# DCPerf: An Open-Source, Battle-Tested Performance Benchmark Suite for Datacenter Workloads

Wei Su, Abhishek Dhanotia, Carlos Torres, Jayneel Gandhi
Neha Gholkar, Shobhit Kanaujia, Maxim Naumov, Kalyan Subramanian
Valentin Andrei, Yifan Yuan, Chunqiang Tang
Meta Platforms
Menlo Park, USA

## Abstract

We present DCPerf, the first open-source performance benchmark suite actively used to inform procurement decisions for millions of CPU in hyperscale datacenters. Although numerous benchmarks exist, our evaluation reveals that they inaccurately project server performance for datacenter workloads or fail to scale to resemble production workloads on modern many-core servers. DCPerf distinguishes itself in two aspects: (1) it faithfully models essential software architectures and features of datacenter applications, such as microservice architecture and highly optimized multi-process or multi-thread concurrency; and (2) it strives to align its performance characteristics with those of production workloads, at both the system level and microarchitecture level. Both are made possible by our direct access to the source code and hyperscale production deployments of datacenter workloads. Additionally, we share real-world examples of using DCPerf in critical decision-making, such as selecting future CPU SKUs and guiding CPU vendors in optimizing their designs. Our evaluation demonstrates that DCPerf accurately projects the performance of representative production workloads within a 3.3% error margin across four generations of production servers introduced over a span of six years, with core counts varying widely from 36 to 176.

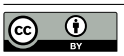
## CCS Concepts

• **Computer systems organization → Architectures**; **Cloud computing**; • **Software and its engineering → Software architectures**; • **General and reference → Performance**.

## Keywords

Benchmarking, Cloud Computing, Performance

**ACM Reference Format:**
Wei Su, Abhishek Dhanotia, Carlos Torres, Jayneel Gandhi, Neha Gholkar, Shobhit Kanaujia, Maxim Naumov, Kalyan Subramanian, Valentin Andrei, Yifan Yuan, Chunqiang Tang. 2025. DCPerf: An Open-Source, Battle-Tested Performance Benchmark Suite for Datacenter Workloads. In *Proceedings of the 52nd Annual International Symposium on Computer Architecture (ISCA '25), June 21–25, 2025, Tokyo, Japan.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3695053.3731411




*ISCA '25, Tokyo, Japan*

ACM ISBN 979-8-4007-1261-6/2025/06
https://doi.org/10.1145/3695053.3731411

## 1 Introduction

Datacenter CPUs represent a significant and rapidly expanding market, projected to grow from $12.8 billion in 2024 to $48.9 billion by 2033 [8]. At Meta, we operate a hyperscale fleet of millions of servers. Organizations of all sizes—from hyperscalers like Meta to smaller datacenter operators and public cloud customers—universally rely on performance benchmarks to select the most cost-effective CPUs. On the supply side, CPU vendors also depend on benchmarks to guide CPU designs and optimizations.

To address these needs, the industry and research community have developed a wide variety of benchmarks. However, developing benchmarks for datacenter workloads presents challenges that prior work has not sufficiently addressed. Below, we outline the challenges and our approach to addressing them in DCPerf [9], Meta's open-source benchmark suite for datacenter workloads.

The first challenge is that the high complexity of datacenter workloads makes them difficult to model accurately. Specifically, Meta operates thousands of services interconnected by an intricate RPC call graph [21], and the largest service (i.e., the frontend web application for Facebook) comprises millions of lines of code, receives code contributions from over ten thousand engineers, and undergoes thousands of code changes daily.

Our evaluation shows that existing benchmarks fail to accurately represent these complex datacenter workloads. For instance, performance projections based on SPEC CPU 2017 [4] overestimate the performance of Meta's latest server SKU by 28% compared to its actual performance on our production workloads. This disparity is unsurprising, as SPEC CPU was designed to represent single-process, computation-intensive workloads, rather than capture complex, distributed datacenter workloads. While some benchmarks like CloudSuite [11] were designed to model datacenter workloads, their unoptimized implementations for modern processors limit their representativeness. For example, evaluations on our production servers with 176 cores reveal that CloudSuite's benchmark for in-memory analytics fails to drive CPU utilization above 20% due to its limited scalability on modern servers with many cores.

To accurately model datacenter workloads, we identify the primary workload categories that account for the majority of our fleet capacity and create simplified benchmarks to model each category. Some workload categories are common across the industry, such as web applications, data caching, big-data processing, and video encoding, while others target social-network workloads, such as newsfeed ranking.

Compared to other benchmark developers, we have two unique advantages. First, as we have direct access to both the source code

and production deployments of datacenter workloads, we can ensure that the software architecture of DCPerf's benchmarks closely resembles the production workloads and periodically calibrate their performance to align with that of the production workloads. Second, we regularly use DCPerf to drive decisions in procuring millions of CPUs and guide CPU vendors in optimizing their designs. This regular exercise provides valuable feedback for improving DCPerf. In contrast, many past benchmarks have become outdated due to the absence of a strong mandate for continuous improvement.

The second challenge in developing benchmarks for datacenter workloads is that it is insufficient for a benchmark to simply achieve performance similar to the real-world application it models at the macro level in terms of throughput, latency, and CPU utilization. Instead, its performance characteristics at the microarchitecture level must also be sufficiently close. Otherwise, even if their performance is similar for the current generation of CPUs, as the next-generation CPU evolves with a different microarchitecture, performance may diverge significantly.

The key microarchitecture-level performance characteristics include: (1) instructions per cycle (IPC); (2) cache miss rates; (3) branch misprediction; (4) memory bandwidth usage; (5) effective CPU frequency; (6) overall power consumption and its breakdown across various CPU and server components; (7) instruction stall causes; (8) CPU cycles spent in the OS kernel and user space; and (9) CPU cycles spent in application logic and "datacenter tax" [23], such as libraries for RPC and compression.

To address these complexities, we devise a holistic approach to evaluate benchmark fidelity against production workloads across all aforementioned aspects. We then iteratively refine the benchmarks to reduce performance gaps relative to production workloads.

We make the following contributions in this paper.

**Novelty**: We develop a comprehensive approach to (1) faithfully modeling key software features of production workloads, such as highly optimized multi-process or multi-thread concurrency, and (2) measuring and improving benchmark fidelity against production workloads across a broad range of microarchitecture metrics. This novel combination is made possible by our direct access to the source code and production deployments of datacenter workloads. This approach enables the creation of benchmarks that accurately reflect real-world applications with millions of lines of code, within a 3.3% error margin. To our knowledge, this level of comprehensiveness and accuracy has not been reported previously.

**Impact**: Over the past three years, DCPerf has served as Meta's primary tool to inform CPU selection across x86 and ARM, influencing procurement decisions for millions of CPUs. Additionally, it has guided CPU vendors in optimizing their products effectively. For instance, in 2023, it enabled a CPU vendor to implement approximately 10 microarchitecture optimizations, resulting in an overall 38% performance improvement for our web application that runs on more than half a million servers. Finally, by making DCPerf open-source [9], we hope to inspire industry peers to also share their benchmarks, such as those for search and e-commerce.

**Experiences**: We share real-world examples of using DCPerf in critical decision-making, such as selecting future CPU SKUs and guiding CPU vendors in optimizing their designs. These kinds of experiences are rarely reported in research literature but offer valuable insights and motivation for future research.

**Broad Usage:** Although this paper focuses on applying DCPerf to CPU selection and optimization due to space constraints, DCPerf, as a general-purpose benchmark suite for datacenter applications, can help evaluate performance improvements or regressions in common software components it utilizes, including compilers, runtimes (e.g., PHP/HHVM and Python/Django), common libraries (e.g. Thrift and Folly), Memcache, Spark, or the OS kernel. For instance, Section 5.3 demonstrates how DCPerf helps identify scalability bottlenecks in the Linux kernel. Furthermore, DCPerf can help assess the effectiveness of a wide range of research ideas, such as resource allocation, resource isolation, performance modeling, performance optimization, fault diagnosis, and power management. These use cases are similar to existing full-application benchmarks like RUBiS [5, 33], TPC-W[27], and BigDataBench[43], but with the added advantage that DCPerf is well-calibrated with production datacenter workloads.

## 2 Requirements and Design Considerations

Before delving into the design of DCPerf and its benchmarks, we first outline the requirements and key design considerations.

### 2.1 Easily Deployable Outside Meta

While DCPerf is designed to accurately model Meta's production workloads, a key requirement is that CPU vendors can independently set up and run DCPerf without dependence on Meta's production environment. This independence enables CPU vendors to optimize the CPU's design, microcode, firmware, and configuration during the early stages of CPU development, even before Meta has access to any CPU samples. Although Meta has a performance testing platform [6] capable of running production code and identifying minor performance differences, it is unsuitable for external use due to its dependencies on Meta's production environment.

Moreover, although DCPerf is designed to model datacenter applications often running at large scale, to make it practical for developers outside Meta to use, it must operate on just one or a few servers without requiring a large-scale setup. In most cases, its benchmarks need only a single server to run. For benchmarks deployed as distributed systems, where the primary component running on one server depends on auxiliary components running on two or three other servers, only the primary component's performance is assessed. This component must be deployed on the server being evaluated, while auxiliary components can be deployed on any server. Overall, we have designed DCPerf to require only a few servers and streamlined the benchmarking process into three simple steps: clone the repository, build, and run the benchmarks.

