EnergAt: Fine-Grained Energy Attribution for Multi-Tenancy

Hongyu Hè*
ETH Zurich
Switzerland

Michal Friedman
ETH Zurich
Switzerland

Theodoros Rekatsinas
Apple Inc.
Switzerland

ABSTRACT

In the post-Moore's Law era, relying solely on hardware advancements for automatic performance gains is no longer feasible without increased energy consumption, due to the end of Dennard scaling. Consequently, computing accounts for an increasing amount of global energy usage, contradicting the objective of sustainable computing. The lack of hardware support and the absence of a standardized, software-centric method for the precise tracing of energy provenance exacerbates the issue. Aiming to overcome this challenge, we argue that fine-grained software energy attribution is attainable, even with limited hardware support. To support our position, we present a thread-level, NUMA-aware energy attribution method for CPU and DRAM in multi-tenant environments. The evaluation of our prototype implementation, EnergAt, demonstrates the validity, effectiveness, and robustness of our theoretical model, even in the presence of the noisy-neighbor effect. We envisage a sustainable cloud environment and emphasize the importance of collective efforts to improve software energy efficiency.

CCS CONCEPTS

• **Hardware → Power estimation and optimization**; • **Software and its engineering → Software organization and properties**; • **Computer systems organization**;

KEYWORDS

software energy attribution, sustainable computing, energy efficiency, multi-tenancy, non-uniform memory access

ACM Reference Format:
Hongyu Hè, Michal Friedman, and Theodoros Rekatsinas. 2023. EnergAt: Fine-Grained Energy Attribution for Multi-Tenancy. In *2nd Workshop on Sustainable Computer Systems (HotCarbon '23), July 9, 2023, Boston, MA, USA.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3604930.3605716

1 INTRODUCTION

In the post-Moore era, accelerating application by hardware advancements typically requires computing systems to burn more energy. This leads to increasing amounts of global energy consumption [23, 27, 38, 51], which impairs the sustainability of computing operations. Unlike hardware, software design and optimization often neglect their impact on energy efficiency and carbon footprints. This phenomenon is primarily due to challenges in *software energy attribution*. Energy attribution of software aims to determine the share of energy consumed by the target application and its subtasks (*energy provenance*), excluding the fraction used by other collocated applications. Task-level energy attribution in multi-tenant environments could not only facilitate energy-aware decision-making in the cloud but also help developers gain first-hand insights into the energy efficiency and carbon footprint of their applications.

However, such fine-grained attribution is particularly hard due to the lack of support from both hardware and software. First, energy-related statistics measured by hardware are typically *coarse-grained* (at the device/socket-level) and does not support runtime or software-level information (fine-grained) for energy attribution. For instance, the energy consumed by a program running on a CPU core for a period of time cannot be directly obtained from hardware, since CPU power is measured at the socket level [48]. Second, the problem of software attribution is exacerbated by the increasing heterogeneity and multi-tenancy in the cloud. For example, together with CPUs, GPUs, and storage, specialized accelerators such as TPUs and FPGAs are increasingly shared among many tenants in the cloud [10, 26, 28, 31, 37, 41, 46]. Moreover, the push for compute-storage disaggregation [5, 7] complicates the problem of accurately attributing energy at the application/subtask level further. Therefore, hardware support alone cannot effectively solve the problem of fine-grained energy attribution, and software solutions are needed.

Gap. In recent years, several tools have been developed to measure software energy consumption (e.g., [6, 12, 21, 25, 34, 52]). These tools have primarily focused on usability, accessibility, and the interpretability of their outputs, but they do not aim for fine-grained energy attribution. Specifically, they either assume that the measured application is not collocated with other tasks, treating the total energy consumption of the host machine as that of the target application, or they use coarse-grained energy attribution models at the process level [11, 52] and do not consider NUMA effects in case of multiple CPU sockets (§2). Moreover, these tools do not update the traced subtasks of the target application reactively, which is problematic since processes and threads can be created and deleted at runtime. Furthermore, these tools do not separate their own energy cost from the measurements of target. We find that the lack of such accounting in fine-grained, software-centric attribution methodology [4] leads to more than 46.3% overestimation and 93.3% underestimation (Fig. 1), which could be detrimental for sustainable runtime operations.

Our position. We argue that developing fine-grained energy attribution models is feasible even with coarse-grained hardware support. These models could serve as the foundation for building sustainable cloud environments. In order to achieve this objective, precise and validated accounting of software energy provenance is

*Corresponding author <hongyu.he@inf.ethz.ch>.

HotCarbon '23, July 9, 2023, Boston, MA, USA

ACM ISBN 979-8-4007-0242-6/23/07…$15.00
https://doi.org/10.1145/3604930.3605716

required in multi-tenant environments. To support our stance with concrete exposition, we make the following contributions:

(1) We propose a thread-level, NUMA-aware method for CPU and DRAM energy attribution in a multi-tenant environment (§3).
(2) To evaluate our theoretical model, we provide a prototype implementation (§4) and present preliminary results (§5) that demonstrate the attribution model's validity, effectiveness, and robustness to the noisy-neighbor effect in multi-tenant environments.
(3) Building upon the fine-grained tracing of software energy provenance, we envisage a cloud environment in which decentralized, software-centric energy attribution supports a *logically* centralized runtime control to conduct energy-aware operations and provide feedback to users about their applications' energy efficiency (§7). We also present insights into promising opportunities and prominent challenges thereof.

2 COARSE-GRAINED ENERGY ATTRIBUTION

Coarse hardware support. Commodity hardware mostly supports coarse-grained power estimation. For instance, as of the Sandy Bridge generation, Intel processors come with a built-in power meter, the Running Average Power Limit (RAPL) [48]. It provides an interface to the accumulated energy consumption of various components, e.g., CPU package and DRAM. Although its energy reporting is mainly based on kernel events [48], RAPL has been proven to be sufficiently accurate and can capture millisecond-level energy events [22, 29, 48]. GPUs also support power reporting, e.g., NVIDIA's NVML [40] and ROCm SMI [2] from AMD. For FPGAs, Xilinx-AMD provides dedicated tools for power estimation [3]. Unfortunately, none of these devices inherently supports fine-grained energy measurement and attribution. For example, energy measurements from RAPL are reported at the socket level. Similarly, although fractional sharing of GPU [41] and FPGA [37] is now possible, their energy reporting is still at the device/user level.

