Cloudspecs: Cloud Hardware Evolution Through the Looking Glass

Till Steinert
till.steinert@tum.de
Technische Universität München

Maximilian Kuschewski
maximilian.kuschewski@tum.de
Technische Universität München

Viktor Leis
leis@in.tum.de
Technische Universität München

ABSTRACT

Public cloud hardware has evolved rapidly, with vendors offering hundreds of instance types differing in CPU, memory, storage, and networking capabilities. This paper presents a comprehensive analysis of cloud hardware trends from 2015 to 2025, focusing on AWS and comparing with other clouds and on-premise hardware. While network bandwidth improved per dollar by one order of magnitude, CPU and DRAM gains are much smaller and NVMe storage performance has stagnated since 2016. A cross-cloud price comparison shows that smaller providers undercut the "big three" hyperscalers by up to 5× for commodity VMs. Our findings reveal shifting bottlenecks and highlight new challenges in designing cost-efficient cloud-native data-intensive systems. An interactive tool supports further exploration of the evolving instance landscape.

1 INTRODUCTION

Shift to the cloud. A growing number of database systems are now deployed on public cloud platforms such as Amazon Web Services (AWS), Microsoft Azure, and Google Cloud [8, 11, 14, 36]. In these environments, database systems typically run on virtual machine instances connected via Ethernet. As a result, the system's performance, scalability, and cost are constrained by the available instance types and their associated prices [28, 34].

Cost-efficiency vs. system architecture. In the cloud, what often matters most is not peak performance, but performance per dollar. This shifts the focus from raw throughput to economic efficiency, making cost a first-class design consideration. For instance, depending on the relative pricing of CPU, RAM, SSD storage, and networking, it may or may not be cost-effective to cache data or to disaggregate the system into separate compute, caching, and storage layers [10]. Designing cost-efficient cloud systems therefore requires a deep understanding of the cloud hardware landscape – including the range of instance types, storage options, and pricing models offered by cloud providers.

A vast, multi-dimensional landscape. Cloud providers offer hundreds of virtual machine instance types, with new configurations introduced regularly. As Figure 1 shows, since 2015 the number of AWS instance types has grown from 52 to 1,057. Google Cloud currently offers over 400 instance types and Microsoft Azure more than 1,000. Instances differ in key dimensions such as CPU type, memory size, network bandwidth, and storage capacity – all of which impact both performance and cost. Navigating this complex and ever-changing space is a core challenge in cloud system design.

Contributions. The goal of this paper is to demystify the cloud hardware landscape by analyzing current cloud offerings and tracing their evolution over the past decade. We focus our analysis on AWS, which is the largest cloud vendor, but also compare with on-premise hardware and other cloud vendors. In addition, we discuss the implications of this evolving hardware ecosystem for data-intensive systems. Given the multi-dimensional nature of the cloud instance space, any static analysis is inherently limited. To support deeper exploration beyond the paper, we therefore provide an interactive visualization tool that enables cloud system architects to answer specific instance selection questions rigorously. Our tool is available at `https://cloudspecs.fyi`.

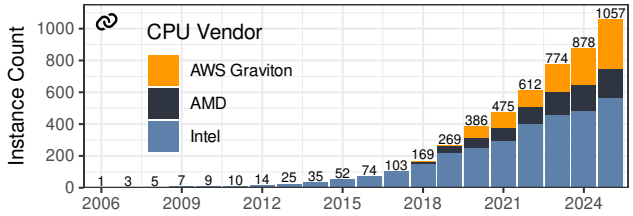

Figure 1: Available AWS Instances by CPU vendor & year

Note: Link symbols (🔗) in figures are clickable, pointing to an interactive online reproducibility interface for that figure.

2 CLOUD HARDWARE TRENDS

Data sources. AWS publishes extensive hardware specifications through several overlapping but incomplete APIs and websites. For example, the *Describe Instances* API contains many important hardware specifications, while some instance storage performance data is only available on a webpage and prices are available through the *Price List* API. We therefore extended the open source *instances.vantage.sh* data crawler with additional data sources like the *instancetyp.es* website that contains instance release dates to produce a curated database combining all information relevant to this paper. This process is fully automatic and we plan to maintain this database in the future to facilitate tracing the evolution of cloud hardware. We use data as of October 2025.

Instance families and slices. AWS allows renting "virtual slices" of the same physical server. The *i3* instance family, for example, is available in the *large*, *xlarge*, *2xlarge*, *4xlarge*, *8xlarge*, and *16xlarge* sizes. Within a family, the price and hardware generally scale proportionally, e.g., *2xlarge* costs twice as much as *xlarge* and also has twice as many CPU cores, DRAM, instance storage capacity, etc. Thus, in most cases the cost-normalized metrics (e.g., cores per dollar) are exactly equal for all instance slices within one family. Whenever this is not the case (e.g., network bandwidth), we show the largest slice. For most analyses, it is therefore sufficient to show only the family name (*i3*) rather than the full instance name *i3.2xlarge*.

 16th Annual Conference on Innovative Data Systems Research (CIDR '26). January 18-21, 2026, Chaminade, USA.

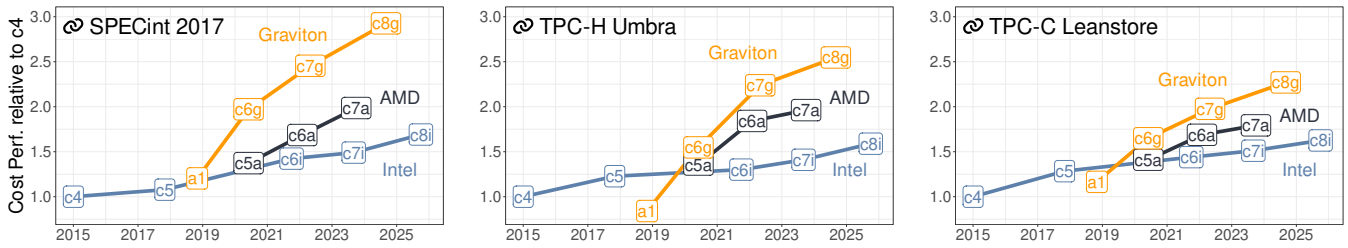

Figure 2: Relative cost-performance evolution of SPECint (), Umbra (), and Leanstore () by instance release date

Instance selection. Because we focus on common database use cases, the main analysis excludes instances with accelerators and FPGAs, and the economics of GPU instances are discussed separately in Section 2.5. We also ignore instances with shared/overcommitted CPUs (i.e., burstable *t* and *flex* instances). This leaves us with 742 relevant instances across 98 instance families, which differ in terms of CPU, RAM, storage, networking capabilities, and pricing. Note that our online database still contains *all* instances, allowing more detailed analyses of GPUs and accelerators in the future.

