Giving Old Servers New Life at Hyperscale

Jaylen Wang
Carnegie Mellon University
jaylenw@andrew.cmu.edu

Udit Gupta
Cornell University
ugupta@cornell.edu

Akshitha Sriraman
Carnegie Mellon University
akshitha@cmu.edu

I. Introduction

Scientists have made it clear that anthropogenic climate change is one of the greatest threats to both global health and the planet's ecosystem [7], [8]. To mitigate climate change, systems researchers have a particular role to play, as recent studies estimate that 2% of global emissions are due to Information and Communication Technology (ICT) [16]. Critically, projections show that ICT could constitute 20% of anthropogenic emissions by 2030, with much of the increase attributable to an increased demand for cloud computing [15].

To reduce emissions and waste, sustainable resource management has focused on ways to reduce, reuse, and recycle common resources such as oil and plastics; similar strategies are understudied for managing data center server hardware, where current practice is to dispose of server hardware every three to four years [9]. Minimizing hardware waste is especially crucial; previous work has shown that up to 50% of data center system emissions are "embodied" emissions, which result from the manufacturing and transport of hardware [14].

Prior work has highlighted the issue of embodied emissions and the potential benefits of extending data center server lifetimes [13], [14]. However, few works have focused on practical methods for extending data center server lifetimes.

We take a step towards understanding how older hardware can be reused in a data center while preserving end-to-end service performance (i.e., tail latency). We focus on performance as it is a key driving factor behind server refreshes, along with other factors such as marketing considerations, failure rates, etc. We particularly focus on the modern microservice paradigm, wherein a complex web service is composed of numerous distributed microservices such as HTTP connection termination, key-value serving, query rewriting, and protocol routing [17]. Since the microservice paradigm improves web service development and scalability [12], [19], it has been adopted by several companies such as Amazon [1], Netflix [5], Gilt [3], LinkedIn [4], and SoundCloud [6].

We analyze the impact on performance when running all and parts of a microservice-based application on older server generations. From our results, we find operating regions where older hardware does not reduce service performance compared to running on newer servers. For example, we observe that some older hardware can still achieve performance Service Level Objectives (SLOs) under lower request loads. We also find that certain microservices and regions of a microservice call graph are more tolerant to being run on older hardware.

	Intel		AMD	
	Xeon E5-2660 v2	**Xeon E5-2660 v3**	**EPYC 7542**	**EPYC 7543**
Microarchitecture	Ivy Bridge (2012)	Haswell (2013)	Rome (2019)	Milan (2021)
Cores/Threads	10/20	10/20	32/64	32/64
Node	22 nm	22 nm	7 nm	7 nm
Base/Turbo (GHz)	2.2 / 3	2.6 / 3.3	2.9 / 3.4	2.8 / 3.7
LLC Cache Size	25 MB	25 MB	128 MB	256 MB
TDP (W)	95	105	225	225
RAM (DDR4)	256GB (1.6 GHz)	160GB (2.133 GHz)	256GB (3.2 GHz)	512GB (3.2 GHz)
Disk (SATA)	2 TB HDD	480 GB SSD	1.6 TB SSD	2 TB SSD
NIC	10Gb (PCIe v3)	10 Gb (PCIe v3)	25 Gb (PCIe v4.0)	25 Gb (PCIe v4.0)

TABLE I
Characteristics of two generations (old on the left, new on the right) of Intel and AMD servers used in our experiments.

We show that there are "carbon inefficiencies" in current data center resource management strategies, and better carbon efficiency can be achieved by scheduling on older hardware when appropriate. We motivate the need for new microservice scheduling strategies that consider hardware generations' embodied and operational characteristics. Our work motivates a scheduling system that leverages microservices' performance tolerance to older hardware generations to prevent environmentally costly server refreshes and hardware waste.

II. Characterizing Server Generations

We characterize how common data center services perform on different hardware generations.

Characterization methodology. We compare microservice-based applications' performance on two AMD and Intel server generations. The servers are of the same SKU and only differ by their generation. The servers are located in CloudLab data centers [10]. Their characteristics are summarized in Table I.

We show results for DeathStarBench's `Social Network` service [11], which allows users to create posts with text and images that are processed. This functionality is implemented as thirty core microservices that communicate via Apache Thrift Remote Procedure Calls [2]. For brevity, we omit results for a different DeathStarBench application, `Hotel Reservation` which displays similar results and insights.

Our setup allows us to evenly distribute the thirty microservices in `Social Network` across the number of nodes under test, pin each microservice to a single socket, and keep microservice placements constant. In a given server, we also constrain each microservice to the same amount of RAM. For all experiments, we use an open-loop load generator and sweep across low to high queries-per-second (QPS) conditions until a saturation throughput is reached, while recording tail latencies.

