The Odd One Out: Energy is not like Other Metrics

VAASTAV ANAND, Max Planck Institute for Software-Systems, Germany
ZHIQIANG XIE, Max Planck Institute for Software-Systems, Germany
MATHEUS STOLET, Max Planck Institute for Software-Systems, Germany
ROBERTA DE VITI, Max Planck Institute for Software-Systems, Germany
THOMAS DAVIDSON, Max Planck Institute for Software-Systems, Germany
REYHANEH KARIMIPOUR, Max Planck Institute for Software-Systems, Germany
SAFYA ALZAYAT, Max Planck Institute for Software-Systems, Germany
JONATHAN MACE, Max Planck Institute for Software-Systems, Germany

Energy requirements for datacenters are growing at a fast pace. Existing techniques for making datacenters efficient focus on hardware. However, the gain in energy efficiency that can be achieved without making the applications energy-aware is limited. To overcome this limitation, recent work has proposed making the *software* running in datacenters energy aware. To do so, we must be able to track energy consumption at various granularities at the software level – (i) process level; (ii) application level; (iii) end-to-end request level.

Currently, existing software energy-tracking techniques primarily focus on tracking energy at the process or application level; only a few techniques track energy at an end-to-end request level. However, not tracking energy at an end-to-end request level can lead to false software optimizations and cause a decrease in energy efficiency.

To track energy at an end-to-end request level, we can leverage end-to-end tracking techniques for other metrics such as distributed tracing. However, we posit that energy cannot be treated as just another metric and that we cannot use existing frameworks without modifications. In this paper, we discuss how energy is different from other metrics and describe an energy-tracking workflow that leverages these differences and tracing techniques in order to track energy consumption of end-to-end requests.

CCS Concepts: • **Networks → Cloud computing**.

Additional Key Words and Phrases: observability, distributed tracing, energy measurement

1 INTRODUCTION

Datacenters deployed by cloud providers are responsible for 1% of the world's total energy consumption [42, 63]. The energy requirements of the cloud are growing unsustainably, with estimates showing that cloud computing may require 8% of the world's energy by the end of the current decade [63, 68]. To ensure that datacenters and cloud computing do not waste energy, we must strive to make datacenters as energy efficient as possible [2].

Researchers have come up with various techniques for increasing energy efficiency in datacenters. These techniques include, but are not limited to, using alternative energy sources for powering datacenters [29, 49], smart cooling of datacenters [66], swarm-based dynamic workload placement [3, 22, 72], user-specified energy policies for datacenter resource allocation [10, 14], dynamically adapting the energy consumption of a datacenter network [33], and switching datacenter networks to multichannel lightwave networks [21].

However, the energy efficiency that can be achieved without considering application design is limited, and there have been calls to make datacenter applications more energy-aware [2]. To do so, we must be able to track the energy usage and energy provenance of applications [2]. A key decision when tracking energy is deciding the granularity of measurements. We can track energy at multiple granularities – hardware, process, application, and end-to-end request – with each granularity enabling finer-grained control of energy usage.

Current techniques focus on energy at the granularity of a datacenter, hardware, process, or application (§3.2). However, we believe that tracking energy of end-to-end requests is just as important. In fact, as microservices and serverless architectures increase in popularity, tracking energy of a single process (or even machine node) in isolation is not useful. As a single request can visit thousands of service components, focusing only on separate measurements can be misleading. For example, if a service *A* compresses the data to be sent to service *B*, the network energy consumption for transferring the data will be reduced. However, the processor might consume more energy for compressing the data at *A*, leading to an increased overall energy consumption [54]. Thus, without the full end-to-end profile of a request, system designers might make incorrect decisions when trying to mitigate energy inefficiency. Currently, to the knowledge of the authors, there is no way of tracking energy at an end-to-end request level.

To attribute the energy usage for a given request, it is essential to track the request across different components of the application and system stack, and to measure the energy spent processing the request by each of these components. In principle we can use distributed tracing frameworks which work at the end-to-end request granularity [76]; these tracing frameworks record traces of the computations performed for a request by different parts of the application across multiple machines [6, 26, 79].