Prices. Given how easy the cloud makes switching instances, comparing instance specifications without taking into account their prices is misleading. In most cases, we therefore normalize instance metrics relative to an instance's hourly price. We use prices from the *us-east-1* region (North Virginia), which is the largest and often also the cheapest region. While AWS offers multiple discount models, we use on-demand prices as the basis of comparison. Spot instance prices are highly variable based on short term availability fluctuations. The discount rates of savings and reserved instances are broadly similar across instance types. Using on-demand prices will therefore merely overestimate the true cost by some factor but will not distort the overall price structure. Also note that AWS has not decreased on-demand prices for any instance since 2018. Disruptions occur through the introduction of new instance types and the slow obsolescence of older ones.

2.1 CPU Trends

More cores. Just like the on-premise world, the cloud has seen tremendous growth in multi-core parallelism. The biggest AWS instance, *u7in-24tb.224xlarge*, has 448 cores. Focusing on compute-optimized instances only, the maximum core count increased by one order of magnitude over the past ten years: The *c4* instance family, introduced in 2015, provides a maximum of 18 cores, while the *c7a* family, introduced in 2023, features up to 192 cores with the *c7a.48xlarge* instance. However, as we will see, this increase does not translate to a higher performance per dollar.

More CPU vendors. A second major CPU trend, shown in Figure 1, is that Intel is not the only CPU vendor any longer. AMD CPUs and particularly the ARM-based AWS Graviton CPUs have become common as well. Indeed, since 2024 more than half of newly-released instances use Graviton CPUs.

CPU benchmarking. Having dozens of CPU models that differ in terms of their ISA (x86, ARM), vendor (Intel, AMD, AWS), and CPU generation (e.g., Epyc 1 through Epyc 5) makes choosing the best instance difficult. Adding to this difficulty is the fact that AWS specifies the degree of parallelism as *"vCPUs"*, which, depending on the instance type, represent either hyperthreads (most x86 instances) or full cores (all Graviton instances and *c7a*). To compare CPU performance more accurately, we ran three benchmarks: the CPU performance benchmarking suite SPECint 2017 [7], in-memory TPC-H (scale factor 10) on the database system Umbra [29], and in-memory TPC-C (25 warehouses) on the storage engine Leanstore [24]. We ran the experiments on the *2xlarge* slices of compute-optimized instances using all available hardware threads.

CPU cost-performance evolution. To see how cost-performance evolved over the last 10 years, we normalize the results by the performance and cost of the *c4* instance such that higher values indicate better cost-performance. Figure 2 shows the results for the three multi-threaded benchmarks. For SPECint, cost-performance improves by about 3× over ten years. A big part of this improvement is due to the introduction of Graviton instances, without which the gain would be closer to 2×. It is therefore not surprising that major cloud-native systems such as Snowflake have been switching to Graviton instances [33]. For the in-memory database benchmarks, the gains are even lower, ranging from 2× to 2.5×. We speculate that this is because in-memory database performance is strongly affected by memory and cache latency, but leave a detailed microarchitectural analysis for future work.

On-premise comparison. Historically, Moore's law (more transistors per area and therefore per dollar) and Dennard scaling (higher performance via higher clock frequencies) led to cost-normalized CPU performance doubling roughly every two years. Over a decade, this would imply five doublings and a cost-performance gain of 32×. This raises the question of whether the relative CPU cost-performance stagnation is due to higher margins for hyperscalers or due to technological stagnation, i.e., "the end of Moore's law". To answer this, we analyzed AMD server CPUs from 2017 to 2025. Based on list prices, core counts, clock frequencies, and published IPC metrics, we observe a similar trend (1.7×) with on-premise AMD server CPUs as within AWS. Thus, it seems that the cost-performance stagnation for server CPUs is not cloud-specific, but a general tectonic shift in the technological landscape.

2.2 Main Memory

Memory capacity. Besides core count, AWS instances also vary greatly in their DRAM capacity. The cost-normalized DRAM capacity, shown in Figure 3, exhibits very little improvement over time. The only major change was the introduction of an additional