Coarse-grained software energy accounting. Several recent tools have been developed to measure software energy consumption. Many of them focus on machine learning (ML) workloads, which are particularly energy-intensive [25, 45, 54]. To name a few, CodeCarbon [34] reports the energy usage of a program, measuring the consumption of CPU, DRAM, and GPU. It features a user-friendly API, a UI and exports interpretable results. Carbontracker [6] offers similar measurements and predicts the energy consumption of ML training based on a few epochs. Specifically built for ML applications, Experiment Impact Tracker (Impact Tracker) [25] collects energy measurements for both CPU and GPU, and allows users to generate environmental impact statements for their experiments. Another popular tool is Scaphandre [21] which has integrated support for power measurements in Kubernetes.

Unfortunately, existing tools only employ coarse-grained energy attribution models as summarized below. Consider a target application $\mathcal{A}$ that potentially consists of multiple tasks a running on a set of devices D. Then, *coarse-grained* energy attribution models can be formulated as:

$$E^D_{\text{total}} \leftarrow \text{Sample power meter of } D \text{ every } T_{\text{sample}} \tag{1}$$

$$E^D_{\mathcal{A}} = E^D_{\text{total}} \cdot \left[\left(\sum\nolimits_{a \in \mathcal{A}} U^D_a \right) / U^D_{\text{total}} \right], \tag{2}$$

where E^D_{total} and $E^D_{\mathcal{A}}$ are the total energy consumed by the server machine and application $\mathcal{A}$ on D, respectively. Every T_{sample} time, the corresponding energy meter of the device is sampled to obtain the accumulated energy for this period. U^D_a and U^D_{total} are the resource usage of the application task a and that of all tasks present on the server machine. We argue that this energy attribution method is insufficient in precisely capturing applications' power dynamics. Specifically, it neither takes into account NUMA effects [8, 14, 35, 55] in the presence of multiple sockets nor does it distinguish threads from processes when tracing energy provenance (§3.1). This method is also prone to the noisy-neighbor effect [15, 16] in a multi-tenant environment, where the energy attribution of an application is interfered by collocated tasks on the same host.

3 THREAD-LEVEL AND NUMA-AWARE ENERGY ATTRIBUTION

Although present-day hardware only provides coarse-grained energy measurement capabilities, we believe that there is still immense potential to achieve fine-grained energy attribution with a software-based approach [4]. In this work, we demonstrate fine-grained energy attribution of CPU and DRAM with coarse-grained measurements from Intel RAPL meter. CPU and DRAM are primary consumers of software energy. Even for GPU-dependent ML applications, they together still account for more than 30% of total energy use [6, 54].

3.1 Relevant Factors in Energy Attribution

Multiple sockets. Server-class machines generally have ≥ 2 CPU sockets, e.g., for higher memory capacity and fault tolerance. Unfortunately, when measuring and attributing energy, prior work (§2) does not take into account NUMA effects, and the application resources are aggregated over all sockets (Eq. 2). This approach can be problematic for accurate attribution due to the potential imbalance of resource allocation in NUMA architectures [8]. For example, a dual-socket machine has only one user task running, whose CPU times are 30 s and 180 s on each of the two sockets. If the *total* CPU times and the measured energy consumption of the two sockets are (100 s, 30 J) and (200 s, 50 J) respectively, then the CPU energy consumption attributed to the task should be $(30/100 \times 30 + 180/200 \times 50)$ J, instead of $[(30+180)/(100+200) \times (30+50)]$ J. Note that, apart from CPU time, another crucial factor making a difference here is CPU utilization. Specifically, utilization typically varies among sockets at a certain point in time, and power scales non-linearly with it [19, 49, 50]. Consequently, the same amount of CPU time would result in different power dynamics at different utilization levels. Therefore, relying solely on CPU time as an aggregated proxy is insufficient for estimating energy consumption across multiple sockets. The same principle also applies when accounting for memory's energy consumption. In practice, however, energy variability caused by NUMA memory allocation plays a less significant role, compared to CPU.

Threads vs. processes. High parallelism is prevalent in modern applications. Apart from traditional high-performance computing workloads, large numbers of parallel tasks in ML pipelines (e.g., feature stores [20, 42, 53]) can amount to $1/3^{rd}$ of the total energy

consumption, exceeding the amount of energy used by model training of large-scale jobs [54]. To precisely attribute CPU energy for a parallel application, one needs to obtain the CPU time for each of its tasks (processes *and* kernel threads) per socket. For example, a parallel application spawns several threads and processes, each of which could have different runtime statistics on different sockets, depending on the placement decisions made by the scheduler.

Furthermore, when querying resource statistics, tools such as `ps` and `top` either (1) use the total resource usage of the process group (PG)[1] as that of a single task therein (process/thread), or (2) separate the statistics for each task, given different flags. Unfortunately, existing libraries do not always handle the two cases properly. For example, many energy tools rely on the library `psutil` [47], which reports total resource usage of the PG when asked for that of a thread (case (2)). In turn, when tracing energy provenance at the process level, case (1) can result in imprecise measurements (since the resource usages of threads and processes cannot be distinguished for individual accounting), and case (2) can lead to underestimation (as only the resource usage of processes is accounted and that of threads is ignored). Thus, the fact that a in Eq. 2 purely represents processes and ingores threads is problematic for fine-grained attribution. Note that, for resources shared between the parent process and its threads (e.g., stack memory), making such a distinction is unnecessary. However, energy tracking tools themselves should *not* be created as threads of the application it measures. Otherwise, the resources used by the energy tool would entangle with its target application, which is also a pitfall in existing methods.

Noisy-neighbor effects. Nowadays, applications are typically deployed in the cloud, where they are colocated with other tasks, sharing resources on the same host. Multi-tenancy creates "noisy-neighbor" effects [15, 16], by which the performance of an application is interfered by its "neighbors". In the presence of such interference, only the energy consumed by the target application should be accounted.

3.2 Fine-Grained Attribution Model

Taking into account the aforementioned factors (§3.1), we propose a thread-level energy attribution model that is NUMA-aware and robust to the noisy-neighbor effect.