However, we posit that energy cannot be treated as just another metric and does not trivially integrate with distributed tracing frameworks. We identify three factors that distinguish energy from other typical metrics used today: (1) energy consumption is all-encompassing, i.e., every aspect of a system from software to hardware consumes energy; (2) the same computation running

Authors' addresses: Vaastav Anand, vaastav@mpi-sws.org, Max Planck Institute for Software-Systems, Germany; Zhiqiang Xie, zxie@mpi-sws.org, Max Planck Institute for Software-Systems, Germany; Matheus Stolet, mstolet@mpi-sws.org, Max Planck Institute for Software-Systems, Germany; Roberta De Viti, rdeviti@mpi-sws.org, Max Planck Institute for Software-Systems, Germany; Thomas Davidson, tdavidso@mpi-sws.org, Max Planck Institute for Software-Systems, Germany; Reyhaneh Karimipour, rkarimip@mpi-sws.org, Max Planck Institute for Software-Systems, Germany; Safya Alzayat, salzayat@mpi-sws.org, Max Planck Institute for Software-Systems, Germany; Jonathan Mace, jcmace@mpi-sws.org, Max Planck Institute for Software-Systems, Campus E1 5, Saarbruecken, Saarland, Germany, 66123.

on different hardware, platforms, or environments, can vary substantially in its energy usage; (3) the motivation behind measuring energy consumption is different, as it is neither performance nor correctness driven; and (4) energy cannot be measured at extremely fine granularities like, e.g., the performance counters of modern systems.

Based on these differences, we consider how to track energy consumption at the end-to-end request level, and how existing coarse-grained energy-measurement techniques might integrate with end-to-end distributed tracing frameworks. In this paper, we first motivate the use of end-to-end request level energy tracking by describing potential use cases for leveraging measurement of energy consumption at the end-to-end request level (§2); then, we compare and contrast the measurement of energy consumption with other existing metrics (§3), propose a model for tracking energy at the end-to-end request level (§4), and finally provide future directions that would help improve energy measurement (§5).

2 END-TO-END MEASUREMENT USE CASES

We begin with several example use cases of how application developers might leverage end-to-end energy tracking.

Design Exploration. Changing the design and implementation of an application can drastically impact its energy consumption [46]. There has been significant work investigating the effect of different designs on the energy efficiency of applications. These techniques analyze and optimize along various dimensions and at various levels in the software stack. Examples include the choice of implementation language [67], the application's memory usage profile [36, 85], parallelizability of the application [83], usage and storing of data [53], thread management [70], implementation choices for commonly used libraries [58, 71], use of compression [13], use of software design patterns [52], and co-location of RPC services [35].

Several use cases require the ability to measure energy for small blocks of code *internal* to the application, that may not directly map to a specific thread, core, or process [30]. Consider the compression example from §1; it is necessary to measure the energy consumption of the compression logic to compare the application with and without compression.

Energy Adaptive Computing. Metric measurements often feed directly into control mechanisms at runtime, such as request scheduling, load-balancing, and data quality decisions. Energy is no different, and numerous prior works use real-time energy measurements to make energy-based tradeoffs at runtime [2, 4, 5, 9, 11, 12, 15, 27, 34, 45, 77, 87]. Similar to recent work examining the use of machine learning models to service requests faster at the cost of accuracy [31, 84], energy-adaptive computing can also make decisions between accuracy and energy consumption.

In the context of web applications, we define energy-adaptive computing as the ability of the application to choose different execution paths leading to potentially different outputs for the same input based on the energy already used by a request. Under energy-adaptive computing, every application component can decide what computation should be performed for a request, in order to service that request within a given energy budget. A hypothetical example would be Google Search providing accurate results limited to the first page of results, and not pre-computing results for the subsequent page unless explicitly asked, if a search request has already used a large fraction of its energy budget. This specific use case is viable because the click-through rate exponentially decreases as the position of the result increases [8], leading to results on page 2 having a click-through rate lower than 7% [60, 69]. In a similar vein, the Green framework reduced the energy consumption of Bing Search by limiting the maximum number of documents that each search query processed [5].