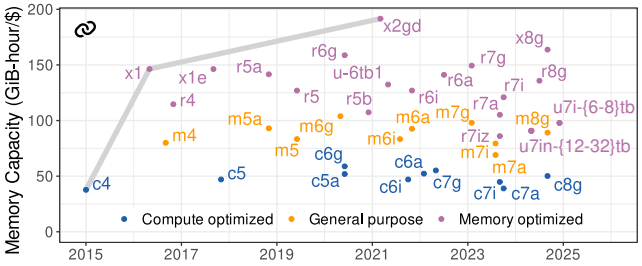

Figure 3: Memory capacity development

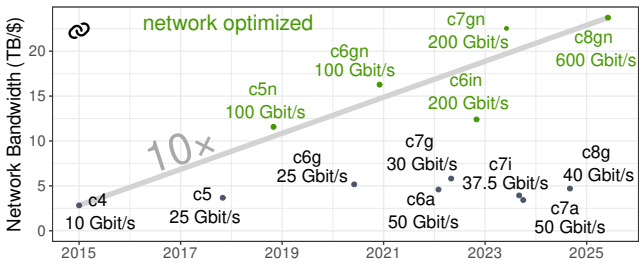

Figure 4: Network bandwidth development

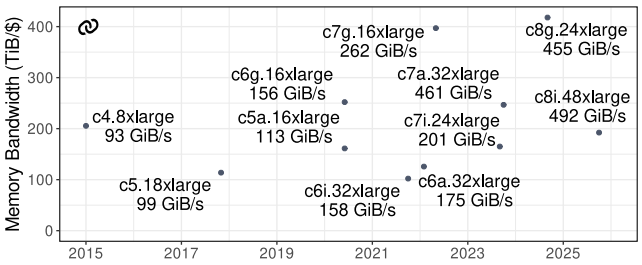

Figure 5: Memory bandwidth development

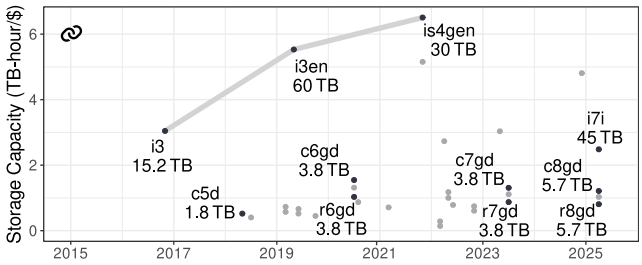

Figure 6: NVMe SSD capacity development

group of *memory-optimized* instances (with an *x* prefix) in 2016, which offer roughly 3.3× more GiB-hours/$ than the best compute-optimized instances. AWS also offers instances with very large DRAM capacities of up to 32 TiB (*u7in-32tb.224xlarge*), though their capacity per dollar is significantly worse than the *x* and *r* instances, as Figure 3 shows.

Memory bandwidth. During the past ten years, server CPUs transitioned from DDR3 to DDR5 while supporting more memory channels and higher transfer speeds. Because AWS does not publish memory bandwidth specifications, we measured the memory bandwidth on compute-optimized instances. The results are shown in Figure 5. While the absolute single-CPU-socket bandwidth increased from 93 GiB/s to 492 GiB/s ($\hat{\approx}$ 5×), in cost-normalized terms, we see a gain of 2×.

On-premise comparison. We again ask the question how these trends compare with on-premise hardware. In terms of capacity, historical data [2] suggests that commodity DRAM prices decreased by roughly 3× during the past ten years. On-premise memory bandwidth per dollar has similarly improved by a factor of three (when accounting for DRAM price fluctuations). Thus, both memory capacity and bandwidth cost improvements in AWS closely track the corresponding developments in the non-cloud server market.

2.3 Networking

Ethernet gains. Besides CPU and main memory, the only other hardware resource that all instances have is Ethernet. Figure 4 shows the evolution of cost-normalized network bandwidth for compute-optimized instances. In terms of absolute network bandwidth, we see an improvement from 10 Gbit/s to 600 Gbit/s (60×). Even in cost-normalized terms, networking bandwidth improved by 10×. This change stems from the introduction of network-optimized instances (with the *n* suffix). The first such instance based on proprietary Nitro virtualization technology, *c5n*, was introduced in 2018. For non-network optimized instances (e.g., c4, c5, c6a, c7g) the cost-normalized network bandwidth barely changed.

On-premise comparison. Looking at Ethernet network cards from Mellanox, we find that their speed has increased from 100 Gbit/s (ConnectX-4, released in 2014) to 400 Gbit/s (ConnectX-7, released in 2022) [18]. We cannot compare the cost-normalized evolution of cloud network hardware to on-premises, because we were not able to find reliable and realistic prices for data center networking hardware (e.g., 200 Gbit/s Ethernet switches). This raises the question of whether economic considerations motivated AWS to manufacture proprietary Ethernet technology, allowing them to advance technology more quickly than on-premise alternatives.

2.4 NVMe Instance Storage

Remote vs. local storage. Most modern AWS instances do not have built-in storage but rely on remote block devices (*EBS* in AWS) attached through some form of network. However, there are also instances that have built-in storage devices, which can be magnetic disks, SATA SSDs, or NVMe SSDs. We focus on instance storage based on NVMe SSDs, which have evolved considerably over the past decade, and are of particular importance for database systems due to being much faster than older storage types.

NVMe capacity. The first instance family with NVMe SSDs, *i3*, was introduced in 2016. As of 2025, there are 36 such instance families. As Figure 6 shows, the first disruption after *i3* in terms of storage capacity per dollar was the introduction of the *i3en* instance family in 2019, which offers up to 60 TB of capacity and doubles storage capacity per dollar. Since then, we see very little change despite the introduction of dozens of new instances with NVMe SSDs.

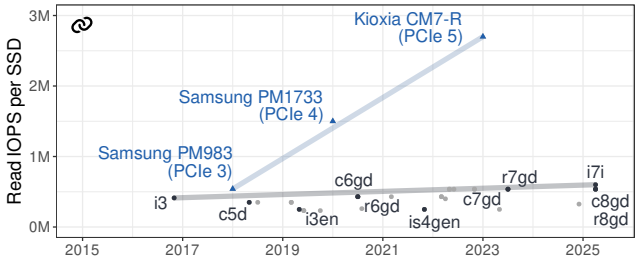

Figure 7: AWS vs. on-premise NVMe SSD performance

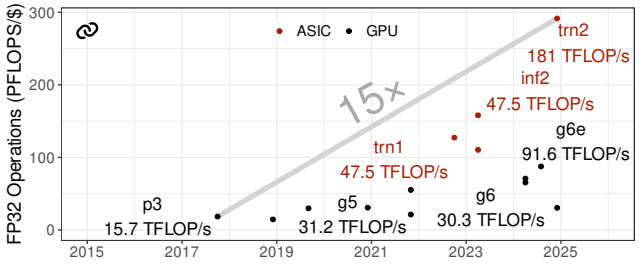

Figure 8: FLOPs per Dollar for GPUs and accelerators

NVMe performance. Remarkably, if we look at I/O performance per dollar, we find that the *i3* instance, introduced in 2016, is still the best instance by almost 2×. This is true both in terms of sequential read throughput and random 4 KiB reads.