### 2.2 High Fidelity with Production Workloads

In the past, some benchmarks were designed to mimic the functionality of real-world applications but did not faithfully replicate their software architectures or traffic patterns due to a lack of access to proprietary information. For instance, RUBiS [5, 33] and TPC-W [27] mimic an auction site and an e-commerce site, respectively, and are widely used in performance studies. However, we

consider such benchmarks insufficient, as they cannot accurately project the modeled application's performance on new CPUs.

In contrast, the DCPerf benchmark's software architectures and traffic patterns closely resemble those of the production workloads they model. For example, while many caching benchmarks implement a look-aside cache, DCPerf uses a read-through cache because our production systems employ it to simplify application logic. Moreover, we model the "datacenter tax" [23] associated with RPC, compression, and various libraries used in production. We also ensure that the benchmark's threading model matches that of the production system, e.g., using separate thread pools to handle fast and slow code paths depending on factors such as cache hits. On machines with many CPU cores, the benchmark spawns multiple instances to model the production system's multi-tenancy setup and ensure scalability. In contrast, insufficient scalability on many-core machines is a key limitation of CloudSuite [11]. Finally, the benchmark enforces the same service level objectives (SLOs) used in production, such as maximizing throughput while maintaining the 95th-percentile latency under 500ms for our newsfeed benchmark.

In addition to software architecture, DCPerf generates traffic patterns or uses datasets that represent production systems. For example, the distribution of request and response sizes is replicated from production systems. In the benchmark for big-data processing, the dataset is scaled down compared to the production dataset, but we ensure that each server processes an amount of data similar to that in production, and the dataset retains features such as table schema, data types, cardinality, and the number of distinct values.

Moreover, while prior benchmarks often focus on application-level performance, DCPerf strives to ensure characteristics aligned at the microarchitecture level (*e.g.*, IPC, cache misses, and instruction stall causes). To achieve this, DCPerf collects detailed statistics during each benchmarking run and analyzes them afterward. Furthermore, as the set of microarchitecture features evolves over time, DCPerf adopts an extensible framework that facilitates the easy addition of new features. This approach is preferred over hardcoding these features into DCPerf's core implementation, ensuring that the benchmark remains adaptable and scalable as new requirements emerge.

## 2.3 Balancing Performance and Power

Prior benchmarks often focus on performance, typically measured as throughput under the constraint of meeting SLOs such as latency and error rates. However, power capacity has always been a key limiting factor in datacenters. As organizations race toward artificial general intelligence (AGI), the shortage of datacenter power has become more urgent than ever. Consequently, DCPerf must help evaluate the trade-offs among power, performance, and total cost of ownership (TCO), rather than focusing solely on performance.

CPU thermal design power (TDP) represents the worst-case power consumption under maximum load. However, reserving datacenter power based on TDP is inefficient. In practice, most application's instruction mix cannot push the CPU to its TDP, even at 100% utilization. Furthermore, they are typically restricted from running near full CPU utilization to avoid violating SLOs. Therefore, for server deployment and capacity modeling, we use budgeted power, which reflects power consumption under high but practical loads. This situation typically arises when some servers must handle a load spike due to another datacenter region failing entirely.

TCO consists of two components: capital expenditures (Capex) and operating expenses (Opex). Capex covers the purchase of physical hardware. Opex represents the ongoing costs required to keep servers operational, such as expenses for power and maintenance.

DCPerf is designed to capture both performance per unit of power consumption (Perf/Watt) and performance per TCO (Perf/$). While higher values of both metrics are preferred, they are not always aligned. For instance, CPU *X* may offer higher Perf/Watt but lower Perf/$, whereas CPU *Y* may have lower Perf/Watt but higher Perf/$. The decision depends on business priorities. For example, even though CPU *X* incurs a higher TCO, it might be preferred if its power efficiency enables the installation of more AI servers in a power-constrained datacenter, potentially delivering substantial business value. To help evaluate the trade-off between Perf/Watt and Perf/$, DCPerf records detailed statistics on CPU clock frequency and power consumption during benchmarking.

# 3 DCPerf Framework and Benchmarks

In this section, we present the DCPerf framework and the current set of benchmarks included in DCPerf.

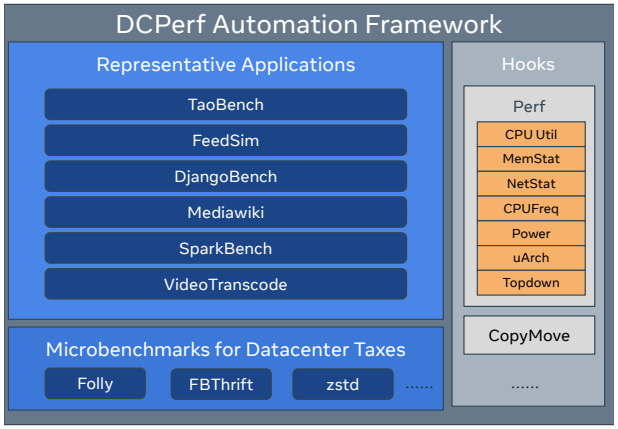



**Figure 1: DCPerf software architecture.**

## 3.1 DCPerf Overview

Figure 1 illustrates the overall architecture of DCPerf. We explain its components below.

**Automation framework.** To simplify the usage of DCPerf, the automation framework provides high-level commands such as `install` and `run` to install and run benchmarks without requiring the user to deal with the complexity of building the benchmarks or managing software dependencies.

**Result reporting.** After a benchmark run finishes, DCPerf reports the benchmark parameters and results, along with key information about the system being tested (e.g., CPU model, memory size, and kernel version). In addition to application performance metrics, such as throughput, DCPerf also reports a per-benchmark normalized score representing the performance of the machine being tested relative to a known baseline machine. After running all

| Workload | Web | Ranking | Data Caching | Big Data | Media Processing |
|---|---|---|---|---|---|
| **Benchmarks in DCPerf** | MediaWiki DjangoBench | FeedSim | TaoBench | SparkBench | VideoTranscode |
| **Performance metrics** | Peak RPS | RPS under latency SLO | Peak RPS and cache hit rate | Throughput | Throughput |
| **Req. proc. time** | Seconds | Seconds | Milliseconds | Minutes | Minutes |
| **Peak CPU util.** | 90-100% | 50-70% | 80% | 60-80% | 95-100% |
| **Thread-to-core ratio** | N(100) | N(10) | N(10) | N(1) | N(1) |
| **Per-server RPS** | N(1K) | N(100) | N(1M) | N(10) | N(10) |
| **RPC fanout** | N(100) | N(10) | N(10) | N(10) | 0 |
| **Instructions per request** | N(1B) | N(10B) | N(1K) | N(10B) | N(1M) |

**Table 1: Workloads modeled in DCPerf. $N(n)$ means a quantity of the same order of magnitude as $n$. Acronyms: Requests Per Second (RPS) and Remote Procedure Call (RPC).**

benchmarks, DCPerf reports the overall score, which is the geometric mean of all benchmark's scores. Individual benchmark results are stored in JSON format, allowing automation scripts to process them further.

**Extensibility.** DCPerf is designed as an extensible framework through plugins called *hooks*. New hooks for monitoring additional performance metrics can be easily added. Currently, the hooks support the following metrics:

- CPU utilization: both total CPU utilization and breakdowns, such as the percentage of cycles spent in user space, kernel and IRQs.
- Memory utilization: memory and swap space usage.
- Network traffic in bytes per second and packets per second.
- CPU core frequency as reported in `sysfs`.
- Power consumption reported by sensors on the motherboard.
- Top-down microarchitecture metrics: front-end stalls, backend stalls, incorrect speculations, and retiring instructions.
- Detailed microarchitecture metrics: IPC, memory bandwidth usage, cache misses, etc.

In addition to hooks for collecting performance data, DCPerf also supports the addition of hooks to facilitate benchmark execution, reporting, and automation. For example, the *CopyMove* hook can copy or move files, such as logs containing time-series performance data, into a separate folder for each benchmark run, ensuring long-term data preservation and enabling post-analysis.

**Client-server architecture.** Each benchmark is designed as a client-server application. The server component mimics a production workload, while the client component, potentially running on a different machine, sends requests to stress-test the server component via the Thrift [38] RPC protocol. This emulates not only the communication pattern in production, but also the RPC "datacenter tax" [15, 23], which consumes a significant amount of CPU cycles and memory.

## 3.2 DCPerf Benchmarks

We classify Meta's server fleet into three categories: general-purpose servers, storage servers with extensive SSD or HDD capacity designed for databases or distributed file systems, and AI servers equipped with GPUs or accelerators. DCPerf primarily focuses on workloads running on general-purpose servers.

**Workload selection.** Although Meta operates thousands of services on general-purpose servers, they can be classified into a smaller number of workload categories, and we design benchmarks to model each category separately. The workload categories currently modeled in DCPerf are shown in the first row of Table 1. We chose these workloads because they are the top consumers of power in Meta's fleet. Note that the "Web" category consists of two benchmarks, MediaWiki and DjangoBench, which model the frontend web application of Facebook and Instagram, respectively.

**Mediawiki.** The Mediawiki benchmark represents a classic web application. It runs Nginx [31] together with HHVM [28] as the web server, with MediaWiki [1] as the website to serve. It uses MySQL [45] as the backend database and Memcached [12] as the cache to accelerate processing and reduce database load. Siege [13] is used as the load generator to access several endpoints of the MediaWiki website, such as the Barack Obama page from Wikipedia, the edit page, the user login page, and the talk page. The MediaWiki benchmark runs all components on a single machine, pushes CPU utilization above 90%, and measures both the number of requests the HHVM server can handle per second and the latency distribution of the queries.