To be able to effectively control the energy usage of the application at a per-request level to meet energy efficiency goals, developers must have access to per-request energy consumption information as well as an estimate of the total energy used by the application at any given point in time during the execution of the request.

Dead Execution Elimination. Like a router dropping packets when saturated, an application can immediately reject an incoming request to minimize energy waste if a latency SLO dictates that this request will be subsequently dropped. To maximize energy efficiency, requests must be dropped as early as possible in the execution pipeline.

Admission-control algorithms must accurately predict the energy usage of an incoming request. To do so, they require energy to be measured at an end-to-end request level in tandem with aggregate energy-usage metrics at higher granularities. In fact, basing admission-control decisions on high-granularity energy-usage metrics alone may either result in more requests being dropped than necessary, leading to lower throughput, or fewer requests being dropped, leading to energy waste.

The use cases described above have some underlying commonalities. All use cases need energy tracking at the end-to-end request level, whilst also potentially having access to hardware-level energy measurements. Some use cases record measurements at runtime for later offline analysis and modelling, correlating coarse-grained measurements with fine-grained application traces; others use measurements immediately for runtime decision-making and prediction; while others use a combination of background modelling and runtime decision-making. In all cases we face a similar challenge: application-level and end-to-end request level energy usage is not easily measured.

3 MEASURING ENERGY

3.1 How do we measure systems?

A possible approach to measuring energy at an end-to-end request level is to simply apply existing techniques and frameworks designed for other metrics. As mentioned in §1, metrics are useful for a range of use cases and arise at different granularities, which we describe here.

Application. Metrics at the application level focus on performance data such as utilization, saturation, and failure signals. Performance metrics give insight into how an application is running. Common examples of performance metrics are latency, throughput, and queuing time. Latency measurements (e.g., average latency,

latency distribution, and jitter) are important for user-facing applications and are a common service-level objective (SLO) [43, 64, 73]. At application level, throughput is measured as the number of requests or operations completed by the application. Saturation metrics refer to system backlogging, e.g., the length of queues. Failure signals track the correctness behaviour of the application; an example is the total number of API call exceptions [51].

Process. Metrics of interest at the process level include CPU utilization, memory utilization, bytes transmitted and received, and thread count. Data from these metrics can be analyzed to identify any outlier process consuming too much memory or monopolizing the CPU.

Request. End-to-end request level metrics [44, 56, 79] give a full-system view and help gauge how a user is impacted (e.g., latency). Most metrics at the request level provide elementary information that is aggregated at higher granularities to help diagnose system issues. At the request level, metrics are mostly produced rather than consumed. For instance, the end-to-end request latency is recorded at the request level, while other granularities, such as the application, consume and aggregate individual request latency measurements to report the average latency of the application.

Hardware. Metrics at the hardware granularity are parsed, aggregated, and analyzed by coarser-grained metrics to solve a multitude of software or hardware issues. For instance, network-level information on link failures, packet drops, and flow helps to ensure successful network operation [86]; spikes in CPU temperature are used to explain reduction of clock speeds [23]; and, cache occupancy is used to detect side-channel attacks [7].

3.2 How do we measure energy?

Energy Consumption Models. According to a recent survey of energy-consumption modelling techniques [18], prior work proposes several methods to model the energy at a datacenter level to make datacenters more energy-efficient [1, 16, 41, 57, 61, 74, 75, 82]. For instance, there are estimation models to approximate the energy usage of servers [48, 62] or of the entire datacenter network, to improve the energy efficiency of the network [33, 50, 80]. Furthermore, there are techniques modelling energy consumption as a function of the resource usage of each application process [19], and others requiring additional external hardware for measurements [24].

