*. 45–56.

[46] Akshitha Sriraman and Abhishek Dhanotia. 2020. Accelerometer: Understanding acceleration opportunities for data center overheads at hyperscale. In *Proceedings of the 25th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. 733–750.

[47] Akshitha Sriraman, Abhishek Dhanotia, and Thomas F. Wenisch. 2019. Softsku: Optimizing server architectures for microservice diversity@ scale. In *Proceedings of the 46th IEEE/ACM International Symposium on Computer Architecture (ISCA)*. 513–526.

[48] Abhishek Tiwari and Josep Torrellas. 2008. Facelift: Hiding and slowing down aging in multicores. In *Proceedings of the 41st IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 129–140.

[49] Ziye Yang, Changpeng Liu, Yanbo Zhou, Xiaodong Liu, and Gang Cao. 2018. Spdk vhost-nvme: Accelerating i/os in virtual machines on nvme ssds via user space vhost target. In *Proceedings of IEEE 8th International Symposium on Cloud and Service Computing (SC2)*. 67–76.

[50] 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 Pennar, Max Demoulin, Piali Choudhuryr, and Anirudh Badam. 2021. The demikernel datapath os architecture for microsecond-scale datacenter systems. In *Proceedings of the ACM 28th Symposium on Operating Systems Principles (SOSP)*. 195–211.

[51] Xiao Zhang, Eric Tune, Robert Hagmann, Rohit Jnagal, Vrigo Gokhale, and John Wilkes. 2013. CPI2: CPU performance isolation for shared compute clusters. In *Proceedings of the 8th ACM European Conference on Computer Systems (EuroSys)*. 379–391.

[52] Jiechen Zhao, Natalie Enright Jerger, and Mingyu Gao. 2021. What can chiplets bring to multi-tenant clouds?. In *Cloud Workshop at Proceedings of the 54th IEEE/ACM International Symposium on Microarchitecture (MICRO)*.

[53] Jiechen Zhao, Iris Uwizeyimana, Karthik Ganesan, Mark C. Jeffrey, and Natalie Enright Jerger. 2022. Altocumulus: Scalable Scheduling for Nanosecond-scale Remote Procedure Calls. In *Proceedings of the 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 423–440.