**DjangoBench.** DjangoBench is a web application designed to mimic Instagram. DjangoBench uses Python, Django, and UWSGI as the backend serving stack. Unlike MediaWiki's multi-threading model, UWSGI uses a multi-process model, spawning a number of worker processes equal to the number of logical CPU cores, enabling it to scale up on machines with many cores. In addition, DjangoBench uses Apache Cassandra as the backend database and Memcached as the cache. During benchmarking, the load generator visits several endpoints, such as `feed`, `timeline`, `seen`, and `inbox`, which mimic the main functionality of Instagram. We measure DjangoBench's throughput and latency distribution during benchmarking.

**FeedSim.** FeedSim models newsfeed ranking and operates on a single machine. It simulates key application logic, including feature extraction, ranking, backend I/O, and response composition. FeedSim is implemented using the open-source framework OLDISim [17], along with a set of libraries representing the datacenter tax, such as Thrift [38], Fizz [18], Snappy [16], and Wangle [35]. The client generates load to determine the maximum request rate FeedSim can handle while maintaining the 95th percentile latency within the SLO of 500ms.

**SparkBench.** SparkBench models query execution in a data warehouse. It uses a synthetic, representative dataset (over 100GB), from which SparkBench builds the Spark table at the beginning of benchmarking. The dataset, the Spark table, as well as the shuffle and temporary data, are stored on a RAID array created from remote NVMe SSDs on storage servers. These remote SSDs are connected to the SparkBench server via NVMe-over-TCP, with sufficient I/O bandwidth representative of real-world data center settings. SparkBench runs Spark [48] or Presto [37] as the backend query processing engine. SparkBench executes a Spark SQL query that scans the full dataset, performs a series of database operations, such as joining and comparison, and then writes query results to a new table. The entire benchmark execution is split into three stages: the first and second stages mainly load data from the tables and

are I/O-intensive, whereas the third stage is computation-intensive. Thus, the total query execution time reflects the end-to-end data warehouse performance, while the execution time of the last stage can be used to evaluate CPU performance.

**TaoBench.** TaoBench is a read-through, in-memory cache modeled after TAO [3]. TaoBench consists of a Memcached-based server and a Memtier-based client. Both the server and the client are modified to mimic TAO request generation and processing at a very high rate. The server spawns a number of so-called fast and slow threads. When a request encounters a cache hit in Memcached, a fast thread simply returns the cached object to the client. However, in the case of a cache miss, the request is dispatched to a slow thread, which simulates backend database lookup delay, new object creation, and Memcached insertion using the `SET` command. The object sizes, hit rate distribution, and network throughput are all modeled after the actual TAO production workload. The benchmark measures cache hit rate and request throughput.

**VideoTranscodeBench.** VideoTranscodeBench uses open-source encoders such as `ffmpeg`, `x264`, and `svt-av1`. We generate realistic configurations for a typical video processing pipeline (resizing and encoding) and leverage the open-source Netflix El Fuente video [25, 30] as a representative dataset. At the beginning of benchmarking, each CPU core is utilized by one `ffmpeg` instance to (1) resize a video clip into multiple resolutions and (2) encode the resized video clip with the specified video encoder. This benchmark is embarrassingly parallel and can push CPU utilization to more than 95%.

**Datacenter Tax Microbenchmarks.** Some common library functions used by datacenter applications, such as those for RPC, encryption, hashing, serialization, concurrency management, and memory operations, are unrelated to the applications' business logic but are necessary for optimal performance at scale in datacenters. We call these the "datacenter tax," which has been reported to comprise nearly 30% of CPU cycles across Google's fleet [23] and 18-82% of CPU cycles depending on the application across Meta's fleet [39]. Because of their importance, we model these functions as a set of microbenchmarks. When performance bottlenecks are identified in these functions during full-workload benchmarking, we use these microbenchmarks to pinpoint the problem and guide targeted optimizations. Additionally, these microbenchmarks are valuable indicators—if a server SKU performs poorly on them, it is likely to exhibit subpar performance for many applications.

## 3.3 DCPerf Implementation

DCPerf implementation is two-fold. For the benchmarks themselves, we selected open-source software stacks and libraries that are being used by Meta or closely mimic Meta's production workloads, and patch them to ensure the characteristics alignment across different levels. Table 2 demonstrates the major software stacks and programming languages of each benchmark. For the DCPerf automation framework, it is implemented mostly with Python and Shell scripts.

We have a small team directly working on DCPerf design, development, and maintenance over the past few years. We continuously improve DCPerf and make major external releases when there are significant changes. We welcome external contributions, and in fact, some CPU vendors have already contributed.

| Benchmark | Software Stacks | Programming Languages |
|---|---|---|
| MediaWiki | HHVM, MediaWiki<br>Memcached, MySQL, Nginx, wrk | PHP, Hack |
| DjangoBench | Django, UWSGI<br>Apache Cassandra, Memcached | Python, C++ |
| FeedSim | OLDIsim<br>Compression (Zlib, Snappy)<br>Crypto (OpenSSL, libsodium, fizz)<br>Protocol (FBThrift, RSocket, Wangle) | C++ |
| TaoBench | Memcached, Memtier<br>Folly, fmt, libevent | C++ |
| SparkBench | Apache Spark, OpenJDK | Java, SparkSQL |
| VideoTranscode | ffmpeg, svt-av1, libaom, x264 | C++ |

**Table 2: Major software stacks and programming languages of DCPerf benchmarks.**

# 4 Evaluation

We evaluate whether DCPerf adequately represents the performance characteristics of datacenter applications. Our evaluation aims to answer the following questions:

- How accurately can DCPerf project the performance of different generations of servers for datacenter workloads? (Section 4.1)
- Do DCPerf benchmarks and real-world applications exhibit similar profiles in the micro-level metrics: instruction stalls (Section 4.2), microarchitecture metrics such as IPC and cache misses (Section 4.3), power consumption (Section 4.4), and CPU cycles spent in application logic and "datacenter tax" (Section 4.5)?
- How does DCPerf compare with other datacenter-oriented benchmarks, such as CloudSuite [11]? (Section 4.6)

## 4.1 Projection Accuracy Across Different CPUs

A primary goal of DCPerf is to help select the most cost-effective and power efficient next-generation x86 or ARM CPUs from new offerings by various vendors. To benefit from the latest technology, Meta typically evaluates cutting-edge CPUs that are still under development. This means that CPU samples are either unavailable to Meta or insufficient to run Meta's hyperscale production workloads for performance measurement. Instead, CPU vendors run DCPerf benchmark traces on their simulators or end-to-end benchmarks on their early silicon samples to guide their design and optimization, and also report the benchmark results to Meta. Meta uses these results to project the performance of production workloads on next-generation CPUs, guiding early decision in CPU selection. As the next-generation CPUs mature and the vendors provide more samples to Meta, Meta measures the actual performance of production workloads on those CPUs, improving the accuracy of performance data over time.

CPU selection relies on DCPerf to accurately project the performance of production workloads across different CPUs. To evaluate the accuracy of DCPerf's projections, we compare the DCPerf-projected performance with the actual performance measured in production at hyperscale, using four different server SKUs deployed at Meta, as shown in Table 3. We also evaluate the projection accuracy of SPEC CPU 2006 and 2017 (hereafter referred to as SPEC

| | SKU1 | SKU2 | SKU3 | SKU4 |
|---|---|---|---|---|
| Logical cores | 36 | 52 | 72 | 176 |
| RAM (GB) | 64 | 64 | 64 | 256 |
| Network bandwidth (Gbps) | 12.5 | 25 | 25 | 50 |
| Storage | 256GB SATA | 512GB NVMe | 512GB NVMe | 1TB NVMe |
| Year of introduction | 2018 | 2021 | 2022 | 2023 |

**Table 3: Specification of x86-based production server SKUs.**

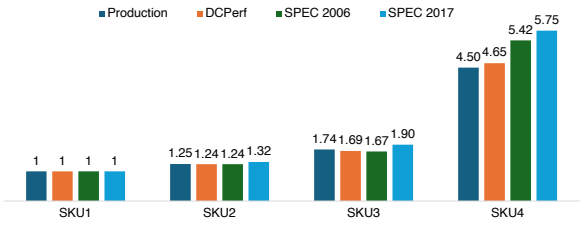



**Figure 2: Performance of different SKUs normalized to SKU1.**

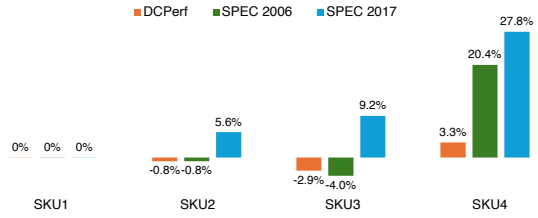



**Figure 3: Relative error of performance projection.**

2006 and 2017 for brevity). We select a subset of the entire SPEC CPU benchmark suite, as we found it better representing Meta's workloads, before DCPerf was introduced. The evaluation of CloudSuite [11] is presented in Section 4.6 because it is not sufficiently scalable to produce comparable results.

Figure 2 shows the comparison result. The performance score of a benchmark suite is the geometric mean of the scores of its individual benchmarks[1], normalized to those on SKU1. The performance score of the production workloads is calculated as the geometric mean of the performance scores of DCperf's counterparts in the production (aggregated from thousands of servers), also normalized to those on SKU1 and weighted by each workload's power consumption in our fleet. To highlight the differences more clearly, we transform the data in Figure 2 into Figure 3, showing the projection errors of the benchmark suites relative to the production workload's actual measurements. The projection errors are 0% for SKU1 because it is used as the baseline for calibration.