On-premise comparison. The stagnation of SSD capacity (since 2019) and I/O throughput (since 2016) stands in stark contrast to on-premise trends. Cost-normalized NVMe capacities have increased by roughly 3× in the same time frame and I/O throughput by over 5×. The growing gap between cloud and on-premise NVMe performance is illustrated in Figure 7, which compares the read IOPS for one NVMe device. While in 2018, AWS NVMe SSD performance was state of the art (e.g., *i3* vs. *Samsung PM983*), this is no longer true. Since 2020, on-premise SSD performance doubled twice with the introduction of PCIe 4 (e.g., *Samsung PM1733*) and PCIe 5 (e.g., *Kioxia CM7-R*) devices, while AWS NVMe performance has stagnated. The recently-announced next generation of PCIe 6 devices (e.g., *Micron 9650*) is projected to increase this gap even further.

2.5 GPUs and Accelerators

As discussed in Section 2.1, cost-performance of CPUs has mostly stagnated, suggesting that hardware specialization may be economical [35]. To investigate this, we compare the development of cost-normalized 32-bit floating point operations over time. For GPU instances, cost-performance has improved by roughly 4.7× from the *p3* to *g6e* families. Since 2019, AWS has also released two generations of instance types (*inf* and *trn*) featuring custom-built machine learning accelerators (ASIC). As Figure 8 shows, the *trn2* accelerator family has twice as many FLOP per second per device as the *g6e* family and 15.7× higher cost-performance than *p3* – indicating that hardware specialization is currently advantageous.

Table 1: Cloud comparison (AMD, 16 cores, 128 GiB RAM)

Cloud		Instance	AMD	$/hour
AWS	US	m6a.8xlarge	Epyc 3	1.3824
Google	US	t2d-standard-32	Epyc 3	1.3519
Azure	US	D32as v5	Epyc 3	1.3760
AWS	DE	m6a.8xlarge	Eypc 3	1.6560
Oracle	US	VM.Standard.E4.Flex	Epyc 3	0.5920
STACKIT	DE	g1a.32d-EU01	Epyc 2	1.5203
OVHCloud	DE	B3-128	Epyc 3	0.8554
Hetzner	DE	CCX53	Epyc 3/4	0.3548

2.6 Comparing Clouds

Cross-cloud comparison. Next, we extend our analysis to other clouds. Because an exhaustive comparison of cloud providers is beyond the scope of this paper, we restrict ourselves to comparing a widely available reference configuration. To do that, we chose instances with 16 core AMD CPUs (32 hyperthreads), 128 GiB of RAM, and without special features like high network bandwidth. We manually collected tax-adjusted prices across seven public clouds in the US and Europe, which are summarized in Table 1.

Hyperscalers. Focusing on the three major hyperscalers (AWS, Google, Azure), we see similar prices for our chosen configuration. Generally, the prices, pricing models, and offerings of three major hyperscalers are very similar to each other. For example, all three clouds have instances with NVMe storage, charge similar amounts for inter-region network traffic, and offer Intel, AMD, and ARM CPUs. There are exceptions to this rule: for example, AWS has an edge in terms of networking speed with its *c7gn*/*c8gn* instances, while Azure does not charge for network traffic between availability zones of the same region. We plan to extend the Cloudspecs database with information from other clouds and leave a detailed comparison of the strengths and weaknesses for future work.

Smaller clouds. The similarity of prices between the three major hyperscalers could indicate a highly-competitive market where competition drives prices of basic infrastructure like common virtual machines types close to marginal cost. However, a price comparison with other clouds seems to falsify this theory. For example, Germany-based Hetzner (*CCX53*) is 3.9× cheaper than AWS (*m6a.8xlarge*) in the *us-east-1* region. The gap increases to 4.7× if we compare Hetzner with AWS in *eu-central-1*. Oracle and OVHCloud also manage to be substantially cheaper than the big three. STACKIT, on the other hand, appears to follow the pricing level of the hyperscalers.

Other factors. Admittedly, hyperscalers offer additional benefits to customers that are hard to match by smaller clouds. In particular, they offer a much larger set of specialized instance types, which can directly translate to lower operational cost for certain use cases. Furthermore, smaller clouds have fewer data centers, which can be important for geo-distributed applications, often charge by hour rather than per minute or second, and have slower network speeds. On the other hand, the three hyperscalers charge extremely high prices for moving data out of their cloud. For example, even ignoring Hetzner's 20TB/month free traffic per instance, AWS charges 50× more for egress per byte than Hetzner.

