RETROSPECTIVE: Profiling a warehouse-scale computer

Svilen Kanev, Juan Pablo Darago, Kim Hazelwood, Parthasarathy Ranganathan,
Tipp Moseley, Gu-Yeon Wei, David Brooks

We wrote "Profiling a WSC" for ISCA 2015 in order to summarize and share our experiences extracting performance information from Google's datacenter fleet. The paper outlined performance bottlenecks and trends that we observed, both in software and hardware, and sketched out a handful of areas for future research. It was intentionally light on solutions and meant to inspire directions for others' work. Almost ten years later, we reflect on the predictions the paper made, check back on which ones panned out, and which of the trends have caught on in industry or academia.

Impact and contributions

Datacenter tax. Arguably the biggest contribution of the paper was coining the term "datacenter tax" – referring to the non-application logic required to run distributed services[1]. Since the paper was published, many others have validated it as an industry-wide phenomenon and used the term to justify reducing these overheads. As just one example, multiple key services in Meta datacenters were shown to spend 50+% of their compute cycles in "tax" code [20].

Hardware "tax" optimizations. We initially proposed classifying functions as "datacenter tax" mostly based on how easy they would be to accelerate in hardware (this is ISCA after all!) – this is why we opted for broadly applicable, low-level code, that is relatively mature and self-contained. Multiple follow-up efforts have taken up designing accelerators for different tax routines. These have included both commercial designs – e.g. Intel's Infrastructure Processing Unit (which accelerates compression and encryption) [8]; as well as open-source academic ones – accelerators for protocol buffers [10], compression [11], memory allocation [9].

One takeaway from these efforts is that *broad* acceleration of "tax"-like routines is qualitatively different from more traditional *deep* acceleration. Accelerating an ensemble of different bottlenecks, worth 5% each, is very different from speeding up the whole application "deeply". Not only does it take more effort, but the potential gains are smaller by definition. It also requires very careful thinking about accelerator placement and communication requirements, and enforces much tighter budgets (in area/power) due to the limited opportunity.

Software "tax" optimizations. While our main motivation for classifying cycles as "datacenter tax" initially was hardware accelerators, we have since realized that optimizing tax functions in software can be incredibly fruitful. Due to these low-level functions' ubiquity, a small number of optimization experts can focus their attention on "tax" code and release optimizations to thousands of services at a time. If the same engineers were to go binary-by-binary and focus on application logic, they would have to spend a lot of effort on each application's idiosyncrasies, and the sophistication of optimizations could suffer. Focusing on a very small set of shared routines instead allows for very outsized impact[2].

Optimizing datacenter tax in software has been a very concerted effort inside Google's WSCs. As just one example data point, since the paper was written, a gaggle of different optimizations over 3 years has resulted in a nearly 2x reduction in memory allocation cycles. After the relatively low-hanging optimization fruit, the scope of that work has evolved from "reduce time spent in datacenter tax" to "optimize tax routines holistically for application productivity" [6]. These sometimes make tradeoffs that increase tax cycles, but are a net positive because of positive externalities (like reduced TLB pressure).

Workload diversification. Another trend we identified was the increasing workload diversity in warehouse-scale computers. That has continued over the years, with compute cycles spread over more and more services. It has specifically accelerated with the broader usage of public clouds.

Recent trends on machine learning workloads, however, add a bit more nuance to our point about diversification. Since the paper was written in 2014, of course, machine learning has taken off, with an insatiable appetite for compute cycles. Training and serving large models does concentrate cycles on a small number of workloads. However, the models themselves still change so rapidly that, over yearlong periods, we still observe significant workload diversity.

General-purpose CPU trends. The paper also analyzed the performance bottlenecks for general-purpose CPUs in a lot of detail. Two of these deserve special mention.

CPU frontends. While not the first work to bring significant attention to the CPU frontend as an increasingly important bottleneck for scale out workloads [4], "Profiling a WSC" confirmed that trend and was often used to motivate frontend improvements. Some follow-up work has included deeper characterization [3], new frontend structures [13], [17], compiler solutions [15], [16], and the adoption of software prefetching for code as standard in the industry [7].

[1]We also considered "distributed systems tax", but decided against it because components like `memcpy` are prominent in non-distributed code, too. Also, "datacenter tax" was plain more catchy.

[2]Note that this is made possible by Google's single-repository, code-lives-at-head approach to software development [19].

Memory bandwidth vs latency. The paper predicted that memory bandwidth was going to be less important than latency, and that's a trend we got wrong. Since 2014, several factors have contributed to bandwidth becoming a major bottleneck in WSCs. On the one hand, there is more demand for memory bandwidth – due largely to the rise in machine learning, ever growing core counts, and the mainstream adoption of chiplet architectures (which cause more duplication than a monolithic cache). On the other, bandwidth supply is still limited, so bandwidth is becoming ever so scarce.