DCPerf more accurately reflects the performance of production workloads on different SKUs. For example, for SKU4, DCPerf's projection is only 3.3% higher than the actual performance, while the projections from SPEC 2006 and 2017 are 20.4% and 27.8% higher, respectively. Interestingly, although SPEC 2017 is considered an improved version of SPEC 2006, its projection for datacenter workloads is actually less accurate than the older SPEC 2006. Regardless, neither SPEC 2006 nor SPEC 2017 is sufficiently accurate for projecting performance for datacenter workloads, especially on servers with many cores.

[1]The score of an individual benchmark is defined as its application metric (such as RPS request per second) normalized to that on SKU1.

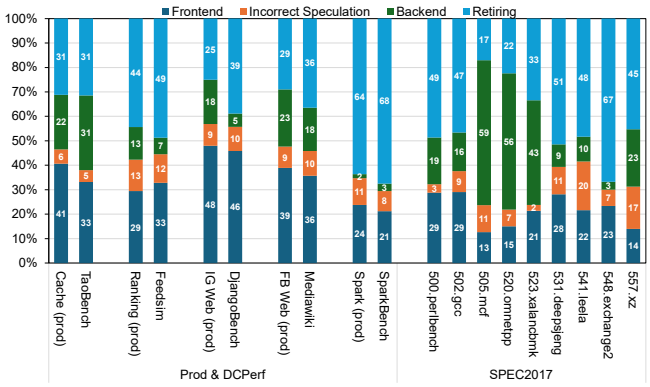



**Figure 4: Comparing TMAM profiles across Prod, DCPerf, and SPEC 2017. "*Cache (prod)*" is the production workload that TaoBench models, "*Ranking (prod)*" is the production workload that FeedSim models, and so forth.**

DCPerf performs robustly across servers with radically different configurations. From SKU1 to SKU4 over six years, overall server performance increases by 4.5 times, the core count rises from 36 to 176, RAM expands from 64GB to 256GB, network bandwidth grows from 12.5Gbps to 50Gbps, and storage transitions from SATA to NVMe. Despite these significant changes, DCPerf's projection errors remain under 3.3%. This demonstrates that our methodology for building benchmarks is resilient to technological evolution.

## 4.2 Top-down Microarchitecture Metrics

In addition to accurately modeling the end-to-end performance of real-world applications, we strive to ensure that DCPerf benchmarks reflect the microarchitecture performance characteristics of these applications. However, perfect alignment in micro-level metrics is unrealistic for several reasons. First, the benchmark's codebase is significantly smaller, by orders of magnitude, compared to its counterpart applications. Second, application code evolves rapidly; for instance, Facebook's web application undergoes thousands of code changes daily. Third, alignment spans more than a dozen microarchitecture metrics, making it inherently multi-dimensional. Adjusting benchmarks to align on certain metrics may lead to divergences in others.

Therefore, our goal is not to achieve perfect alignment but rather to use significant misalignment as an indicator for identifying areas of improvement in the benchmarks, as their refinement is a never-ending process. In this section, we assess alignment using the microarchitecture metrics defined by the Top-down Microarchitecture Analysis Method (TMAM) [47]. In the next section, we employ more fine-grained microarchitecture metrics for this purpose.

TMAM can expose architectural bottlenecks despite the latency-masking optimizations in modern out-of-order CPUs. It identifies bottlenecks in terms of the percentage of "instruction slots", defined as the fraction of hardware resources available to process micro-operations that are wasted due to stalls in each cycle. The stalls are categorized as: (1) frontend stalls due to instruction fetch misses; (2) backend stalls due to pipeline dependencies and data load misses;

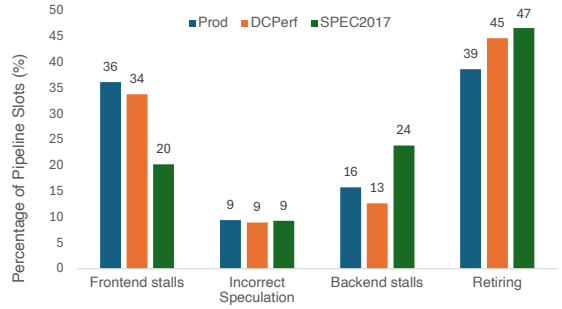



**Figure 5: Average values for the different causes of stalls.**

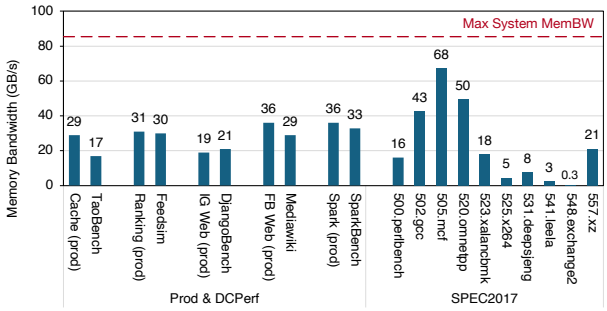



**Figure 7: Memory bandwidth consumption.**

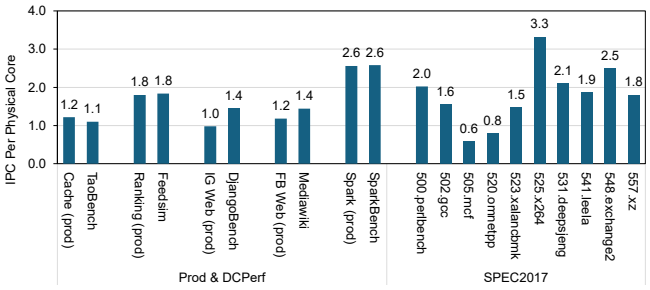



**Figure 6: IPC per physical core (with SMT On).**

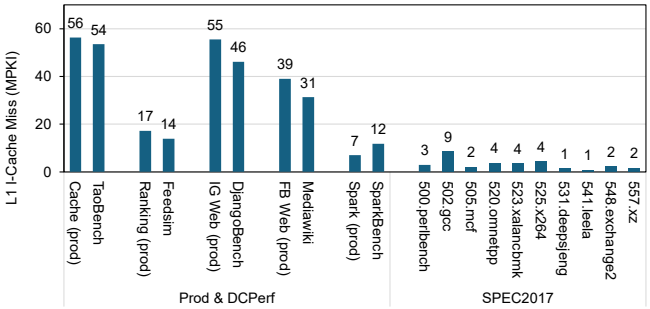



**Figure 8: L1 I-Cache misses (MPKI).**

(3) incorrect speculative execution due to recovery from branch mispredictions; and (4) retiring of useful work.

Figure 4 compares the TMAM profiles across production workloads, DCPerf, and SPEC CPU 2017. These experiments were run on SKU2 described in Table 3, as it is the most widely used SKU in Meta's fleet as of 2024. Overall, DCPerf's TMAM profiles are reasonably close to those of the production workloads, although DjangoBench has a relatively larger disparity with respect to "IG Web (prod)" in stalls due to the backend and retiring. In contrast, the SPEC benchmarks show significantly different TMAM profiles compared to the production workloads, with backend stalls varying much more drastically across the benchmarks. To make the overall trend clearer, we compare the average values in Figure 5. It clearly shows that SPEC has far fewer frontend stalls. This is because the SPEC benchmarks have a small codebase and, hence, fewer frontend stalls due to instruction cache misses.

## 4.3 Fine-Grained Microarchitecture Metrics

In addition to TMAM, we use detailed microarchitecture metrics to evaluate DCPerf.

**IPC (Figure 6).** The IPC of production workloads and DCPerf benchmarks varies similarly within the range of 1.0 to 2.6. In contrast, the IPC of SPEC 2017 benchmarks varies over a much wider range of 0.6 to 3.3. Overall, the IPC of DCPerf benchmarks closely matches that of production workloads, except for a wider gap between IG Web (IPC 1.0) and DjangoBench (IPC 1.4). This difference is consistent with the discrepancy in their TMAM profiles, where DjangoBench has significantly fewer backend stalls compared to IG Web. Further improving DjangoBench is an area of ongoing work.

**Memory bandwidth (Figure 7).** The memory bandwidth consumption of production workloads varies between 19 and 36 GB/s, while DCPerf ranges from 17 to 33 GB/s, both consuming around ~30% of the system's memory bandwidth. In contrast, SPEC benchmarks exhibit a much wider range, from 0.3 to 68 GB/s (as high as ~70% of the system's memory bandwidth). Compared to production workloads, SPEC benchmarks tend to consume either significantly more or significantly less memory bandwidth, indicating that they are not representative of datacenter workloads. In contrast, DCPerf benchmarks generally align closely with the memory bandwidth consumption of production workloads, with one notable exception: TaoBench (17 GB/s) compared to the cache production workload (29 GB/s). This suggests that TaoBench's data working set is smaller, resulting in a higher hit rate in caches. Improving TaoBench's memory profile is an area of ongoing work.

**L1 I-Cache miss (Figure 8).** The production workloads and DCPerf benchmarks exhibit comparable L1 I-Cache misses per kilo instructions (MPKIs), ranging from 7 to 56. Some workloads, such as IG Web and FB Web and their representative benchmarks, have more I-Cache misses due to their large codebase; whereas other workloads like Cache in production and TaoBench, have high I-Cache misses yet with much smaller code footprint because they exhibit frequent context switches resulted from their high thread-to-core oversubscription ratio. In contrast, the SPEC benchmarks have significantly lower miss rates, ranging from 1 to 9. This indicates that the instruction working set of SPEC benchmarks is too small relative to real-world datacenter applications, making them unsuitable for evaluating the impact of I-Cache on performance.