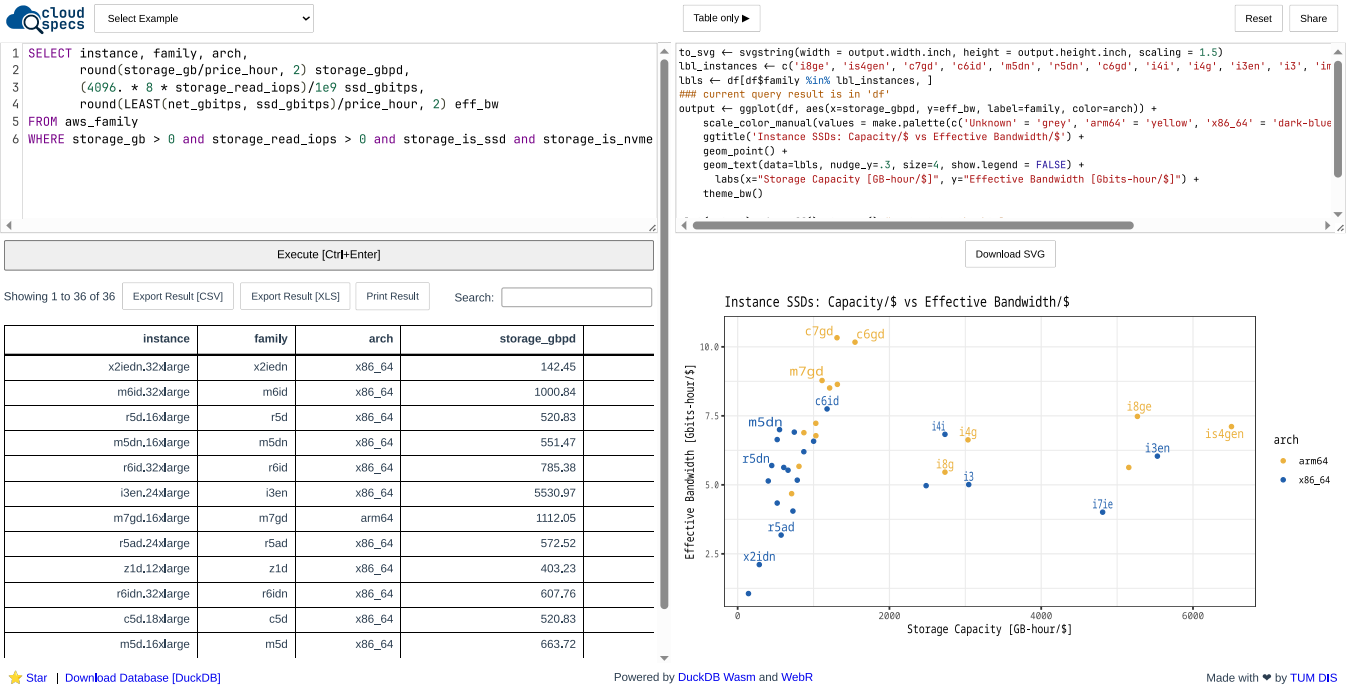

Figure 9: The Web UI of Cloudspecs

3 CLOUDSPECS

The need for use-case-specific analysis. Thus far, we have focused on high-level hardware trends and general guidelines. However, any analysis we can conduct within this paper is necessarily limited. To answer use-case-specific questions, systems engineers must therefore build upon these guidelines and apply economic reasoning to their own cloud-native system architecture.

Lightweight and self-contained data exploration. To facilitate such analyses, we have built Cloudspecs, an open-source interactive data exploration and visualization website. Cloudspecs is built on the WebAssembly technologies DuckDB-WASM [21] and WebR [31], allowing it to run entirely in the browser without server-side computation. As Figure 9 shows, Cloudspecs provides a browser-based SQL interface to a curated database of AWS instances, as well as an R-based data visualization interface. By enabling engineers to write arbitrary SQL and R code, Cloudspecs facilitates complex analyses including aggregations, window functions, nested expressions, interactive plots, and R model fittings.

A curated database of cloud instances. Cloudspecs collects AWS instance specifications from multiple sources and provides them as a curated database. Furthermore, this data is preprocessed and enriched with our own benchmarks. For example, AWS often only specifies an upper bound for network bandwidth (e.g., *"Up to 10 Gbit/s"*) for smaller instances, which we split into *net_peak_gbitps* (the maximum token bucket burst bandwidth) and *net_gbitps* (the guaranteed baseline bandwidth). Moreover, Cloudspecs provides convenience views like *aws* (the subset of instances defined in Section 2) and *aws_family* (the largest non-metal instances per family) for frequently queried instance subsets. The resulting database is a single DuckDB file that can also be downloaded for offline analysis. Our data crawling and preprocessing pipeline is fully automated and we plan to maintain and extend the tool in the future. The Cloudspecs codebase is available on GitHub [1].

3.1 Demonstration

Scenario. In the following, we will demonstrate how systems engineers can use Cloudspecs to make informed decisions about economic hardware selection. Our scenario describes the workflow for selecting a cost-optimal instance for a storage node in a disaggregated key-value store similar to DynamoDB [11]. The storage node has two main objectives for which we will optimize: (1) persistently storing data and (2) serving key-value requests over the network.

Architectural assumptions. For this simplified demonstration, we assume that the hardware can be perfectly utilized, and that values have a fixed-size of ≤ 4 KiB. When the storage node receives a GET request, it (1) performs a B-Tree lookup on the key, (2) reads the requested value from instance-local storage, and (3) returns the value over the network. For simplicity, we assume that all requests access local storage (i.e., there is no in-memory buffer cache) and there is no compression. Because the CPU work is negligible in this scenario and the clear bottleneck is I/O, the processor architecture and generation of the instance is of little importance.

Economic analysis. Given these assumptions, there are two main economic dimensions to consider: (1) storage capacity per dollar, and (2) I/O operations per dollar. Since data is always read from the local storage and sent over the network, the storage nodes' throughput is bounded by the effective bandwidth BW_{eff}, computed as $\text{BW}_{\text{eff}} = min(\text{BW}_{\text{storage}}, \text{BW}_{\text{network}})$. That is, choosing an

instance with a higher network bandwidth does not increase the storage nodes' throughput if storage bandwidth is the bottleneck.
Step 1: Filtering SSD instances. As Section 2.2 shows, main memory in the cloud is very expensive, so data should be persisted on instance-local storage. Furthermore, hard disks are not suitable since our workload randomly accesses small 4 KiB pages. As a first step, we thus filter for NVMe SSD instances (columns *storage_is_ssd* and *storage_is_nvme*), leaving us with 287 potential instances to consider.
Step 2: Filtering slices. Many of the smaller instances in our result set have burstable networking – that is, one can briefly utilize a high peak network bandwidth before being throttled to some baseline bandwidth. Since fluctuating network throughput is undesirable for a key-value store service, we filter out such instances. Furthermore, because cost-normalized metrics (e.g., network bandwidth per dollar) are the same for all slices within a family, it is sufficient to compare instance families as a whole. We do so by comparing only the largest instances of each instance family, none of which suffer from burstable networking. Cloudspecs provides this commonly used set of instances in the convenience view `aws_family`, selecting from which leaves us with 36 candidate instances.
Step 3: Pareto-optimal instances. We visualize the remaining 36 instances as a scatter plot across the two main economic dimensions of interest, storage capacity per dollar and I/O operations per dollar. This plot, visible on the right side of Figure 9, clearly shows that there are four pareto-optimal instances for our use case: The throughput-optimal *c7gd* and *c6gd* instances and the capacity-optimal *is4gen* and *i8ge* instances. Notably, all four instances are Graviton-based. The final choice of instance depends on the average customer workload: If customers access their data infrequently on average, storage cost becomes more important, so one should pick an instance from the *is4gen*/*i8ge* families; otherwise, the *c7gd* or *c6gd* families are preferable.
Step 4: Export result. Finally, we can export our analysis. We can do so by either (1) downloading the resulting table as a CSV or excel sheet, (2) downloading the plot as an SVG image, or (3) saving the dashboard link or sharing it with a coworker. When saving the link, the analysis will automatically stay up to date as new instance types are introduced and captured by our automated crawling pipeline. Note that the links in this paper point to a fixed snapshot of Cloudspecs to ensure reproducibility; the up-to-date instance database is available at `https://cloudspecs.fyi`.