Energy Measurements. Measuring energy begins at the hardware level before being aggregated or sliced at different granularities. Modern Intel processors provide energy measurements for servers through CPU registers that are updated approximately every 1ms [39]. These counters provide numbers for CPU and memory controller power consumption based on the Running Average Power Limit (RAPL) [17]. RAPL measurements can then be further broken down at the process and function levels.

Fine-Grained Measurements. Some libraries build upon Intel's RAPL measurements [39] to provide fine-grained process-level measurements. For example, Scaphandre [37] provides process-level metrics over large time periods sampled at 1s intervals [38]. However, RAPL registers need to be continuously polled for a higher temporal resolution; this polling itself requires consumption of energy which distorts the reported energy information due to the observer effect [81]. HAECER [30] uses RAPL to compute energy consumption for short data paths, e.g., functions. HAECER requires insertion of delay loops to synchronize the update rate of RAPL's counters. While promising, these approaches do not yet meet the low latency demands of microservice applications that often have SLOs on the order of milliseconds or microseconds. Other monitoring tools model energy consumption for functions in a single machine based on raw information available from hardware devices [65].

Power Containers [78] measure request-level power consumption of, e.g., a webserver. Power Containers operate at an OS level by using cheaper, high-frequency CPU, IO, and memory counters as a proxy for calculating finer-grained energy consumption. The approach proposed by Power Containers is promising, but has limiting assumptions about execution structure (e.g., no application-level data dependencies across threads) that do not broadly apply in modern RPC servers.

End-to-End Measurements. Prior work on distributed energy measurement has primarily focused on embedded systems [25, 47]. For example, Quanto [25] requires an energy meter [20] to be attached to every piece of hardware and provides task tracking across different parts of hardware. However, Quanto is designed for embedded systems that use a different, simplified programming model and run on nodes with TinyOS [47]. By contrast, the hardware, concurrency, and execution structure of datacenter applications are significantly more complex.

3.3 How is energy different?

Energy consumption is all-encompassing. Energy consumption is distributed across different system devices. Every single system device contributes to the overall energy usage. For instance, given the rising trends towards programmable devices (e.g., GPUs, TPUs, programmable ASICs, programmable switches) to discuss energy consumption we need to consider the energy consumption of all of these devices. Currently, there is no unifying abstraction that supports measuring energy from all devices in a usable way at a request level. By contrast, other system metrics can simply be measured end-to-end and do not require individual measurements from each device in the system.

Energy consumption is dynamic. Many traditional metrics, such as resource consumption (e.g., disk, CPU, network), depend only on the application logic or state – the same application running on different hardware or platforms will not vary significantly. By contrast, energy consumption can vary significantly depending on factors at all levels, from hardware to application: e.g., the specific hardware; co-location with other applications [35]; physical proximity of distributed components [35]; concurrent requests and tasks within the application [70]; idleness of cores [28]; application-level optimizations like batching and compression [54].

The motivation behind measuring energy is different. The motivation for measuring energy consumption differs from the motivation for metrics in §3.1. In fact, most system metrics are used to diagnose performance issues, maintain quality of service, or debug

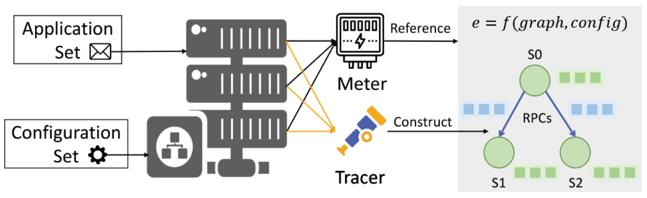

Fig. 1. GNN model training workflow for energy estimation

correctness issues. For instance, systems monitor the tail-latency of requests in order to maintain a certain quality of service; throughput numbers are collected to diagnose performance issues in an application; error or crashes are recorded to identify bugs. Unlike these metrics, energy measurement is neither performance nor correctness driven. Although energy measurement can be used for aggregate analysis, energy measurement has found a very important use in energy-adaptive applications that change behaviour based on energy consumption data.