**CPU Utilization (Figure 9).** The SPEC benchmarks primarily focus on user-space performance, spending minimal time in the

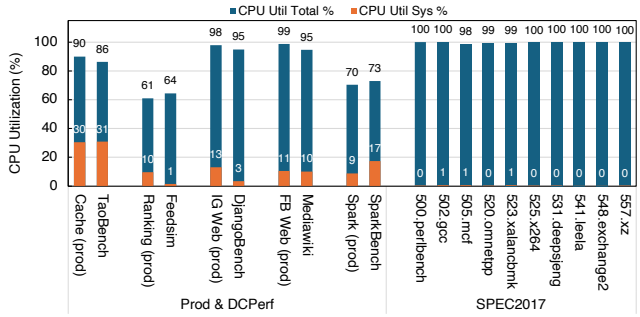



**Figure 9: CPU utilization.**

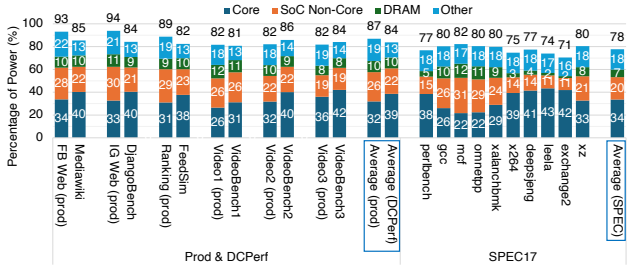



**Figure 10: Power Consumption.**

kernel. In contrast, both production workloads and DCPerf benchmarks spend a significant portion of their time in the kernel. Notably, the cache production workload and TaoBench spend around 30% of their time in the kernel, as the caching logic involves limited user-space processing and significant time in the kernel for network and storage I/O operations. Another difference is that SPEC benchmarks typically drive CPU utilization to 100%, while some production workloads and DCPerf benchmarks only reach 60-70%. This discrepancy stems from several factors: the CPU not being the bottleneck resource, internal synchronization bottlenecks in the code, or SLO violations (e.g., latency and error rates) occurring before CPU utilization reaches 100%.

## 4.4 Power Consumption

We rely on sensors on the server's motherboard to measure power consumption. Figure 10 shows the power consumption broken down into four categories: CPU core, CPU SoC non-core (e.g., interconnect and memory controller), DRAM, and others (e.g., storage, NIC, BMC and fans). Each component's power consumption is normalized to the server's total designed power. In this experiment, VideoBench is configured with three different video quality settings, which affect its power consumption.

Overall, the average power consumption across the production workloads, DCPerf, and SPEC is 87%, 84%, and 78%, respectively, indicating that DCPerf is more accurate than SPEC in modeling power consumption. The breakdown of DCPerf's individual benchmarks' power consumption is reasonably close to that of the production workload it models. However, overall, DCPerf tends to overrepresent the CPU core's power consumption and underrepresent the power consumption of non-core and "other" components. This is an area for improvement for DCPerf.

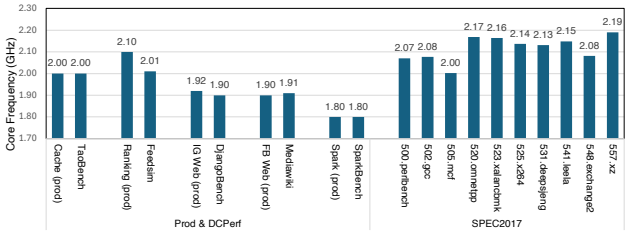



**Figure 11: Core frequency in GHz.**

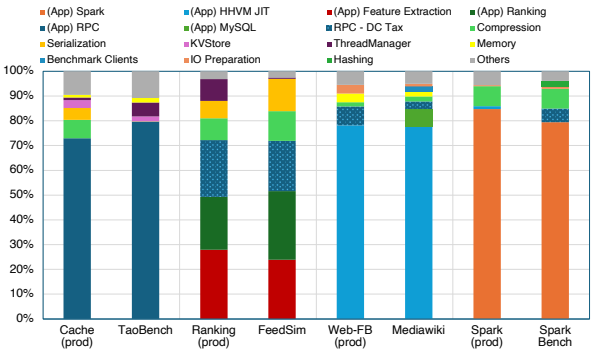


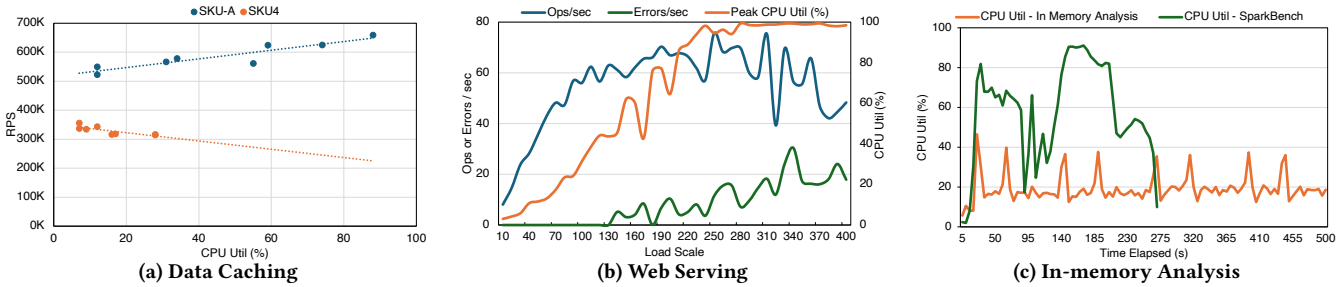

**Figure 12: Breakdown of CPU cycles spent in hot functions. Functions starting with "(App)" are application logics while the rest are the datacenter tax.**

As clock frequency impacts power consumption, we show the CPU core frequency when running different workloads in Figure 11. The production workloads and DCPerf benchmarks exhibit similar core frequencies, averaging 1.94 GHz and 1.92 GHz, respectively. In contrast, the SPEC benchmarks typically run at higher frequencies overall, averaging 2.12 GHz. Despite the higher CPU core frequency, the overall power consumption of the SPEC benchmarks still trends lower than that of the production workloads and DCPerf, because they do not sufficiently exercise the diverse components in CPUs and, more broadly, in servers as a whole.

## 4.5 Datacenter Tax versus Application Logic

Previous work [39] [23] [22] have shown that datacenter applications spend a significant number of cycles on library code not directly related to application logic, such as RPC and compression. These functions are often referred to as the "datacenter tax." When developing the DCPerf benchmarks, we profile production workloads and make efforts to tune the code composition in the benchmarks so that the ratios of CPU cycles spent on application logic and various categories of datacenter tax are close to those in production workloads.

Figure 12 shows the breakdown of CPU cycles spent in different hot functions, some belonging to application logic and others to the datacenter tax. Although the impact of different categories of the tax varies widely across workloads, DCPerf benchmarks, overall, reasonably reflect the datacenter tax. One exception is that TaoBench spends significantly less time on compression and serialization compared to the production workload it models. Addressing




(a) Data Caching (b) Web Serving (c) In-memory Analysis

**Figure 13: CloudSuite's benchmarking results.**

this is an area for future work. Despite this limitation, DCPerf represents a significant advancement in accounting for and evaluating the datacenter tax, an aspect overlooked by previous benchmarks.

## 4.6 Evaluating Alternatives: CloudSuite

To identify reasonable comparison baselines for evaluating DCPerf, we explored several relevant benchmark suites [10, 11, 14, 24, 34, 41, 44, 46] that share similar goals of benchmarking datacenter applications. However, we found that they all fell short of representing Meta's datacenter workloads. In this section, we present performance results for CloudSuite [11] as an example.

CloudSuite's Data Caching benchmark runs Memcached with the Twitter dataset [29]. Its function is similar to DCPerf's TaoBench, though Data Caching does not implement the behavior of a read-through cache. We ran Data Caching on both SKU4 and SKU-A servers (see Tables 3 and 4 for server configurations). To drive up CPU utilization and maximize throughput, we experimented with different configurations for Data Caching, varying the number of server instances, server threads, and client threads. We report the results for the best configuration in Figure 13a, which shows the requests per second (RPS) achieved on the two server SKUs at various levels of CPU utilization.

On SKU-A with 72 cores, when CPU utilization increases from 12% to 88%, a 7.3-fold increase, the throughput increases by only 26%, showing that Data Caching has limited scalability. On SKU4 with 176 cores, the throughput actually decreases as both the thread pool size and CPU utilization increase, indicating a performance anomaly. In addition, the portion of "datacenter tax" in the benchmark is not modeled accurately to reflect the CPU cycles consumption in Meta's production. Moreover, we encountered segmentation faults in the client when trying to use more than five Memcached server instances. Overall, this experiment shows that Data Caching is not optimized to scale effectively on modern servers with very high core counts.