Does it even matter? When the authors were discussing the contents of this document, an interesting debate formed. One camp argued the original paper was too heavy on microarchitectural details, and that is less and less important in today's environment dominated by system-wide concerns (disk, network, memory bandwidth) and accelerators (GPUs, TPUs). The other camp argued that general-purpose architecture is less glorious, but still continues to make large strides over time. Since the IvyBridges we profiled in 2014, the amount of compute per socket has improved by a factor of 10, through a series of 1% incremental *roofshots*. Eventually, while we all agreed that the paper could have benefited from a more stronger systems focus, we didn't reach a resolution on the role of future microarchitectural innovation – solving philosophical debates is an exercise left to the reader.

Methodology contributions. *Benchmarks.* The paper joined a small choir arguing that using SPEC CPU is not representative of WSC workloads. For a long time we even considered the title *"The datacenter doesn't run SPEC!"*, but eventually (boring) conventional wisdom prevailed. Since then, there has been some exciting work that represents datacenter workloads better – DeathStarBench [5], TailBench [12], Fleetbench [1] and Google memory traces [2]. Unfortunately, it is still extremely common practice to overestimate the predictive power of SPEC CPU and use it outside its intended purpose. Benchmarking is an area where we can learn from the machine learning community: suites like MLPerf [18] are much more representative of production workloads.

Finally, *fleetwide profiling* is considered standard in hyperscalars today – there is little doubt about the benefits of finding optimization opportunities with live traffic. On the lower level, TopDown microarchitecture analysis [21] was novel in 2014, and required lots of introduction in the paper text. It is much more standard, and supported by performance tooling now.

Lessons and observations

In no particular order:

- Workload intuition and deep understanding of workloads is important. Please continue writing and accepting characterization papers!
- As a corollary, running a benchmark suite and getting a score out is easy, but often not the full picture. Especially beware small and non-representative benchmarks [14].
- There is a lot of room to optimize the low-level "tax" overheads. That said, it is important to remember the big picture – a single top-level algorithmic improvement can easily trump years' worth of low-level work.

Acknowledgements

We'd like to thank our colleagues at Google. Specifically, the Google-Wide Profiling team builds the infrastructure which was used for the original paper, and which has continued to uncover optimization opportunities for more than a decade. The Efficiency League team takes these and routinely turns them into resource savings, with surprising ... efficiency. In particular, we want to acknowledge Chris Kennelly (Google) who has has built on the findings from this paper and helped with significant ideas in reducing the datacenter tax, as well as provided valuable input to this manuscript.

References

[1] "Fleetbench," https://github.com/google/fleetbench, 2022.

[2] "Google workload traces," https://dynamorio.org/google_workload_traces.html, 2022.

[3] G. Ayers, N. P. Nagendra *et al.*, "Asmdb: understanding and mitigating front-end stalls in warehouse-scale computers," in *International Symposium on Computer Architecture (ISCA)*, 2019.

[4] M. Ferdman *et al.*, "Clearing the clouds: a study of emerging scale-out workloads on modern hardware," 2012.

[5] Y. Gan *et al.*, "An open-source benchmark suite for microservices and their hardware-software implications for cloud & edge systems," in *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2019.

[6] A. H. Hunter *et al.*, "Beyond malloc efficiency to fleet efficiency: a hugepage-aware memory allocator," 2021.

[7] Intel, "Chapter 9: Code prefetch instruction updates," in *Intel Architecture Instruction Set Extensions and Future Features Programming Reference*, December 2022.

[8] Intel, "Powering infrastructure to help shape the data center of the future," 2022.

[9] S. Kanev *et al.*, "Mallacc: Accelerating memory allocation," in *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2017.

[10] S. Karandikar *et al.*, "A hardware accelerator for protocol buffers," in *International Symposium on Microarchitecture (MICRO)*, 2021.

[11] S. Karandikar *et al.*, "Cdpu: Co-designing compression and decompression processing units for hyperscale systems," in *International Symposium on Computer Architecture (ISCA)*, 2023.

[12] H. Kasture and D. Sanchez, "Tailbench: a benchmark suite and evaluation methodology for latency-critical applications," in *International Symposium on Workload Characterization (IISWC)*, 2016.

[13] C. Kaynak *et al.*, "Confluence: unified instruction supply for scale-out servers," in *International Symposium on Microarchitecture (MICRO)*, 2015.

[14] C. Kennelly and A. Evlogimenos, "Beware microbenchmarks bearing gifts," https://abseil.io/fast/39, 2023.

[15] T. A. Khan *et al.*, "Whisper: Profile-guided branch misprediction elimination for data center applications," in *International Symposium on Microarchitecture (MICRO)*.

[16] T. A. Khan *et al.*, "I-spy: Context-driven conditional instruction prefetching with coalescing," in *International Symposium on Microarchitecture (MICRO)*, 2020.

[17] R. Kumar *et al.*, "Blasting through the front-end bottleneck with shotgun," in *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2018.

[18] P. Mattson *et al.*, "Mlperf: An industry standard benchmark suite for machine learning performance," *IEEE Micro*, 2020.

[19] R. Potvin *et al.*, "Why Google stores billions of lines of code in a single repository," *Communications of the ACM*, 2016.

[20] A. Sriraman and A. Dhanotia, "Accelerometer: Understanding acceleration opportunities for data center overheads at hyperscale," in *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2020.

[21] A. Yasin, "A top-down method for performance analysis and counters architecture," in *International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2014.