Hardware does not support fine-grained software-level measurements. Unlike other metrics, commodity hardware does not yet support energy tracking and measurement at fine-grained software level granularities. For instance, Intel provides energy measurements in servers via its RAPL interface. However, using RAPL presents three main disadvantages: (i) the RAPL registers need to be continuously polled, which consumes more energy; (ii) RAPL measurements can only provide granularity up to 1ms [39, Vol.3,Ch.14]; (iii) RAPL might be subject to side-channel attacks [40]. An alternative to RAPL is to adopt specialized hardware that allows finer-grained energy tracking. However, as accurate energy tracking must cross process and machine level boundaries, we need solutions at finer-grained software level granularities.

4 MODELING ENERGY CONSUMPTION OF INDIVIDUAL REQUESTS

Since it is not feasible to directly measure the energy consumption of individual requests, we believe that any energy tracking solution would need to adopt a hybrid-measurement and model-based approach to *estimate* energy consumption. To support the use cases in §2, we have to support energy modelling in both an offline mode and an online mode. The offline mode should provide energy measurements of a request for offline analysis. The online mode should track the real-time energy usage of a request and predict the future energy usage of that request at any point during the request execution, while inducing minimal overhead.

4.1 Leveraging Distributed Tracing

As highlighted in §3.2, we cannot directly measure energy at a fine-grained request level, but we can make estimates. Current approaches, such as Power Containers [78], model the energy usage of a request for a single server based on CPU, IO, and memory counters. These counters are then combined with coarser energy measurements to attribute consumption to the tasks executing within a time interval. In the distributed setting, Facebook has leveraged distributed tracing to obtain CPU utilization information from each node in the system for understanding the relationship between power consumption and CPU utilization [28]. Each approach on its own does not provide all the data needed to model energy consumption at an end-to-end request level. However, by combining these two approaches, we can obtain the relevant information for modelling energy consumption. In particular, we leverage distributed tracing for data collection in three ways – (i) Metric Collection, (ii) API Profiling, and (iii) Data Correlation.

Metric Collection. First, we use distributed tracing to collect fine-grained system information (e.g., the topology of services and machines visited by a request, hardware configurations, execution time, bytes to write to storage or send via network, and utilization of CPU, GPU, and other accelerators). Then, we leverage such information to compose machine learning (ML) features that are highly correlated with energy consumption. To do so, we employ feature engineering, namely the process of selecting and transforming data into features for supervised ML. Feature engineering has proven to be crucial in applying ML to optimize system behavior [32, 59].

API Profiling. For each request, we extract the list of APIs executed by the request; for each API, we measure the energy consumption locally, on each machine visited by the request, using Intel's RAPL registers. From these measurements, we derive a baseline for the energy consumption of each API. Note that we are not using the Intel RAPL register during production, but *prior* to production, to get fine-grained isolated energy measurements.

Data Correlation. To generate meaningful predictions in production, any model would need energy-usage measurements of different parts of an end-to-end request. We leverage existing infrastructure for *context propagation* in distributed tracing [55] to propagate general tracing metrics as well as energy measurements between processes, in order to build a complete profile for each request. Each profile consists of information on all the services visited by the request, plus the order and dependencies among these services, which can be used to build an invocation timeline or a service graph.

4.2 Design Concerns – Offline Mode

For the offline mode, we propose a Graph Neural Network (GNN) model to better utilize the structural information in request traces; this structural information reveals order and causality of the services visited by the request, and can be used to detail communications between these services and their energy consumption.

We use distributed tracing to build the service graph of a request, then we train parametrized GNN functions on the graph. As shown in Figure 1, we model energy consumption in four stages: (i) Graph Creation; (ii) Graph Augmentation; (iii) Model Training; and (iv) Model Usage.

Graph Creation. We use request traces to construct a service graph where nodes represent the services visited by the request, and edges represent communication between nodes.

Graph Augmentation. We annotate each node (service) with its hardware configuration, additional metrics collected by the tracer, and (local) energy measurements of each API exposed by that node.

Furthermore, we annotate each edge with network device specifications, bandwidth, and bytes to be transmitted on that edge.