CloudSuite's Web Serving benchmark runs the open-source social networking engine Elgg [7] using PHP and Nginx, with Memcached providing caching and MariaDB serving as the database. This benchmark is similar to DCPerf's MediaWiki benchmark. We run Web Serving on an SKU4 server while varying the benchmark's load-scale factor from 10 to 400. Figure 13b shows the throughput (Ops/sec), peak CPU utilization, and error rates. Web Serving's throughput slows down after the load scale exceeds 100, even though CPU utilization continues increasing linearly until it reaches 100%. The rate of errors, most notably "504 Gateway Timeout," increases after the load scale exceeds 140, even while CPU utilization is still below 50%. Once again, this experiment shows that Web Serving is not optimized to scale effectively on servers with many cores.

| | SKU-A | SKU-B |
|---|---|---|
| Logical cores | 72 | 160 |
| L1-I cache size (normalized) | 4× | 1× |
| RAM (GB) | 256 | 256 |
| Network bandwidth (Gbps) | 50 | 50 |
| Server Power (Watt) | 175 | 275 |

**Table 4: Specification of the ARM-based new server SKUs.**

CloudSuite's In-memory Analytics benchmark uses Apache Spark to run an alternating least squares (ALS) filtering algorithm [42] on a user-movie rating dataset called MovieLens [19]. This benchmark shares similarities with both SparkBench (in terms of the software stack) and FeedSim (in terms of its function, focusing on ranking rather than big-data queries). The MovieLens dataset's uncompressed size is around 1.2GB. We run this benchmark on an SKU4 server and compare its execution time and CPU utilization with SparkBench in Figure 13c. This benchmark only achieves about 20% CPU utilization throughout the run. We explored different configurations, such as Spark parameters for parallelism, number of executor workers, and executor cores, but failed to push the CPU utilization of this benchmark higher. Once again, this experiment shows that the benchmark is not optimized to scale effectively on servers with many cores.

# 5 Using DCPerf: Case Studies

DCPerf is Meta's primary tool for evaluating prospective CPUs, determining server configurations, guiding vendors in optimizing CPU microarchitectures, and identifying systems software inefficiencies. We describe several case studies below.

## 5.1 Choosing ARM-Based New Server SKUs

While the server SKUs in Table 3 are all based on x86, in 2023, we began designing a new server SKU based on ARM. The two server SKU candidates, SKU-A and SKU-B, shown in Table 4, use ARM CPUs from different vendors. In the early stages of server design, we had only a few testing servers for SKU-A and SKU-B, which were insufficient to set up and run large-scale, real-world production workloads for testing purposes. Therefore, we use DCPerf to evaluate these testing servers and compare them with the existing x86-based SKU1 and SKU4 servers shown in Table 3.

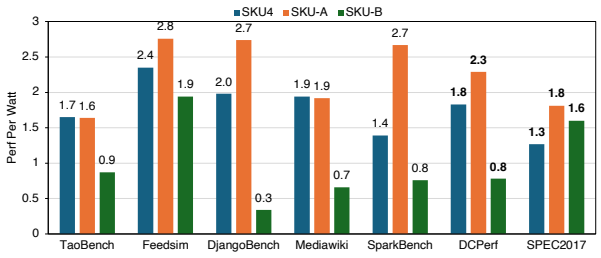



**Figure 14: Comparing Perf/Watt across server SKUs.**

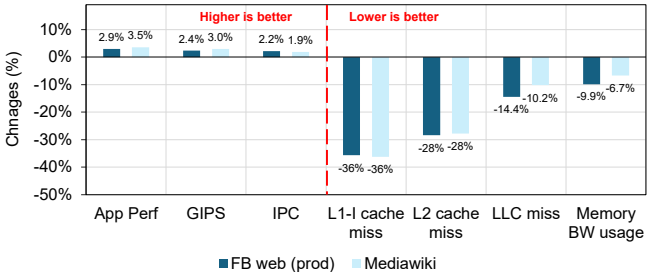



**Figure 15: Impact of the vendor's improvement in the CPU's cache replacement algorithm. GIPS on the x-axis means Giga Instructions Per Second.**

As described in Section 2.3, due to the ongoing shortage of power in datacenters, a key metric we consider is the performance that a server delivers per unit of power consumption (Perf/Watt), as opposed to considering absolute performance alone. We calculate the Perf/Watt metric as follows. While running DCPerf on a server, we collect performance numbers and monitor the server's power consumption. Dividing a benchmark's performance number by the average server power consumption during the benchmark's steady-state run yields the Perf/Watt metric. However, as the benchmarks report performance results in different scales and even different units, we need to normalize the results. We divide each benchmark's raw Perf/Watt result on the new server by its Perf/Watt result on the x86-based SKU1 server, which serves as the baseline for comparison. Finally, we calculate the geometric mean of the Perf/Watt metrics across all DCPerf benchmarks to produce a single Perf/Watt number for the DCPerf suite.

Figure 14 compares Perf/Watt across different server SKUs. In terms of Perf/Watt, ARM-based SKU-A outperforms x86-based SKU4, our latest server SKU running in production, by 25% overall, with the largest gain of 92% for SparkBench. In contrast, SKU-B underperforms SKU4 by 57% overall, with the largest loss of 85% for DjangoBench and 63% for Mediawiki, respectively. These results demonstrate that DCPerf is effective in identifying SKU-B's weaknesses in running datacenter applications, especially its inefficiency in handling user-facing web workloads, due to its smaller L1 I-Cache, which is not well-suited for the large code base of web workloads. With DCPerf's help, we decided to choose SKU-A over SKU-B as the next-generation ARM-based server SKU in our fleet.

This evaluation also shows that comparing Perf/Watt may lead to a different conclusion than comparing absolute performance. This is especially important when comparing ARM-based servers, which may be more power-efficient, with x86-based servers, which may offer better absolute performance. Specifically, ARM-based SKU-A has lower absolute performance but better Perf/Watt compared to x86-based SKU4. However, it is important to note that this is not a general conclusion about ARM and x86; SKU selection critically depends on the specific implementation of individual CPU products. For example, ARM-based SKU-B is inferior to x86-based SKU4 in terms of both Perf/Watt and absolute performance.

In this experiment, we also evaluate SPEC 2017's effectiveness in comparing the server SKUs. If we had relied on SPEC for decision-making, we would have incorrectly concluded that the ARM-based SKU-B is better than the x86-based SKU4. Moreover, because SKU-A and SKU-B are comparable in terms of Perf/Watt (1.8 versus 1.6) in the SPEC results, we would not have been able to decisively reject SKU-B and would have had to compare subtle differences in many other factors, such as price, vendor support, and CPU reliability, hoping that one of those factors would provide a significant enough difference to facilitate decision-making.

Finally, Figure 14 also demonstrates that different benchmarks, and their corresponding production workloads, scale performance differently across two server SKUs. Specifically, in terms of Perf/Watt, SKU-A outperforms SKU4 by 92% for SparkBench, while both perform nearly identically for MediaWiki. Although it is theoretically possible to design SKUs optimized for specific applications, this approach incurs high costs for introducing and maintaining additional SKUs in a hyperscale fleet and leads to resource waste due to SKU mismatches. For example, when one application's load grows slower than anticipated, the oversupply of its custom server SKU cannot be efficiently utilized by other applications. Therefore, SKU selection must consider benchmarks representing a wide range of workloads rather than focusing on individual ones. This is why, in Figure 2, we compare the aggregate performance across benchmarks in the suites rather than individual benchmarks.

## 5.2 Guiding Vendors to Optimize CPU Design

DCPerf not only benefits Meta in server SKU selection but also benefits CPU vendors by allowing them to independently run DCPerf in their in-house development environments to iteratively improve their new CPU products. In 2023, we collaborated with a CPU vendor to introduce their next-generation CPU to our fleet and optimize its performance in the process. One of the microarchitecture optimizations the vendor conducted was to iteratively improve the microcode for managing the CPU's cache replacement algorithm to enhance the cache hit rate.

As shown in Figure 15, in the vendor's development environment, this optimization improved the MediaWiki benchmark's performance by 3.5%, increased IPC by 1.9%, and reduced misses in the L1 I-cache by 36% and in the L2 cache by 28%. Later, we tested this optimization in Meta's production environment and confirmed a 2.9% performance improvement on Meta's Facebook web application, which runs on more than half a million servers and consists of millions of lines of code. Additionally, we confirmed improvements in the Facebook web application's microarchitecture metrics similar to those shown in Figure 15. In contrast, testing on SPEC 2017 revealed no noticeable performance changes. Without DCPerf, the vendor could not have made this optimization relying only on the standard SPEC benchmarks.

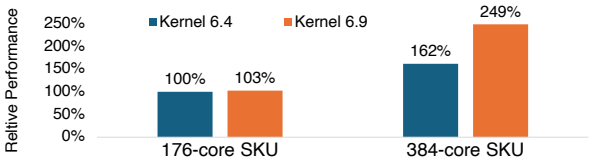



**Figure 16: TaoBench's relative performance with different Linux kernels and server SKUs.**

In addition to the optimization described above, the CPU vendor implemented approximately ten other microarchitecture optimizations under the guidance of DCPerf, such as tuning the uncore frequency and policies for managing TLB coherence. Altogether, these optimizations resulted in a 38% performance boost and a 47% Perf/Watt improvement for Meta's Facebook web application. Throughout the process, the vendor's ability to quickly and independently evaluate their optimizations in their development environment using DCPerf, which is easy to deploy yet effectively represent Meta's production workloads, was critical to their success.

## 5.3 Identifying Issues in the OS Kernel

In addition to optimizing CPU design and server selection, DCPerf also helps identify performance issues in system software such as the OS kernel. In 2024, to proactively prepare for scaling both our production workloads and DCPerf benchmarks to CPUs with significantly more cores, we tested DCPerf on server SKUs with 176 and 384 logical cores, respectively. During this process, we observed abnormal performance results for TaoBench. TaoBench's performance on the 384-core SKU was only 1.6 times its performance on the 176-core SKU, whereas we expected at least $\frac{384}{176} = 2.2$ times higher performance because the 384-core SKU also has other improvements beyond the core count increase.