Static power. The first step in our model is to measure the static power of the host on which the target application runs. The static power is assumed to be independent of the load and should not be confused with the idle power, which is consumed by the server machine in various sleep states [1, 23, 24]. This value can either be obtained from the manufacturer's datasheet or more practically, via a sampling procedure. For each socket $s \in S$ and a sampling period T_{static}, the average static power $(P_{\text{static}})^s$ is given by:

$$\left(P^D_{\text{static}}\right)^s = (\text{Sample energy value of } D \text{ for } T_{\text{static}}) \,/ T_{\text{static}}. \tag{3}$$

Then, at runtime, the total energy used by the host $\left(E^D_{\text{total}}\right)^s$ can be obtained using Eq. 1 for each s, and the static energy consumption $\left(E^D_{\text{static}}\right)^s$ can be obtained periodically:

$$\left(E^D_{\text{static}}\right)^s = \left(P^D_{\text{static}}\right)^s \cdot T_{\text{sample}}. \tag{4}$$

Attributing CPU energy with thread-level metrics. To attribute CPU energy, we first obtain the power offset $\left(E^{\text{CPU}}_{\Delta}\right)^s$ by subtracting the static power from the total:

$$\left(E^{\text{CPU}}_{\Delta}\right)^s = \left(E^{\text{CPU}}_{\text{total}}\right)^s - \left(E^{\text{CPU}}_{\text{static}}\right)^s. \tag{5}$$

Having obtained the host-wide energy statistics, we now quantify the resource usage of $\mathcal{A}$ in *thread* granularity. Specifically, we estimate the *CPU residence rate* for every process *and* thread $a \in \mathcal{A}$ on each socket s, i.e., the fraction of time task a was scheduled on s:

$$\mathbb{P}^{\text{CPU}}(s \mid a) \approx \left(\int_{t=t'}^{t'+T_{\text{sample}}} \mathbb{1}_{\{a \text{ on } s\}} dt \right) \Big/ T_{\text{sample}}, \tag{6}$$

where dt in practice is the discretized time steps for sampling kernel scheduling decisions. With Eq. 6, we approximate the CPU time of $\mathcal{A}$ on s with an expectation conditioned on the kernel scheduling decisions:

$$\left(T^{\text{CPU}}_{\mathcal{A}}\right)^s = \mathbb{E}\left[T^{\text{CPU}}_{\mathcal{A}} \mid s\right] \approx \sum_{a \in \mathcal{A}} \mathbb{P}^{\text{CPU}}(s \mid a) \cdot T^{\text{CPU}}_{a}, \tag{7}$$

where T^{CPU}_{a} is the total CPU time of a on all $s \in S$.

Furthermore, to combat the noisy-neighbor effect, we propose the concept of *energy credit* denoted $C^D_{\mathcal{A}}$, that is, how much a fraction of the energy consumption of D should be attributed to $\mathcal{A}$. Specifically, we employ the proportion of $\mathcal{A}$'s CPU time over that of all tasks on the server as a proxy for the CPU energy credit on s:

$$(T_{\text{total}})^s \leftarrow \text{Total CPU time (kernel + user) of } s \tag{8}$$

$$\left(C^{\text{CPU}}_{\mathcal{A}}\right)^s = \left[\left(T^{\text{CPU}}_{\mathcal{A}}\right)^s \Big/ \left(T^{\text{CPU}}_{\text{total}}\right)^s \right]^{\gamma}, \tag{9}$$

where $(T_{\text{total}})^s$ is the server-wide CPU time per socket and $0 \geq \gamma \leq 1$ is a scaling factor that takes into account machine-specific non-linearity, since the energy consumption of CPU does not scale linearly with the utilization [19, 49, 50]. Specifically, the trend flattens gradually as utilization gets higher. Using the CPU energy credit, we compute the energy consumption of $\mathcal{A}$ by aggregating values from all sockets:

$$E^{\text{CPU}}_{\mathcal{A}} = \sum_{s \in S} \left(E^{\text{CPU}}_{\Delta}\right)^s \cdot \left(C^{\text{CPU}}_{\mathcal{A}}\right)^s + \left(E^{\text{CPU}}_{\text{static}}\right)^s. \tag{10}$$

Attributing DRAM energy with NUMA memory statistics. The energy attribution for DRAM is similar, except that we no longer consider threads individually, since they share memory under the same PG. However, the imbalance in memory allocation of NUMA architectures still needs to be dealt with care. Firstly, we obtain the server-wide memory usage per socket $(M_{\text{total}})^s$ and the memory offset $\left(E^{\text{DRAM}}_{\Delta}\right)^s$:

$$(M_{\text{total}})^s \leftarrow \text{Total available NUMA memory on } s \tag{11}$$

$$\left(E^{\text{DRAM}}_{\Delta}\right)^s = \left(E^{\text{DRAM}}_{\text{total}}\right)^s - \left(E^{\text{DRAM}}_{\text{static}}\right)^s. \tag{12}$$

[1] Resource group created by the parent process.

Next, we measure *memory residence rate* for each *process* $a \in \mathcal{A}$, i.e., the fraction of *private* NUMA memories allocated on s, excluding shared memories (whose ownership is hard to reason about):

$$\mathbb{P}^{\text{DRAM}}(s \mid a) \approx \mathbb{E}\left[\left\{\left(M_a^{\Delta t}\right)^s \Big/ \left(M_{\text{total}}^{\Delta t}\right)^s\right\}^{T_{\text{sample}}}\right], \tag{13}$$

where {} encloses a collection of memory samples on s over a sampling period T_{sample} with discretized steps Δt. Then, the total NUMA memory of $\mathcal{A}$ on s can be approximated by:

$$(M_{\mathcal{A}})^s = \mathbb{E}\left[M_{\mathcal{A}} \mid s\right] \approx \sum_{a \in \mathcal{A}} \mathbb{P}^{\text{DRAM}}(s \mid a) \cdot (M_a)^s. \tag{14}$$

Now, we represent the *memory energy credit* of $\mathcal{A}$ on s as the fraction of private memory of $\mathcal{A}$:

$$\left(C_{\mathcal{A}}^{\text{DRAM}}\right)^s = \left[(M_{\mathcal{A}})^s \,/\, (M_{\text{total}})^s\right]^{\sigma}, \tag{15}$$

where, similar to γ in Eq. 9, σ is the machine-specific scaling factor. With the formulated memory credit in Eq. 15, the DRAM energy attribution of $\mathcal{A}$ can be computed by:

$$E_{\mathcal{A}}^{\text{DRAM}} = \sum_{s \in S} \left(E_{\Delta}^{\text{DRAM}}\right)^s \cdot \left(C_{\mathcal{A}}^{\text{DRAM}}\right)^s + \left(E_{\text{static}}^{\text{DRAM}}\right)^s. \tag{16}$$

4 ENERGAT: PROTOTYPE IMPLEMENTATION

To evaluate our theoretical model, we develop and open source[2] a prototype implementation of our attribution model, named EnergAt.