3.2 Towards Interactive Data Sharing

Cloudspecs is self-contained and runs entirely in the browser. It is hosted on a static file server without external dependencies. The database and sample queries are trivially interchangeable. Furthermore, link sharing improves reproducibility by directly linking source code for plots from within a paper. Our GitHub repository contains code for setting up a Cloudspecs-like website and an example LaTeX package for creating clickable figures. For these reasons, we believe that Cloudspecs can be generalized to generic, self-contained, interactive research data sharing packages.

4 RELATED WORK

Economics of the cloud. Given the close coupling of resources and pricing models in the cloud, it is unsurprising that cloud economics have been studied from both consumer [5] and vendor [13] perspectives. Jim Grays' five minute rule [12] has recently been revisited in the context of caching in cloud analytics systems [10]. Similar models study the architecture of analytical [25, 37] and transactional [17] cloud systems. Research efforts like OptimusCloud [28] and CherryPick [4] make cost a first-class consideration for systems design. Cloudspecs can aid researchers with prototyping and as a data source for similar analytical models.
Cloud hardware benchmarking. A plethora of prior work has experimentally studied cloud hardware from microarchitectural [19, 22], network [30, 41], and security [27] perspectives.

5 LESSONS FOR CLOUD SYSTEMS

Gains from specialization. Cloud hardware is becoming more diverse, which makes instance selection more difficult but also leads to large opportunities. Selecting an instance optimized for a particular resource bottleneck such as *c8gn* for networking over a more generic one such as *c8g* can improve performance by 5×. This assumes that the software is capable of exploiting all hardware resources.
Scale-up opportunities. Although the cloud is strongly associated with distributed systems, it also enables another form of specialization: extreme scale up. In AWS, instances with 448 cores, 32 TiB of DRAM, 120 TB of NVMe storage, and 600 Gbit/s network bandwidth are available. The availability of such large instances also makes it possible to design large single-node systems.
ARM is taking over. In comparison with prior decades, x86 CPU performance per dollar has stagnated. The biggest compute cost improvement comes from the introduction of Graviton instances. Within a handful of years, ARM instances went from being an obscure niche option to the default choice. Today, Graviton-based instances are not only the best choice in terms of compute (*c8g*), but also network bandwidth (*c8gn*), main memory capacity (*x2gd*), and NVMe capacity (*is4gen*). Similar trends can be observed in other clouds as well. It currently seems like developers should seriously consider porting their systems to ARM. However, the jury is still out on the long-term impact on CPU vendor diversity and prices.
Memory and NVMe. Main memory prices have largely stagnated since 2016. Data-intensive applications can resort to NVMe, which was the major disruption in the storage space in the past decade. Of course, NVMe SSDs have very different characteristics from DRAM, potentially requiring major architectural changes [15, 16].
Network bandwidth explosion. By far the largest gains in cost-normalized terms over the past decade was for networking, which improved by one order of magnitude in AWS. Given the relative stagnation of other hardware resources, this challenges the traditional assumption that network bandwidth is the bottleneck in distributed systems, and can have major architectural implications. For example, the AWS Aurora OLTP architecture was specifically designed to minimize network traffic, which raises the question of whether today's hardware landscape would favor other designs [17, 36]. As CPU speeds stagnate and network bandwidth increases, the CPU cycle budget per network packet continues to decline.

Fully exploiting this resource may require major system changes such as kernel bypassing [18, 26, 39, 40].

NVMe performance stagnation. Broadly, most of the hardware trends in the cloud follow similar developments as the general server hardware market. One major exception is the stagnation of NVMe performance in AWS. Together with the simultaneous explosion of network speeds, this raises new architectural questions. For example, cloud native warehouses such as Snowflake use NVMe as caches, but given fast networks, it may be more economical to directly read from S3 [9, 23].

Hardware accelerators. Given the stagnation of CPU speeds, specialized accelerators have the potential for major disruption. Indeed, recent work on leveraging GPUs for database systems [6, 20, 38] suggests GPUs are becoming increasingly relevant for data processing systems. Our analysis shows that accelerators could offer additional optimization potential. Because other performance characteristics like GPU memory capacity and bandwidth (both inter-GPU and between instances via the network) also impact instance choice, building cost-optimal machine learning systems in the cloud requires a model-based approach [17, 25].

Multi-cloud opportunities. We saw that alterative clouds present substantial opportunities for cost savings in comparison with the three market leaders. Optimizing multi-cloud architectures while considering data movement cost [32] therefore has great potential.

6 FUTURE WORK

Cloudspecs: An interactive approach. Our lessons represent a snapshot of the current cloud hardware landscape, which may change rapidly. A disruptive new instance type (e.g., featuring PCIe 5 SSDs with 7× the current bandwidth) could render some architectures relying exclusively on networking outdated over night. Cloudspecs allows data systems engineers and researchers to revisit their assumptions about the cloud hardware design space. We plan to keep Cloudspecs up-to-date with new instance releases and benchmarks for this purpose. The up-to-date version of Cloudspecs is available at `https://cloudspecs.fyi`.