Among many investigations we explored, one thing we suspected is that the kernel's limited scalability could be the cause of this performance anomaly. Therefore, we repeated the experiment on different kernel versions. Ultimately, we found that upgrading the Linux kernel from 6.4 to 6.9 resolved the anomaly. TaoBench's performance on the 384-core SKU increased to 2.5 times its performance on the 176-core SKU, aligning with our expectation. This experience indicated that kernel 6.4 had scalability issues when running on servers with many cores.

We further investigated this issue using DCPerf's extensible hooks, which enabled us to analyze hotspots in TaoBench's execution. On the 384-core SKU, we observed significant overhead in kernel 6.4's scheduling functions (e.g., `enqueue_task_fair`) and the `nanosleep()` system call invoked by TaoBench. Further analysis revealed that the root cause was lock contention on a counter used for tracking system load, exacerbated by the large number of CPU cores and threads. This issue was mitigated in kernel 6.9 by a patch that reduced the update frequency of the counter [26].

To illustrate the impact of the issue and its resolution, Figure 16 shows TaoBench's performance across various kernel versions and server SKUs. On the 176-core SKU, the performance difference between kernel 6.4 and 6.9 is only 3%. However, on the 384-core SKU, TaoBench achieves $\frac{249\%}{162\%} - 1 = 54\%$ higher performance with kernel 6.9 compared to kernel 6.4.

Our experience debugging this issue highlights several lessons. First, as CPU core counts continue to grow rapidly, both system software (e.g., OS kernels) and applications are likely to encounter scaling bottlenecks; therefore, organizations must proactively identify and address these issues. For example, limited scalability makes CloudSuite [11] unrepresentative of modern datacenter applications. Second, debugging performance issues in system software with DCPerf is far easier than with complex datacenter applications in production. Without DCPerf, testing production caching workloads on exploratory server SKUs available in very limited quantities would have been impractical.

# 6 Takeways: Insights and Lessons

In the previous sections, we shared our insights and lessons learned from developing and using DCPerf. To make these takeaways more accessible, we summarize them in this section.

**Limitations of popular benchmarks:** Users of SPEC CPU and other popular benchmarks should be aware of their limitations in representing datacenter workloads. Compared to real-world datacenter workloads, SPEC CPU overestimates runtime CPU frequency and significantly overstates the performance of many-core CPUs, while underestimating instruction cache misses, server power consumption, and CPU cycles spent in the OS kernel. Additionally, many full-system benchmarks, such as CloudSuite, fail to scale effectively on many-core CPUs, leading to a significant underestimation of those CPUs' actual performance.

**Benchmark generalization:** The majority of benchmarks in DCPerf represent common workloads across the industry, including web-serving, caching, data analytics, and media processing. Although these benchmarks are common, their specific configurations are derived from Meta's workload characteristics, such as the size distribution of cached objects. If other organizations wish to have DCPerf represent their own workload characteristics, it is possible with some effort to change benchmark configurations to match their workloads.

**Many-core CPU:** As the core counts of current and future CPUs grow rapidly, system software (e.g., the OS kernel), applications, and benchmarks will all face significant scaling challenges. This is evident in the Linux kernel performance anomaly (Section 5.3) and the limited scalability of CloudSuite (Section 4.6). Organizations must invest sufficiently in software scalability ahead of time.

**Perf/Watt:** Due to power shortages in datacenters, Perf/Watt is as important as TCO and absolute performance. CPU vendors must prioritize Perf/Watt as a primary metric for optimization.

**ARM versus x86:** Our evaluation shows that ARM CPUs are now a viable option for datacenter use, with some ARM CPUs offering better Perf/Watt compared to certain x86 CPUs. However, the choice between ARM and x86 depends on the specific CPU implementation, as some ARM CPUs we evaluated are inferior to certain x86 CPUs in both Perf/Watt and absolute performance.

**Post-silicon CPU optimization:** Even after a CPU's tapeout and manufacturing, significant opportunities remain to enhance the CPU performance through optimizations in microcode, firmware, and various configurations. For instance, as described in Section 5.2, DCPerf enabled a vendor to boost the performance of their next-generation CPU for the Facebook web application by 38%. This

underscores the importance of developing representative benchmarks to support such post-silicon optimizations.

**Modeling software architecture:** To achieve accurate performance projections, it is insufficient to merely model the functional behavior of datacenter applications; instead, benchmarks must capture key aspects of their software architecture, as these applications tend to be highly optimized. For example, TAO [3] utilizes separate thread pools for fast and slow paths and adopts a read-through cache instead of the more commonly used look-aside cache.

**Aligning microarchitecture metrics:** Since benchmarks are drastically simplified representations of real-world applications, perfect alignment between their microarchitecture performance characteristics is unrealistic. However, significant misalignment can serve as a useful indicator for identifying areas to improve benchmarks, as their refinement is a never-ending process.

**Putting benchmarks on the critical path:** Our three years of experience in developing and using DCPerf suggest that benchmark development is an iterative process requiring continuous investments to keep up with hardware evolution, workload changes, and emerging use cases. Through our exploration of existing benchmarks for potential adoption, we found that most had become outdated. Identifying business needs that rely on high-quality benchmarks on the critical path is essential, as it drives continuous improvements—a key factor behind DCPerf's success at Meta.

## 7 Related Work

**Benchmark suites.** Over the past decades, there have been continuous efforts to build benchmarks for various workloads [4, 10, 11, 14, 20, 24, 34, 41, 44, 46]. For example, SPEC [4], which has evolved for multiple generations, is the most popular suite for CPU benchmarking. In addition to SPEC CPU, which is focused on CPU core performance, it also has cloud-oriented benchmarks, such as SPEC Cloud IaaS, SPEC jbb, and SPECvirt. Another example is TPC [20], which models and benchmarks Online Transaction Processing (OLTP) workloads that use transactions per second as a metric for performance comparisons.

CloudSuite [11] provides a collection of cloud benchmarks and reveals several microarchitectural implications and the differences between scale-out workloads and SPEC workloads. BigDataBench [44] collects benchmarks at different levels, ranging from the simplest micro-benchmarks to full applications. Tailbench [24] focuses on the performance of latency-critical applications. $\mu$Suite [41] identifies four RPC-based OLDI applications with three microservice tiers (front-end, mid-tier, and leaf) to study OS and network performance overhead, particularly the mid-tier, which behaves as both an RPC client and an RPC server. DeathStarBench [14] is another recent benchmark suite for large-scale cloud applications with tens of RPC-based or RESTful microservices.

However, as discussed in Section 1 and Section 4.6, these benchmarks have limitations, such as benchmark categories do not match hyperscale workloads, mismatches with real-world software architecture, missing system components, limited scalability, and inadequate performance and power representativeness. For example, the benchmarks included in SPEC Cloud IaaS and SPEC jbb are significantly different from Meta's production workloads. Although CloudSuite contains relevant benchmarks, its benchmarks cannot scale on many-core servers, as described in Section 4.6.

**Profiling datacenter workloads.** Datacenter workloads have been studied from multiple aspects. Google conducted profiling of fleet-wide CPU usage [23], big-data processing systems [15], and the RPC stack [36]. Meta conducted studies on datacenter networks [2, 32], microservice architecture [21, 39], and hardware optimization and acceleration opportunities [39] [40]. DCPerf benefits from these profiling efforts by incorporating their findings into the benchmarks to represent production workloads.

## 8 Conclusion and Future Work

DCPerf is the first performance benchmark suite actively used to inform procurement decisions for millions of CPUs in hyperscale datacenters while also remaining open source. Our evaluation demonstrates that DCPerf accurately projects the performance of representative production workloads within a 3.3% error margin across multiple server generations. Our future work includes broadening DCPerf's coverage, especially AI-related workloads [49], whose fleet sizes have been expanding rapidly, and improving DCPerf's projection accuracy. We hope that DCPerf will inspire industry peers to open-source their well-calibrated benchmarks as well, such as those for search and e-commerce.

## Acknowledgments

We would like to thank numerous researchers and engineers from various teams at Meta who directly worked and/or provided input on DCPerf's design and implementation. We also thank our external CPU vendors and ISCA anonymous reviewers for their feedback and suggestions.

## References

[1] Daniel J Barrett. 2008. *MediaWiki: Wikipedia and beyond.* " O'Reilly Media, Inc.".
[2] Theophilus Benson, Aditya Akella, and David A. Maltz. 2010. Network traffic characteristics of data centers in the wild. In *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement (IMC '10)*.
[3] Nathan Bronson, Zach Amsden, George Cabrera, Prasad Chakka, Peter Dimov, Hui Ding, Jack Ferris, Anthony Giardullo, Sachin Kulkarni, Harry Li, et al. 2013. TAO: Facebook's distributed data store for the social graph. In *2013 USENIX Annual Technical Conference (USENIX ATC 13)*.
[4] James Bucek, Klaus-Dieter Lange, and Jóakim v. Kistowski. 2018. SPEC CPU2017: Next-Generation Compute Benchmark. In *Companion of the 2018 ACM/SPEC International Conference on Performance Engineering (ICPE '18)*.
[5] Emmanuel Cecchet, Anupam Chanda, Sameh Elnikety, Julie Marguerite, and Willy Zwaenepoel. 2003. Performance comparison of middleware architectures for generating dynamic web content. In *Proceedings of the 2003 ACM/IFIP/USENIX International Middleware Conference (Middleware'03)*.
[6] Mike Chow, Yang Wang, William Wang, Ayichew Hailu, Rohan Bopardikar, Bin Zhang, Jialiang Qu, David Meisner, Santosh Sonawane, Yunqi Zhang, Rodrigo Paim, Mack Ward, Ivor Huang, Matt McNally, Daniel Hodges, Zoltan Farkas, Caner Gocmen, Elvis Huang, and Chunqiang Tang. 2024. ServiceLab: Preventing Tiny Performance Regressions at Hyperscale through Pre-Production Testing. In *Proceedings of the 28th Symposium on Operating Systems Principles (SOSP'24)*.
[7] Cash Costello. 2012. *Elgg 1.8 social networking.* Packt Publishing Ltd.
[8] Custom Market Insights. 2024. Global Data Center CPU Market 2024–2033. https://www.custommarketinsights.com/report/data-center-cpu-market/
[9] DCPerf. 2024. https://github.com/facebookresearch/DCPerf
[10] Andrea Detti, Ludovico Funari, and Luca Petrucci. 2023. uBench: An Open-Source Factory of Benchmark Microservice Applications. *IEEE Transactions on Parallel and Distributed Systems* 34, 3 (2023).
[11] 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. In *Proceedings of the Seventeenth*

*International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'12).*

[12] Brad Fitzpatrick. 2004. Distributed caching with memcached. *Linux journal* 2004, 124 (2004), 5.