Firstly, to cleanly distinguish the energy consumption of the tool and the user program (§2), we implement EnergAt as a *separate process* of the target application. Once it obtains the static power information, its main process creates a daemon *thread* that samples the RAPL meter and relevant thread-level NUMA events every T_{sample} time (Eq. 1). Table 1 lists the sampled counters and their corresponding metrics used by the attribution model. Apart from the maximum domain values and the clock speed that are obtained once at the beginning, all other counters are sampled at this frequency. This sampling period currently is set to 10 ms and can be adjusted based on the type of application it targets. Note that, even the smallest sampling interval supported by RAPL (1 ms) is larger than the minimum scheduling granularity of the Linux kernel. Nevertheless, EnergAt only aims for an approximation of the conditional probabilities (Eq. 6 and 13) with low measurement overheads. Given our evaluation results (§5), the 10 ms interval appears empirically sufficient for precise energy attribution.

Based on the statistics collected by the daemon thread, the parent process of EnergAt computes the thread-level resource usages $\left(T_{\mathcal{A}}^{\text{CPU}}\right)^s$ and $(M_{\mathcal{A}})^s$ aggregated by sockets (Eq. 7 and 14) . It then calculates the CPU and memory energy credits by Eq. 9 and 15. Finally, EnergAt attributes energy based on the energy credits (Eq. 10 and 16) and stores energy traces in its database, which could be queried later.

5 ATTRIBUTION MODEL EVALUATION

This section presents preliminary evaluation results of our energy attribution model (§3) implemented in EnergAt. We employ the Linux benchmarking tool `stress-ng` [18, 30] to create four types of workloads (Table 2). The aim of these microbenchmarks is to (1) cover different utilization levels of the two devices and (2) emulate the noisy-neighbor effect. The testbed we use in the following experiments is a dual-socket server, where each NUMA node has 8 Intel Xeon E5-2630 CPUs (16 logical cores) and 32 GiB of DRAM. We run each workload for one hour, averaging results over five runs.

Counters	Metrics
Intel RAPL package and DRAM domains	Accumulated energy consumption of CPU packages and DRAM (through the `sysfs` interface)
Intel RAPL maximum counter values	Maximum ranges of each domain for detecting and mitigating counter overflow
Memory statistics from the `numactl` package	Total, used, and private memory statistics for processes and the operating system on a per-NUMA-node basis
`/proc/*/task/*/stat`	User and kernel times for each task and its children
`CLK_TCK` value	Number of clock ticks per second

Table 1: Descriptions of metrics from sampled counters.

Workload	Description
`cpu`	Sweeps CPU utilization from 0 to 100% with equal numbers of processes and threads loaded with matrix operations
`mem`	Sweeps memory usage from 0 to 100% with one process continuously calling `mmap`/`unmap`
`mix`	Keeps both the CPU and memory utilization at 50% using the same methods as that of `cpu` and `mem`
`mix (w/ neighbor)`	Launches two `mix` workloads, treating one as the target application and another as the "noisy neighbor" collocated on the same host

Table 2: Descriptions of employed microbenchmarks.

Model validation. We start by validating the total energy consumed by the host server measured by EnergAt. To this end, we make a popular Firefox plugin [39] run independently for reference. As shown in Fig. 1, the total energy consumption measured by EnergAt closely matches that of the Firefox plugin.

Next, we validate our fine-grained energy attribution model. Since direct validation is infeasible [52], we conduct validation by summation [52]. In Fig. 2, we trace the energy provenance of *all* jobs present on the host, including the corresponding microbenchmarks, shown in light gray, and sum their attributed energies together (light gray bars) to compare with the total energy value of the machine (white and dark gray bars). In other words, the attribution model is indirectly validated if the energy attribution of all the jobs, plus the energy used by EnergAt (black bars), amounts to the same value as that of the total energy consumed by the host server.

This validation is non-trivial due to two main factors. Firstly, the power model is non-linear in nature. Secondly, each entity (thread/process) is traced independently. In other words, the attributed energies cannot simply be summed up to match the total energy consumption if the model fails to accurately assign energy to each entity individually. The resulting summation of attributed energies from all collocated tasks closely matches the total values on all three workloads (Table 2) with an average error margin of 4.52%. We anticipated a bit higher error for the `mem` workload, as EnergAt currently only considers private memories in attribution and disregards any shared memories (Eq. 13), which we defer to future work.

Robustness to noisy-neighbor effect. Now, we evaluate the robustness and the effectiveness of the energy crediting method (Eq. 9 and 15) in the presence of the noisy-neighbor effect. To this

[2] https://github.com/HongyuHe/energat

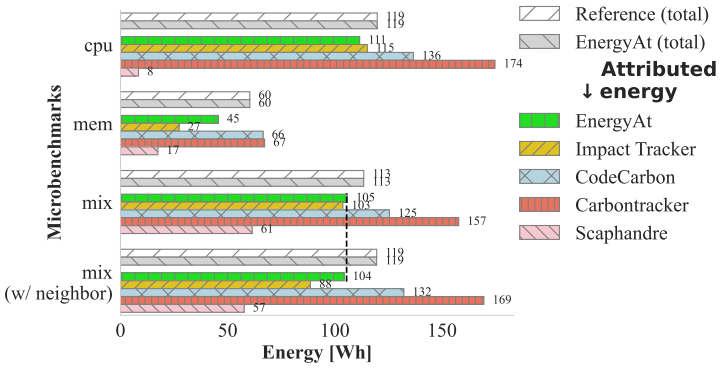

Figure 1: Drastically different results from various tools on four microbenchmarks (Table 2). The bars below the `(total)` values are the attributed energies to the corresponding benchmark tasks by different tools. We highlight three observations: (1) existing methods can exhibit more than 46.3% overestimation and 93.3% underestimation of the *attributed* energy compared to the *total* values, (2) the total energy consumption measured by our fine-grained energy attribution model EnergAt matches that of the reference value, and (3) the dotted line illustrates that EnergAt is robust to the noisy-neighbor effect.