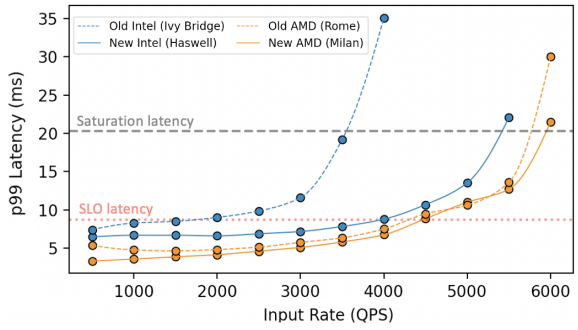

Fig. 1. Tail (99th%) latency across different load conditions (in QPS) for older and newer Intel and AMD server SKUs. Older servers satisfy the SLO at certain low load conditions.

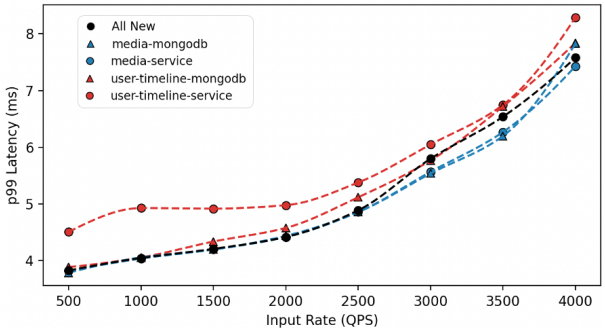

Fig. 2. Impact on end-to-end service tail (99th%) latency upon placing a certain microservice on older hardware. Certain microservices are more (e.g., *media-service*) and less (e.g., *user-timeline-service*) tolerant to placement on older hardware.

A. End-to-End Service Characterization Across Generations

We first study how an end-to-end service behaves on different server generations. We distribute `Social Network`'s thirty microservices across fifteen nodes of the same server type (i.e., same SKU and generation).

Fig. 1 shows the resulting plot of latency vs. QPS. Latency above the gray dotted line cannot meaningfully be measured as the system is under saturation, where queuing delays grow unbounded. The red dotted line shows a performance-guided SLO target taken as the latency achieved at 75% of the saturation load for the best performing server ("New AMD").

We find that at certain lower load conditions, both of the older servers can still achieve the SLO (often within a latency margin of 2–3 milliseconds). This result indicates that using newer servers to serve lower load conditions is a carbon inefficiency, as older servers can still achieve SLO targets. Hence, it is worth exploring if scheduling services on older hardware under lower loads can save embodied emissions.

B. Microservice-Based Characterization Across Generations

While Fig. 1 shows the end-to-end service's performance when running on older hardware, it does not show a specific microservice's performance tolerance. Studying fine-grained microservice-level placement strategies that minimize performance impact might reveal further carbon optimization opportunities.

To this end, we conduct a set of experiments where all microservices are initially placed on fifteen newer servers under the same experimental setup as in §II-A. We then place one microservice on an older server, while keeping all other microservices on the newer nodes. We perform a QPS sweep and repeat for each of the thirty microservices. We report our results on only the two generations of AMD nodes for brevity.

In Fig. 2, we show the results for a set of representative microservices (those on the same call path are colored the same). Other microservices show similar trends; we omit them for brevity. We compare the performance of placing a specific microservice on older hardware to an "All New" configuration, indicating the impact on end-to-end service latency.

Certain microservices, such as *user-timeline-service* and *user-timeline-mongodb*, consistently exhibit higher latencies when placed on older hardware compared to "All New", indicating lower tolerance. Conversely, microservices such as *media-service* and *media-mongodb* are more tolerant. The *media* microservices even perform better at high loads, since they run without contention on an older machine. While the microservice model's modularity is often exploited for performance, the imbalance between microservices in tolerances to older hardware suggests that optimizing individual microservice scheduling can improve carbon efficiency.

We also find that microservices on the same call path exhibit similar tolerances to placement on older hardware, regardless of their underlying functionality. This observation suggests that a microservice's location in the call graph can determine its tolerance to older hardware. Further research is needed to determine the specific features of call graph regions that make them more amenable to placement on older nodes.

III. System Implications and Carbon Accounting

To reduce carbon emissions and waste, we can use observations from §II to develop an end-to-end system that optimizes scheduling and placement strategies, by balancing performance and carbon tradeoffs to extend server lifetimes. Such a system can incorporate (1) offline profiling, similar to §II, to inform an initial carbon-efficient placement and (2) online monitoring, to ensure that latency SLOs are met.

Accurately evaluating such a scheduler's policies requires accounting for operational and embodied emission reductions. To measure operational emissions, we can track power usage. However, measuring embodied emissions is more challenging. Prior work used lifetime as a measure for embodied carbon, but their methods do not consider how performance impacts lifetime [13], [18]. To address this challenge, we propose a performance-aware lifetime model that considers the ability to achieve SLOs for individual service components (e.g., a group of microservices). This approach accounts for a server's ability to provide useful computing power, even if only for a region of a microservice call graph. We will expand our study to include further server generations, to better reason about lifetimes and perform carbon accounting for data center hardware.