[13] Jeff Fulmer. 2022. Joedog/Siege: Siege is an HTTP load tester and benchmarking utility. https://github.com/JoeDog/siege

[14] 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 24th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'19).*

[15] Abraham Gonzalez, Aasheesh Kolli, Samira Khan, Sihang Liu, Vidushi Dadu, Sagar Karandikar, Jichuan Chang, Krste Asanovic, and Parthasarathy Ranganathan. 2023. Profiling Hyperscale Big Data Processing. In *Proceedings of the 50th Annual International Symposium on Computer Architecture (ISCA'23).*

[16] Google. [n. d.]. Snappy, a fast compressor/decompressor. https://google.github.io/snappy/

[17] Google. 2015. Benchmarking web search latencies. https://cloudplatform.googleblog.com/2015/03/benchmarking-web-search-latencies.html

[18] Alex Guzman, Kyle Nekritz, and Subodh Iyengar. 2021. Deploying TLS 1.3 at scale with Fizz, a performant open source TLS library. https://engineering.fb.com/2018/08/06/security/fizz/

[19] F Maxwell Harper and Joseph A Konstan. 2015. The movielens datasets: History and context. *ACM Transactions on Interactive Intelligent Systems (TIIS)* 5, 4 (2015).

[20] Trish Hogan. 2009. Overview of TPC Benchmark E: The Next Generation of OLTP Benchmarks. In *Performance Evaluation and Benchmarking.*

[21] Darby Huye, Yuri Shkuro, and Raja R. Sambasivan. 2023. Lifting the veil on Meta's microservice architecture: Analyses of topology and request workflows. In *2023 USENIX Annual Technical Conference (USENIX ATC 23).*

[22] Geonhwa Jeong, Bikash Sharma, Nick Terrell, Abhishek Dhanotia, Zhiwei Zhao, Niket Agarwal, Arun Kejariwal, and Tushar Krishna. 2023. Characterization of data compression in datacenters. In *Proceedings of the 2023 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS'23).*

[23] 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 Annual International Symposium on Computer Architecture (ISCA'15).*

[24] Harshad Kasture and Daniel Sanchez. 2016. Tailbench: a benchmark suite and evaluation methodology for latency-critical applications. In *2016 IEEE International Symposium on Workload Characterization (IISWC'16).*

[25] Ioannis Katsavounidis. 2015. NETFLIX – "EL Fuente" video sequence details and scenes. https://www.cdvl.org/media/1x5dnwt1/elfuente_summary.pdf

[26] Aaron Lu. 2023. sched/fair: Ratelimit update to tg->load_avg. https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/commit/?id=1528c661c24b407e92194426b0adbb43de859ce0

[27] Daniel A Menascé. 2002. TPC-W: A benchmark for e-commerce. *IEEE Internet Computing* 6, 3 (2002).

[28] Guilherme Ottoni. 2018. HHVM JIT: A Profile-guided, Region-based Compiler for PHP and Hack. In *Proceedings of the 39th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI'18).*

[29] Tapti Palit, Yongming Shen, and Michael Ferdman. 2016. Demystifying cloud benchmarking. In *Proceedings of the 2016 IEEE international symposium on performance analysis of systems and software (ISPASS'16).*

[30] Margaret H Pinson. 2013. The consumer digital video library [best of the web]. *IEEE Signal Processing Magazine* 30, 4 (2013).

[31] Will Reese. 2008. Nginx: the high-performance web server and reverse proxy. *Linux Journal* 2008, 173 (2008), 2.

[32] Arjun Roy, Hongyi Zeng, Jasmeet Bagga, George Porter, and Alex C. Snoeren. 2015. Inside the Social Network's (Datacenter) Network. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication (SIGCOMM'15).*

[33] RuBiS. 2024. https://github.com/uillianluiz/RUBiS

[34] Mohammad Reza Saleh Sedghpour, Aleksandra Obeso Duque, Xuejun Cai, Björn Skubic, Erik Elmroth, Cristian Klein, and Johan Tordsson. 2023. HydraGen: A Microservice Benchmark Generator. In *2023 IEEE 16th International Conference on Cloud Computing (CLOUD'23).*

[35] James Sedgwick. 2019. Wangle - an asynchronous C++ networking and RPC Library. https://engineering.fb.com/2016/04/20/networking-traffic/wangle-an-asynchronous-c-networking-and-rpc-library/

[36] Korakit Seemakhupt, Brent E. Stephens, Samira Khan, Sihang Liu, Hassan Wassel, Soheil Hassas Yeganeh, Alex C. Snoeren, Arvind Krishnamurthy, David E. Culler, and Henry M. Levy. 2023. A Cloud-Scale Characterization of Remote Procedure Calls. In *Proceedings of the 29th Symposium on Operating Systems Principles (SOSP'23).*

[37] Raghav Sethi, Martin Traverso, Dain Sundstrom, David Phillips, Wenlei Xie, Yutian Sun, Nezih Yegitbasi, Haozhun Jin, Eric Hwang, Nileema Shingte, et al. 2019. Presto: SQL on everything. In *Proceedings of the 2019 IEEE 35th International Conference on Data Engineering (ICDE'19).*

[38] Mark Slee, Aditya Agarwal, and Marc Kwiatkowski. 2007. Thrift: Scalable cross-language services implementation. *Facebook white paper* 5, 8 (2007), 127.

[39] Akshitha Sriraman and Abhishek Dhanotia. 2020. Accelerometer: Understanding Acceleration Opportunities for Data Center Overheads at Hyperscale. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems.*

[40] Akshitha Sriraman, Abhishek Dhanotia, and Thomas F. Wenisch. 2019. SoftSKU: Optimizing Server Architectures for Microservice Diversity @Scale. In *Proceeding of the 46th Annual International Symposium on Computer Architecture (ISCA'19).*

[41] Akshitha Sriraman and Thomas F. Wenisch. 2018. uSuite: A Benchmark Suite for Microservices. In *2018 IEEE International Symposium on Workload Characterization (IISWC'18).*

[42] Gábor Takács and Domonkos Tikk. 2012. Alternating least squares for personalized ranking. In *Proceedings of the 6th ACM conference on Recommender systems.*

[43] Lei Wang, Jianfeng Zhan, Chunjie Luo, Yuqing Zhu, Qiang Yang, Yongqiang He, Wanling Gao, Zhen Jia, Yingjie Shi, Shujie Zhang, et al. 2014. BigDataBench: A big data benchmark suite from internet services. In *Proceedings of the 2014 IEEE 20th international symposium on high performance computer architecture (HPCA'14).*

[44] Lei Wang, Jianfeng Zhan, Chunjie Luo, Yuqing Zhu, Qiang Yang, Yongqiang He, Wanling Gao, Zhen Jia, Yingjie Shi, Shujie Zhang, Chen Zheng, Gang Lu, Kent Zhan, Xiaona Li, and Bizhu Qiu. 2014. BigDataBench: A big data benchmark suite from internet services. In *Proceedings of the 20th International Symposium on High Performance Computer Architecture (HPCA'14).*

[45] Michael Widenius and David Axmark. 2002. *MySQL reference manual: documentation from the source.* " O'Reilly Media, Inc.".

[46] Yanan Yang, Xiangyu Kong, Laiping Zhao, Yiming Li, Huanyu Zhang, Jie Li, Heng Qi, and Keqiu Li. 2022. SDCBench: A Benchmark Suite for Workload Colocation and Evaluation in Datacenters. *Intelligent Computing* 2022 (2022).

[47] Ahmad Yasin. 2014. A top-down method for performance analysis and counters architecture. In *Proceedings of the 2014 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS'14).*

[48] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauly, Michael J Franklin, Scott Shenker, and Ion Stoica. 2012. Resilient distributed datasets: A Fault-Tolerant abstraction for In-Memory cluster computing. In *Proceedings of the 9th USENIX symposium on networked systems design and implementation (NSDI'12).*

[49] Mark Zhao, Niket Agarwal, Aarti Basant, Buğra Gedik, Satadru Pan, Mustafa Ozdal, Rakesh Komuravelli, Jerry Pan, Tianshu Bao, Haowei Lu, Sundaram Narayanan, Jack Langman, Kevin Wilfong, Harsha Rastogi, Carole-Jean Wu, Christos Kozyrakis, and Parik Pol. 2022. Understanding data storage and ingestion for large-scale deep recommendation model training: Industrial product. In *Proceedings of the 49th Annual International Symposium on Computer Architecture (ISCA'22).*