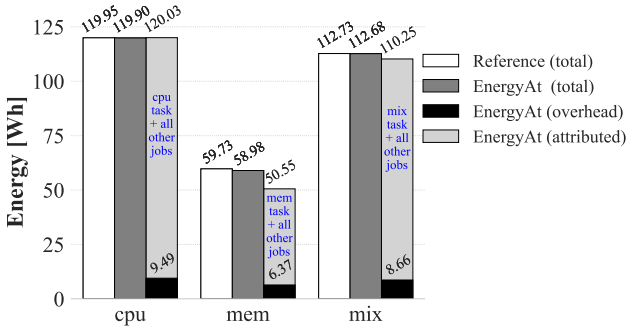

Figure 2: Validation by summation for our fine-grained energy attribution model. The model is indirectly validated if the light gray part plus the black portion is equal to the total value (white or dark gray bars).

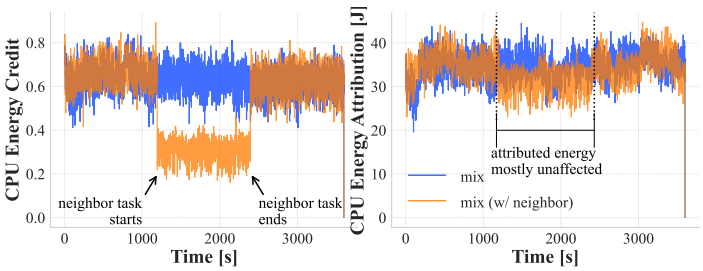

Figure 3: Change of energy credit ($C_{\mathcal{A}}^{\text{CPU}}$, Eq. 9) assigned to the target application (left), reacting to the launch of a "neighbor task" running the same workload as the target application on the same host, while the actual amount of energy attributed to the target remains mostly unaffected (right).

end, we employ the `mix (w/ neighbor)` microbenchmark (Table 2). Although energy consumption will be slightly different between runs even on the same machine with the same workload, we expect that the total energy attributed to the same workload should be approximately the same, regardless of whether it runs in an isolated environment or co-locate with a neighboring task. When the neighbor task starts, the total energy consumption of the server increases due to higher resource usage, and the energy credit assigned to the target also drops, since the relative fraction of resources used by the target application reduces (Fig. 3 left). This reactive change in energy credit assignment assures that the energy attributed to the target by EnergAt remains almost unaffected (Fig. 3 right and Fig. 1), while the measurements from other attribution tools change substantially due to the noisy-neighbor effect.

Low attribution overhead. We aim for not only a precise energy attribution method but also a low-overhead one for practical usage. Thus, unlike existing methods, we explicitly separate the energy used by EnergAt from the traced application and its subtasks (§4). When tracing the energy provenance of a single application (Fig. 1), the energy overhead of EnergAt is 6.5% on average. The energy cost is 8.9% on average when tracing all jobs on the server (Fig. 2).

6 LIMITATIONS

Although our proposed method demonstrates promising results, several limitations are crucial to be addressed in future work. First, the power modeling (Eg. 10 and 16) does not take into account other relavent factors, such as various I/O and caches [52]. Also, it only considers private memories, which results in lower energy attribution on the `mem` microbenchmark (Fig. 2). This shortcoming is in part due to the restricted hardware interface and the overhead of sampling corresponding counters at a fine-grained level. For instance, unlike well-integrated RAPL interfaces, accounting disk and network I/O requires external power meters. Similarly, obtaining *both* per-thread and per-socket cache events is non-trivial. Additionally, setting machine-specific model parameters (γ, δ) currently needs hand-tuning for a certain platform. Second, the energy cost of the prototype is non-negligible, which can be partially ascribed to the inefficiency of reading various counters.

Moreover, although EnergAt automatically pins its process and daemon thread to the least-loaded core, it could still impose a higher performance penalty, compared to the coarse-grained tools. While being an inherent tradeoff between granularity and cost, this drawback could be mitigated through a more optimized implementation of the proposed attribution model. For example, the importance of various counter values differs by specific use cases [11], and in turn, they should be sampled at different granularity to reduce the overhead. Lastly, validating fine-grained energy attribution model remains to be a prominent challenge. Validation by summation [52] (§5) is rather limited in that it cannot provide insight into the energy attribution of each traced entity. Since fine-grained validation is virtually impossible in a real system, full-system simulation (e.g., gem5 [36]) could be of help for this purpose.

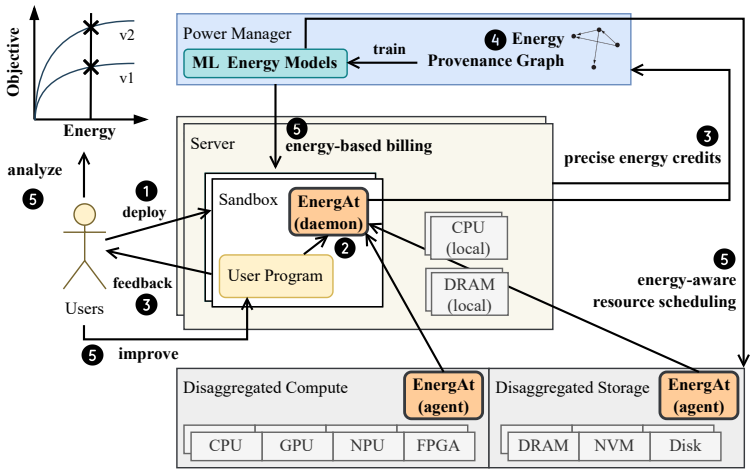

Figure 4: Depiction of a sustainable cloud environment where both developers and providers make informed decisions based on *distributed, fine-grained* energy attribution.

7 CHALLENGES AND OPPORTUNITIES

We lay out a vision of sustainable cloud environments (Fig. 4) where EnergAt can help both the users and cloud providers incorporate energy into their operation and development cycles.

In a sustainable cloud environment, service providers can inject EnergAt into user sandboxes (e.g., as a side-car container), upon application deployment (❶). The tool runs alongside the user program as a daemon, monitoring the power dynamic by collecting performance counters and energy readings from the local machine and from remote agents located on disaggregated compute and/or storage nodes (❷). Then, EnergAt attributes energy to the user application and reports corresponding energy credits to the Power Manager of the cluster, informing the application owner of the energy consumption (❸). The precise energy credits can be used to construct provenance graphs [33, 43, 44] (❹) for tracing the power relationships among deployed applications. Such graph representations can be leveraged to train high-quality ML models that facilitate power management. Furthermore, users can analyze and improve their software energy efficiency based on the detailed feedback (❺). Similarly, cloud providers can make energy-aware decisions accordingly (❺), e.g., energy-based billing and workload migration for mitigating hot spots. That said, there are as many promising opportunities as there are challenges in this virtuous cycle.