Interactive data sharing. We believe that the Cloudspecs framework is also useful for other use cases. As discussed in Section 3.2, researchers can utilize the Cloudspecs framework to set up their own interactive reproducibility sites that function fully in the browser. A researcher merely needs to replace the Cloudspecs database with their own DuckDB file and host the site on a static file server such as GitHub Pages.

Database benchmarks. Benchmarking leaderboards like ClickBench [3] have enjoyed tremendous success in the database community. However, such leaderboards often struggle with concisely displaying the large number of dimensions inherent to the benchmarking data, such as different hardware configurations, hot runs vs. cold runs, preloaded vs. remote data, self-hosted vs. serverless offerings, single- vs. multi-node, and more. Leaderboards such as ClickBench could support more complex analysis by providing an interactively queryable database and visualization tool similar to Cloudspecs.

Multi-cloud comparison. While our analysis focused mainly on AWS, our methodology can be applied to other cloud providers as well. We intend to extend our analysis and subsequently add additional cloud providers to the Cloudspecs database. However, true cost-performance transparency of cloud providers requires a comparison of performance per dollar across instance types and clouds. Unfortunately, cloud providers do not publish the necessary hardware benchmark results themselves. Cloudspecs already contains some benchmark results such as the SPECint, TPC-H, and TPC-C performance numbers shown in Section 2.1, and we plan to add additional microbenchmarks and cross-cloud support in the future. We therefore believe that a tool like Cloudspecs can help to bridge the gap between benchmarking and cross-cloud cost-performance transparency.

ACKNOWLEDGMENTS

Funded/Co-funded by the European Union (ERC, CODAC, 101041375). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Research Council. Neither the European Union nor the granting authority can be held responsible for them.

REFERENCES

[1] 2025. https://github.com/TUM-DIS/cloudspecs/.
[2] January 15, 2025. https://web.archive.org/web/20250115012702/https://jcmit.net/memoryprice.htm.
[3] Alexey Milovidov. 2022. https://benchmark.clickhouse.com/.
[4] Omid Alipourfard, Hongqiang Harry Liu, Jianshu Chen, Shivaram Venkataraman, Minlan Yu, and Ming Zhang. 2017. CherryPick: Adaptively Unearthing the Best Cloud Configurations for Big Data Analytics. In *NSDI*. USENIX Association, 469–482.
[5] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy H. Katz, Andy Konwinski, Gunho Lee, David A. Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia. 2010. A view of cloud computing. *ACM* 53, 4 (2010), 50–58.
[6] Nils Boeschen, Tobias Ziegler, and Carsten Binnig. 2024. GOLAP: A GPU-in-Data-Path Architecture for High-Speed OLAP. *Proc. ACM Manag. Data* 2, 6 (2024), 237:1–237:26.
[7] James Bucek, Klaus-Dieter Lange, and Jóakim von Kistowski. 2018. SPEC CPU2017: Next-Generation Compute Benchmark. In *ICPE Companion*. 41–42.
[8] Benoît Dageville, Thierry Cruanes, Marcin Zukowski, Vadim Antonov, Artin Avanes, Jon Bock, Jonathan Claybaugh, Daniel Engovatov, Martin Hentschel, Jiansheng Huang, Allison W. Lee, Ashish Motivala, Abdul Q. Munir, Steven Pelley, Peter Povinec, Greg Rahn, Spyridon Triantafyllis, and Philipp Unterbrunner. 2016. The Snowflake Elastic Data Warehouse. In *SIGMOD Conference*. ACM, 215–226.
[9] Dominik Durner, Viktor Leis, and Thomas Neumann. 2023. Exploiting Cloud Object Storage for High-Performance Analytics. *Proc. VLDB Endow.* 16, 11 (2023), 2769–2782.
[10] Kira Duwe, Angelos Anadiotis, Andrew Lamb, Lucas Lersch, Boaz Leskes, Daniel Ritter, and Pınar Tözün. 2025. The Five-Minute Rule for the Cloud: Caching in Analytics Systems. In *CIDR*.
[11] Mostafa Elhemali, Niall Gallagher, Nick Gordon, Joseph Idziorek, Richard Krog, Colin Lazier, Erben Mo, Akhilesh Mritunjai, Somasundaram Perianayagam, Tim Rath, Swami Sivasubramanian, James Christopher Sorenson III, Sroaj Sosothikul, Doug Terry, and Akshat Vig. 2022. Amazon DynamoDB: A Scalable, Predictably Performant, and Fully Managed NoSQL Database Service. In *USENIX ATC*. 1037–1048.
[12] Jim Gray and Gianfranco R. Putzolu. 1987. The 5 Minute Rule for Trading Memory for Disk Accesses and The 10 Byte Rule for Trading Memory for CPU Time. In *SIGMOD*. 395–398.
[13] Albert G. Greenberg, James R. Hamilton, David A. Maltz, and Parveen Patel. 2009. The cost of a cloud: research problems in data center networks. *Comput. Commun. Rev.* 39, 1 (2009), 68–73.
[14] Anurag Gupta, Deepak Agarwal, Derek Tan, Jakub Kulesza, Rahul Pathak, Stefano Stefani, and Vidhya Srinivasan. 2015. Amazon Redshift and the Case for Simpler Data Warehouses. In *SIGMOD Conference*. ACM, 1917–1923.
[15] Gabriel Haas, Michael Haubenschild, and Viktor Leis. 2020. Exploiting Directly-Attached NVMe Arrays in DBMS. In *CIDR*.
[16] Gabriel Haas and Viktor Leis. 2023. What Modern NVMe Storage Can Do, And How To Exploit It: High-Performance I/O for High-Performance Storage Engines. *Proc. VLDB Endow.* 16, 9 (2023), 2090–2102.
[17] Michael Haubenschild and Viktor Leis. 2025. Oltp in the cloud: architectures, tradeoffs, and cost. *VLDB J.* 34, 4 (2025), 42.