REFERENCES

[1] "Amazon," https://gigaom.com/2011/10/12/419-the-biggest-thing-amazon-got-right-the-platform/.

[2] "Apache Thrift - Home." [Online]. Available: https://thrift.apache.org/

[3] "Gilt," www.infoq.com/presentations/scale-gilt.

[4] "Linkedin," www.infoq.com/presentations/linkedin-microservices-urn.

[5] "Netflix," www.nginx.com/blog/microservices-at-netflix-architectural-best-practices/.

[6] "Soundcloud," https://developers.soundcloud.com/blog/building-products-at-soundcloud-part-1-dealing-with-the-monolith.

[7] *Climate Change 2021: The Physical Science Basis. Contribution of Working Group I to the Sixth Assessment Report of the Intergovernmental Panel on Climate Change*. Cambridge, United Kingdom and New York, NY, USA: Cambridge University Press, 2021.

[8] L. Atwoli, A. H. Baqui, T. Benfield, R. Bosurgi, F. Godlee, S. Hancocks, R. Horton, L. Laybourn-Langton, C. A. Monteiro, I. Norman, K. Patrick, N. Praities, M. G. Olde Rikkert, E. J. Rubin, P. Sahni, R. Smith, N. Talley, S. Turale, and D. Vázquez, "Call for Emergency Action to Limit Global Temperature Increases, Restore Biodiversity, and Protect Health," *New England Journal of Medicine*, no. 12, Sep. 2021.

[9] L. A. Barroso, J. Clidaras, and U. Hölzle, "The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines, Second edition," *Synthesis Lectures on Computer Architecture*, Jul. 2013.

[10] D. Duplyakin, R. Ricci, A. Maricq, G. Wong, J. Duerig, E. Eide, L. Stoller, M. Hibler, D. Johnson, K. Webb, A. Akella, K. Wang, G. Ricart, L. Landweber, C. Elliott, M. Zink, E. Cecchet, S. Kar, and P. Mishra, "The design and operation of cloudlab," in *Proceedings of the 2019 USENIX Conference on Usenix Annual Technical Conference*, ser. USENIX ATC '19. USA: USENIX Association, Jul. 2019.

[11] Y. Gan, Y. Zhang, D. Cheng, A. Shetty, P. Rathi, N. Katarki, A. Bruno, J. Hu, B. Ritchken, B. Jackson, K. Hu, M. Pancholi, Y. He, B. Clancy, C. Colen, F. Wen, C. Leung, S. Wang, L. Zaruvinsky, M. Espinosa, R. Lin, Z. Liu, J. Padilla, and C. Delimitrou, "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*, ser. ASPLOS '19. New York, NY, USA: Association for Computing Machinery, Apr. 2019.

[12] M. J. Greeven, H. Yu, and J. Shan, "Why companies must embrace microservices and modular thinking," *MIT Sloan Management Review*, Summer 2021. [Online]. Available: https://www.proquest.com/scholarly-journals/why-companies-must-embrace-microservices-modular/docview/2555433700/se-2

[13] U. Gupta, M. Elgamal, G. Hills, G.-Y. Wei, H.-H. S. Lee, D. Brooks, and C.-J. Wu, "ACT: designing sustainable computer systems with an architectural carbon modeling tool," in *Proceedings of the 49th Annual International Symposium on Computer Architecture*, ser. ISCA '22. New York, NY, USA: Association for Computing Machinery, Jun. 2022.

[14] U. Gupta, Y. G. Kim, S. Lee, J. Tse, H.-H. S. Lee, G.-Y. Wei, D. Brooks, and C.-J. Wu, "Chasing Carbon: The Elusive Environmental Footprint of Computing," Oct. 2020. [Online]. Available: http://arxiv.org/abs/2011.02839

[15] N. Jones, "How to stop data centres from gobbling up the world's electricity," *Nature*, Sep. 2018.

[16] B. Knowles, *ACM TechBrief: Computing and Climate Change*. Association for Computing Machinery, 2021.

[17] A. Sriraman and T. F. Wenisch, "μTune: Auto-Tuned Threading for OLDI Microservices," in *Proceedings of the 12th USENIX conference on Operating Systems Design and Implementation*, 2018.

[18] J. Switzer, G. Marcano, R. Kastner, and P. Pannuto, "Junkyard computing: Repurposing discarded smartphones to minimize carbon," 2022.

[19] M. Villamizar, O. Garcés, H. Castro, M. Verano, L. Salamanca, R. Casallas, and S. Gil, "Evaluating the monolithic and microservice architecture pattern to deploy web applications in the cloud," in *10CCC*, 2015.