New interfaces for secure and efficient energy reporting. The lack of secure and efficient interfaces between hardware and software severely inhibits energy measurement (❷). For instance, reading RAPL requires manual timestamp alignment [22] and privileged permission for security [13]. Moreover, virtualization is a similar challenge as energy-related counters are generally not propagated to the virtualized applications. Consequently, most of the energy models for virtualized environments are predictive and treat user programs as black boxes [9, 17, 18, 32]. Thus, tracing fine-grained energy provenance amid layers of virtualization is way more complex and remains an open question.

Energy attribution for new cloud computing services. Multi-tenancy and heterogeneity in the cloud are being taken to the extreme in order to remain profitable in the post-Moore's Law era. For example, various hardware accelerators are increasingly shared among large numbers of tenants. They have fundamentally different inner workings compared to Von Neumann architectures. Furthermore, high-level cloud services (e.g., DBaaS, MLaaS, and FaaS) have emerged and are popularized. Their abstractions are farther away from hardware and highly distributed in nature. These factors not only pose challenges to collecting energy statistics (❷) but also to tracing precise energy provenance (❸, ❹).

NUMA-aware energy optimization. Although this work shows the importance of carefully attributing energy for applications running on multiple sockets, how developers and cloud providers can leverage the proposed the model (❺) remains unexplored. For instance, is the energy consumption of an application the same whether it runs on a different core of the same socket or on a different socket within the same server? The answer to this question would be useful for both users and cloud providers in terms of improving energy efficiency. This question, however, is also challenging as there are temperature effects that impact the actual power usage.

Improving software performance with energy in mind. With the availability of new abstractions and tools, developers can gain insights into the energy dynamics of their applications (❸). In turn, development decisions should not only focus on traditional performance optimization but also consider energy efficiency (❺). For instance, a 10% increase in throughput may not be considered beneficial if it comes at the cost of a 50% higher power consumption. Similarly, in the context of training ML models, energy should be taken into account as an early-stopping criterion, since a mere 0.1% loss reduction may not justify the addition of 100 kWh of energy consumption. Last but not least, it is worth revisiting the energy efficiency of traditional algorithms and data structures whose optimization has primarily focused on performance.

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their many helpful comments. We especially would like to thank Shail Dave for his invaluable assistance throughout the revision process. Shail's extensive feedback on each section and meticulous attention to detail, including the revision of every figure, played a vital role in enhancing the quality of the final paper.

REFERENCES

[1] 2017. package c-states | 12th generation intel® coreTM processors datasheet. https://edc.intel.com/content/www/us/en/design/ipla/software-development-platforms/client/platforms/alder-lake-desktop/12th-generation-intel-core-processors-datasheet-volume-1-of-2/001/package-c-states/
[2] AMD. 2023. ROCm System Management Interface Support Guide v5.3. https://docs.amd.com/bundle/ROCm-System-Management-Interface-Support-Guide-v5.3
[3] Xilinx AMD. 2023. Xilinx Power Estimator (XPE). https://www.xilinx.com/products/technology/power/xpe.html.
[4] Thomas Anderson, Adam Belay, Mosharaf Chowdhury, Asaf Cidon, and Irene Zhang. 2022. Treehouse: A case for carbon-aware datacenter software. *arXiv preprint arXiv:2201.02120* (2022).

[5] Sebastian Angel, Mihir Nanavati, and Siddhartha Sen. 2020. Disaggregation and the application. In *Proceedings of the 12th USENIX Conference on Hot Topics in Cloud Computing*. 15–15.

[6] Lasse F. Wolff Anthony, Benjamin Kanding, and Raghavendra Selvan. 2020. Carbontracker: Tracking and Predicting the Carbon Footprint of Training Deep Learning Models. *CoRR* abs/2007.03051 (2020). arXiv:2007.03051 https://arxiv.org/abs/2007.03051

[7] Laurent Bindschaedler, Ashvin Goel, and Willy Zwaenepoel. 2020. Hailstorm: Disaggregated compute and storage for distributed lsm-based databases. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*. 301–316.

[8] Sergey Blagodurov, Sergey Zhuravlev, Mohammad Dashti, and Alexandra Fedorova. 2011. A Case for NUMA-aware Contention Management on Multicore Systems. In *2011 USENIX Annual Technical Conference, Portland, OR, USA, June 15-17, 2011*, Jason Nieh and Carl A. Waldspurger (Eds.). USENIX Association. https://www.usenix.org/conference/usenixatc11/case-numa-aware-contention-management-multicore-systems

[9] Ata E Husain Bohra and Vipin Chaudhary. 2010. VMeter: Power modelling for virtualized clouds. In *2010 ieee international symposium on parallel & distributed processing, workshops and phd forum (ipdpsw)*. Ieee, 1–8.

[10] Adrian M. Caulfield, Eric S. Chung, Andrew Putnam, Hari Angepat, Jeremy Fowers, Michael Haselman, Stephen Heil, Matt Humphrey, Puneet Kaur, Joo-Young Kim, Daniel Lo, Todd Massengill, Kalin Ovtcharov, Michael Papamichael, Lisa Woods, Sitaram Lanka, Derek Chiou, and Doug Burger. 2016. A cloud-scale acceleration architecture. In *49th Annual IEEE/ACM International Symposium on Microarchitecture, MICRO 2016, Taipei, Taiwan, October 15-19, 2016*. IEEE Computer Society, 7:1–7:13. https://doi.org/10.1109/MICRO.2016.7783710

[11] Maxime Colmant, Mascha Kurpicz, Pascal Felber, Loïc Huertas, Romain Rouvoy, and Anita Sobe. 2015. Process-level power estimation in vm-based systems. In *Proceedings of the Tenth European Conference on Computer Systems*. 1–14.

[12] Stefano Corda, Bram Veenboer, and Emma Tolley. 2022. PMT: Power Measurement Toolkit. *CoRR* abs/2210.03724 (2022). https://doi.org/10.48550/arXiv.2210.03724 arXiv:2210.03724

[13] CVE. 2020. CVE-2020-8694. https://www.cve.org/CVERecord?id=CVE-2020-8694.