[18] Matthias Jasny, Muhammad El-Hindi, Tobias Ziegler, and Carsten Binnig. 2025. A Wake-Up Call for Kernel-Bypass on Modern Hardware. In *DaMoN*. 14:1–14:5.

[19] Qingye Jiang, Young Choon Lee, and Albert Y. Zomaya. 2020. The Power of ARM64 in Public Clouds. In *CCGRID*. IEEE, 459–468.

[20] Marko Kabic, Bowen Wu, Jonas Dann, and Gustavo Alonso. 2025. Powerful GPUs or Fast Interconnects: Analyzing Relational Workloads on Modern GPUs. *Proc. VLDB Endow.* 18, 11 (2025), 4350–4363.

[21] André Kohn, Dominik Moritz, Mark Raasveldt, Hannes Mühleisen, and Thomas Neumann. 2022. DuckDB-Wasm: Fast Analytical Processing for the Web. *Proc. VLDB Endow.* 15, 12 (2022), 3574–3577.

[22] Leonardo X. Kuffo and Peter A. Boncz. 2025. Bang for the Buck: Vector Search on Cloud CPUs. In *DaMoN*. 4:1–4:8.

[23] Maximilian Kuschewski, David Sauerwein, Adnan Alhomssi, and Viktor Leis. 2023. BtrBlocks: Efficient Columnar Compression for Data Lakes. *Proc. ACM Manag. Data* 1, 2 (2023), 118:1–118:26.

[24] Viktor Leis, Michael Haubenschild, Alfons Kemper, and Thomas Neumann. 2018. LeanStore: In-Memory Data Management beyond Main Memory. In *ICDE*. 185–196.

[25] Viktor Leis and Maximilian Kuschewski. 2021. Towards Cost-Optimal Query Processing in the Cloud. *Proc. VLDB Endow.* 14, 9 (2021), 1606–1612.

[26] Alberto Lerner, Carsten Binnig, Philippe Cudré-Mauroux, Rana Hussein, Matthias Jasny, Theo Jepsen, Dan R. K. Ports, Lasse Thostrup, and Tobias Ziegler. 2023. Databases on Modern Networks: A Decade of Research that now comes into Practice. *Proc. VLDB Endow.* 16, 12 (2023), 3894–3897.

[27] Adrian Lutsch, Christian Franck, Muhammad El-Hindi, Zsolt István, and Carsten Binnig. 2025. An Analysis of AWS Nitro Enclaves for Database Workloads. In *DaMoN*. ACM, 5:1–5:8.

[28] Ashraf Mahgoub, Alexander Medoff, Rakesh Kumar, Subrata Mitra, Ana Klimovic, Somali Chaterji, and Saurabh Bagchi. 2020. OPTIMUSCLOUD: Heterogeneous Configuration Optimization for Distributed Databases in the Cloud. In *USENIX ATC*. 189–203.

[29] Thomas Neumann and Michael J. Freitag. 2020. Umbra: A Disk-Based System with In-Memory Performance. In *CIDR*.

[30] Leah Shalev, Hani Ayoub, Nafea Bshara, Yuval Fatael, Ori Golan, Omer Ilany, Anna Levin, Zorik Machulsky, Kevin Milczewski, Marc Olson, Valentin Priescu, Shyam Rajagopal, and Ali Saidi. 2024. The Tail at Amazon Web Services Scale. *IEEE Micro* 44, 5 (2024), 23–29.

[31] George William Stagg, Henry Lionel, and Others. 2023. *webR: The statistical language R compiled to WebAssembly via Emscripten.* https://github.com/r-wasm/webr

[32] Ion Stoica and Scott Shenker. 2021. From cloud computing to sky computing. In *HotOS*. 26–32.

[33] Murray Stokely, Neel Nadgir, Jack Peele, and Orestis Kostakis. 2025. Shaved Ice: Optimal Compute Resource Commitments for Dynamic Multi-Cloud Workloads. In *ICPE*. 124–135.

[34] Junjay Tan, Thanaa M. Ghanem, Matthew Perron, Xiangyao Yu, Michael Stonebraker, David J. DeWitt, Marco Serafini, Ashraf Aboulnaga, and Tim Kraska. 2019. Choosing A Cloud DBMS: Architectures and Tradeoffs. *Proc. VLDB Endow.* 12, 12 (2019), 2170–2182.

[35] Neil C. Thompson and Svenja Spanuth. 2021. The decline of computers as a general purpose technology. *Commun. ACM* 64, 3 (2021), 64–72.

[36] Alexandre Verbitski, Anurag Gupta, Debanjan Saha, Murali Brahmadesam, Kamal Gupta, Raman Mittal, Sailesh Krishnamurthy, Sandor Maurice, Tengiz Kharatishvili, and Xiaofeng Bao. 2017. Amazon Aurora: Design Considerations for High Throughput Cloud-Native Relational Databases. In *SIGMOD Conference*. ACM, 1041–1052.

[37] Ziheng Wang, Emanuel Adamiak, and Alex Aiken. 2024. A Model for Query Execution Over Heterogeneous Instances. In *CIDR*.

[38] Bowen Wu, Wei Cui, Carlo Curino, Matteo Interlandi, and Rathijit Sen. 2025. Terabyte-Scale Analytics in the Blink of an Eye. *CoRR* abs/2506.09226 (2025).

[39] Xinjing Zhou, Viktor Leis, Jinming Hu, Xiangyao Yu, and Michael Stonebraker. 2025. Practical DB-OS Co-Design with Privileged Kernel Bypass. *Proc. ACM Manag. Data* 3, 1 (2025), 64:1–64:27.

[40] Xinjing Zhou, Viktor Leis, Xiangyao Yu, and Michael Stonebraker. 2026. Tux: Efficient Drop-in Networking for Database Systems. In *VLDB*.

[41] Tobias Ziegler, Dwarakanandan Bindiganavile Mohan, Viktor Leis, and Carsten Binnig. 2022. EFA: A Viable Alternative to RDMA over InfiniBand for DBMSs?. In *DaMoN*. ACM, 10:1–10:5.