[14] Mohammad Dashti, Alexandra Fedorova, Justin Funston, Fabien Gaud, Renaud Lachaize, Baptiste Lepers, Vivien Quema, and Mark Roth. 2013. Traffic management: a holistic approach to memory placement on NUMA systems. *ACM SIGPLAN Notices* 48, 4 (2013), 381–394.

[15] Christina Delimitrou and Christos Kozyrakis. 2013. iBench: Quantifying interference for datacenter applications. In *Proceedings of the IEEE International Symposium on Workload Characterization, IISWC 2013, Portland, OR, USA, September 22-24, 2013*. IEEE Computer Society, 23–33. https://doi.org/10.1109/IISWC.2013.6704667

[16] Christina Delimitrou and Christos Kozyrakis. 2017. Bolt: I Know What You Did Last Summer... In The Cloud. In *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS 2017, Xi'an, China, April 8-12, 2017*, Yunji Chen, Olivier Temam, and John Carter (Eds.). ACM, 599–613. https://doi.org/10.1145/3037697.3037703

[17] Gaurav Dhiman, Kresimir Mihic, and Tajana Rosing. 2010. A system for online power prediction in virtualized environments using gaussian mixture models. In *Proceedings of the 47th Design Automation Conference*. 807–812.

[18] Guillaume Fieni, Romain Rouvoy, and Lionel Seinturier. 2020. Smartwatts: Self-calibrating software-defined power meter for containers. In *2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID)*. IEEE, 479–488.

[19] Alexander Gilgur, Brian Coutinho, Iyswarya Narayanan, and Parth Malani. 2021. Transitive Power Modeling for Improving Resource Efficiency in a Hyperscale Datacenter. In *Companion of The Web Conference 2021, Virtual Event / Ljubljana, Slovenia, April 19-23, 2021*, Jure Leskovec, Marko Grobelnik, Marc Najork, Jie Tang, and Leila Zia (Eds.). ACM / IW3C2, 182–191. https://doi.org/10.1145/3442442.3452057

[20] GitHub. [n. d.]. Ralf: A feature store for rapidly changing data. https://www.uber.com/en-CH/blog/michelangelo-machine-learning-platform/. https://github.com/feature-store/ralf

[21] Github. 2023. Scaphandre. https://github.com/hubblo-org/scaphandre.

[22] Marcus Hähnel, Björn Döbel, Marcus Völp, and Hermann Härtig. 2012. Measuring energy consumption for short code paths using RAPL. *SIGMETRICS Perform. Evaluation Rev.* 40, 3 (2012), 13–17. https://doi.org/10.1145/2425248.2425252

[23] Hongyu He. 2021. How Can Datacenters Join the Smart Grid to Address the Climate Crisis? Using simulation to explore power and cost effects of direct participation in the energy market. *CoRR* abs/2108.01776 (2021). arXiv:2108.01776 https://arxiv.org/abs/2108.01776

[24] Franz Christian Heinrich, Tom Cornebize, Augustin Degomme, Arnaud Legrand, Alexandra Carpen-Amarie, Sascha Hunold, Anne-Cécile Orgerie, and Martin Quinson. 2017. Predicting the Energy-Consumption of MPI Applications at Scale Using Only a Single Node. *2017 IEEE International Conference on Cluster Computing (CLUSTER)* (2017), 92–102.

[25] Peter Henderson, Jieru Hu, Joshua Romoff, Emma Brunskill, Dan Jurafsky, and Joelle Pineau. 2020. Towards the Systematic Reporting of the Energy and Carbon Footprints of Machine Learning. *CoRR* abs/2002.05651 (2020). arXiv:2002.05651 https://arxiv.org/abs/2002.05651

[26] John L Hennessy and David A Patterson. 2019. A new golden age for computer architecture. *Commun. ACM* 62, 2 (2019), 48–60.

[27] Nicola Jones et al. 2018. How to stop data centres from gobbling up the world's electricity. *Nature* 561, 7722 (2018), 163–166.

[28] Norman P Jouppi, George Kurian, Sheng Li, Peter Ma, Rahul Nagarajan, Lifeng Nai, Nishant Patil, Suvinay Subramanian, Andy Swing, Brian Towles, et al. 2023. Tpu v4: An optically reconfigurable supercomputer for machine learning with hardware support for embeddings. *arXiv preprint arXiv:2304.01433* (2023).

[29] Kashif Nizam Khan, Mikael Hirki, Tapio Niemi, Jukka K. Nurminen, and Zhonghong Ou. 2018. RAPL in Action: Experiences in Using RAPL for Power Measurements. *ACM Trans. Model. Perform. Evaluation Comput. Syst.* 3, 2 (2018), 9:1–9:26. https://doi.org/10.1145/3177754

[30] Colin Ian King. 2017. Stress-ng. *URL: http://kernel. ubuntu. com/git/cking/stressng. git/(visited on 28/03/2018)* (2017), 39.

[31] Dario Korolija, Timothy Roscoe, and Gustavo Alonso. 2020. Do OS abstractions make sense on FPGAs?. In *14th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2020, Virtual Event, November 4-6, 2020*. USENIX Association, 991–1010. https://www.usenix.org/conference/osdi20/presentation/roscoe

[32] Bhavani Krishnan, Hrishikesh Amur, Ada Gavrilovska, and Karsten Schwan. 2011. VM power metering: feasibility and challenges. *ACM SIGMETRICS Performance Evaluation Review* 38, 3 (2011), 56–60.

[33] Benno Kruit, Hongyu He, and Jacopo Urbani. 2020. Tab2know: Building a knowledge base from tables in scientific papers. In *The Semantic Web–ISWC 2020: 19th International Semantic Web Conference, Athens, Greece, November 2–6, 2020, Proceedings, Part I 19*. Springer, 349–365.

[34] Alexandre Lacoste, Alexandra Luccioni, Victor Schmidt, and Thomas Dandres. 2019. Quantifying the Carbon Emissions of Machine Learning. *CoRR* abs/1910.09700 (2019). arXiv:1910.09700 http://arxiv.org/abs/1910.09700

[35] Baptiste Lepers, Vivien Quéma, and Alexandra Fedorova. 2015. Thread and memory placement on {NUMA} systems: Asymmetry matters. In *2015 {USENIX} Annual Technical Conference ({USENIX}{ATC} 15)*. 277–289.

[36] Jason Lowe-Power, Abdul Mutaal Ahmad, Ayaz Akram, Mohammad Alian, Rico Amslinger, Matteo Andreozzi, Adrià Armejach, Nils Asmussen, Brad Beckmann, Srikant Bharadwaj, et al. 2020. The gem5 simulator: Version 20.0+. *arXiv preprint arXiv:2007.03152* (2020).

[37] Jiacheng Ma, Gefei Zuo, Kevin Loughlin, Xiaohe Cheng, Yanqiang Liu, Abel Mulugeta Eneyew, Zhengwei Qi, and Baris Kasikci. 2020. A Hypervisor for Shared-Memory FPGA Platforms. In *ASPLOS '20: Architectural Support for Programming Languages and Operating Systems, Lausanne, Switzerland, March 16-20, 2020*, James R. Larus, Luis Ceze, and Karin Strauss (Eds.). ACM, 827–844. https://doi.org/10.1145/3373376.3378482

[38] Eric Masanet, Arman Shehabi, Nuoa Lei, Sarah Smith, and Jonathan Koomey. 2020. Recalibrating global data center energy-use estimates. *Science* 367, 6481 (2020), 984–986.

[39] Mozilla. 2023. RAPL command-line utility in the Mozilla tree. https://firefox-source-docs.mozilla.org/performance/tools_power_rapl.html.

[40] NVIDIA. 2023. NVIDIA Management Library (NVML). https://developer.nvidia.com/nvidia-management-library-nvml

[41] NVIDIA. 2023. Unlock Next Level Performance with Virtual GPUs. https://www.nvidia.com/en-us/data-center/virtual-solutions/.

[42] Laurel J. Orr, Atindriyo Sanyal, Xiao Ling, Karan Goel, and Megan Leszczynski. 2021. Managing ML Pipelines: Feature Stores and the Coming Wave of Embedding Ecosystems. *Proc. VLDB Endow.* 14, 12 (2021), 3178–3181. https://doi.org/10.14778/3476311.3476402

[43] Thomas Pasquier, Xueyuan Han, Mark Goldstein, Thomas Moyer, David Eyers, Margo Seltzer, and Jean Bacon. 2017. Practical whole-system provenance capture. In *Proceedings of the 2017 Symposium on Cloud Computing*. 405–418.

[44] Thomas Pasquier, Xueyuan Han, Thomas Moyer, Adam Bates, Olivier Hermant, David Eyers, Jean Bacon, and Margo Seltzer. 2018. Runtime analysis of whole-system provenance. In *Proceedings of the 2018 ACM SIGSAC conference on computer and communications security*. 1601–1616.

[45] David A. Patterson, Joseph Gonzalez, Quoc V. Le, Chen Liang, Lluis-Miquel Munguia, Daniel Rothchild, David R. So, Maud Texier, and Jeff Dean. 2021. Carbon Emissions and Large Neural Network Training. *CoRR* abs/2104.10350 (2021). arXiv:2104.10350 https://arxiv.org/abs/2104.10350

[46] Vignesh T. Ravi, Michela Becchi, Gagan Agrawal, and Srimat T. Chakradhar. 2011. Supporting GPU sharing in cloud environments with a transparent runtime consolidation framework. In *Proceedings of the 20th ACM International Symposium on High Performance Distributed Computing, HPDC 2011, San Jose, CA, USA, June 8-11, 2011*, Arthur B. Maccabe and Douglas Thain (Eds.). ACM, 217–228. https://doi.org/10.1145/1996130.1996160

[47] Giampaolo Rodola. 2023-04-16. psutil documentation. https://psutil.readthedocs.io/en/latest/.

[48] Efraim Rotem, Alon Naveh, Avinash Ananthakrishnan, Eliezer Weissmann, and Doron Rajwan. 2012. Power-Management Architecture of the Intel Microarchitecture Code-Named Sandy Bridge. *IEEE Micro* 32, 2 (2012), 20–27. https://doi.org/10.1109/MM.2012.12

[49] Rathijit Sen and David A. Wood. 2017. Energy-Proportional Computing: A New Definition. *Computer* 50, 8 (2017), 26–33. https://doi.org/10.1109/MC.2017.3001248

[50] Rathijit Sen and David A. Wood. 2017. Pareto Governors for Energy-Optimal Computing. *ACM Trans. Archit. Code Optim.* 14, 1 (2017), 6:1–6:25. https://doi.org/10.1145/3046682

[51] Arman Shehabi, Sarah Smith, Dale Sartor, Richard Brown, Magnus Herrlin, Jonathan Koomey, Eric Masanet, Nathaniel Horner, Inês Azevedo, and William Lintner. 2016. United states data center energy usage report. . (2016).

[52] Kai Shen, Arrvindh Shriraman, Sandhya Dwarkadas, Xiao Zhang, and Zhuan Chen. 2013. Power containers: an OS facility for fine-grained power and energy management on multicore servers. In *Architectural Support for Programming Languages and Operating Systems, ASPLOS 2013, Houston, TX, USA, March 16-20, 2013*, Vivek Sarkar and Rastislav Bodík (Eds.). ACM, 65–76. https://doi.org/10.1145/2451116.2451124

[53] Uber. 2023. Meet Michelangelo: Uber's Machine Learning Platform. https://www.uber.com/en-CH/blog/michelangelo-machine-learning-platform/.

[54] Carole-Jean Wu, Ramya Raghavendra, Udit Gupta, Bilge Acun, Newsha Ardalani, Kiwan Maeng, Gloria Chang, Fiona Aga Behram, Jinshi Huang, Charles Bai, Michael Gschwind, Anurag Gupta, Myle Ott, Anastasia Melnikov, Salvatore Candido, David Brooks, Geeta Chauhan, Benjamin Lee, Hsien-Hsin S. Lee, Bugra Akyildiz, Maximilian Balandat, Joe Spisak, Ravi Jain, Mike Rabbat, and Kim M. Hazelwood. 2022. Sustainable AI: Environmental Implications, Challenges and Opportunities. In *Proceedings of Machine Learning and Systems 2022, MLSys 2022, Santa Clara, CA, USA, August 29 - September 1, 2022*, Diana Marculescu, Yuejie Chi, and Carole-Jean Wu (Eds.). mlsys.org. https://proceedings.mlsys.org/paper/2022/hash/ed3d2c21991e3bef5e069713af9fa6ca-Abstract.html

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