Model Training. We encode annotations into features to train the GNN model via supervised learning. We train the model under different system configuration scenarios, which we randomly generate from three configuration sets: service, workload, and hardware. In particular, we randomly iterate over the sets to deploy the given services, with the given hardware, under the given workload. Finally, we measure the end-to-end request energy consumption on each service; as a reference, we use the energy usage measured from motherboards. We validate the model in real production systems.

Model Usage. After a request execution, the GNN model estimates the energy consumption of that request at every node and edge (of the service graph) visited by the request.

Such a model would enable developers to identify energy usage hotspots and improve the overall energy efficiency of applications. In addition, estimating energy consumption under different hardware configurations would help developers to deploy energy-efficient configurations under different service and workload scenarios.

4.3 Design Concerns – Online Mode

For the online mode, we need a model that can quickly predict current and future energy usage for an executing request. To this end, we propose to deploy a simple *predictor* model at every node of the system. Each predictor leverages the estimates of the offline GNN model for recent requests in a given time window. These predictions become part of the request's baggage [55], which can be used to make energy-aware decisions down the execution pipeline.

Generating Online Estimates. Each node in the service graph of the GNN model tracks the node's energy usage of each API for recent requests. For each API, the node's predictor computes an average energy-usage estimate based on recent requests. Periodically, the server running the GNN model communicates to each node the estimated energy-usage information for that specific node, which each node uses to update its local predictor model.

Measurement Carrier. Each note visited by the request uses its local predictor to annotate that request and its (local) energy-usage estimate. This annotation is propagated along with the request as part of its context-specific baggage [55], which other components can use to make energy-aware computation decisions.

5 FUTURE IMPLICATIONS

In some scenarios, dedicated meters have been attached to certain hardware to enable accurate real-time energy monitoring [20]. While hardware meters are not able to measure end-to-end request energy consumption, their deployment can help the tracer to get more accurate data.

Furthermore, to encourage application developers to treat energy as a first-class citizen, cloud providers can shift towards energy-consumption based pricing.

Finally, we can involve service customers: as more people are committed to reduce carbon emissions, the energy consumption of a request may be of increasing relevance to customers. We envision services displaying to end users the energy-consumption estimates of their requests, along the lines of flight booking services, which annotate each flight option with their estimated carbon emissions to encourage customers to make a more environment-friendly choice.

REFERENCES

[1] Bernard Aebischer and Lorenz M Hilty. 2015. The energy demand of ICT: a historical perspective and current methodological challenges. In *ICT Innovations for Sustainability*. Springer, 71–103.

[2] 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).

[3] Ionut Anghel, Cristina Bianca Pop, Tudor Cioara, Ioan Salomie, and Iulia Vartic. 2013. A swarm-inspired technique for self-organizing and consolidating data centre servers. *Scalable Computing: Practice and Experience* 14, 2 (2013), 69–82.

[4] Timur Babakol, Anthony Canino, Khaled Mahmoud, Rachit Saxena, and Yu David Liu. 2020. Calm energy accounting for multithreaded java applications. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 976–988.

[5] Woongki Baek and Trishul M Chilimbi. 2010. Green: A framework for supporting energy-conscious programming using controlled approximation. In *Proceedings of the 31st ACM SIGPLAN Conference on Programming Language Design and Implementation*. 198–209.

[6] Paul Barham, Austin Donnelly, Rebecca Isaacs, and Richard Mortier. 2004. Using Magpie for request extraction and workload modelling.. In *OSDI*, Vol. 4. 18–18.

[7] Mohammad-Mahdi Bazm, Thibaut Sautereau, Marc Lacoste, Mario Sudholt, and Jean-Marc Menaud. 2018. Cache-based side-channel attacks detection through intel cache monitoring technology and hardware performance counters. In *2018 Third International Conference on Fog and Mobile Edge Computing (FMEC)*. IEEE, 7–12.

[8] Johannes Beus. 2020. Why (almost) everything you knew about Google CTR is no longer valid. https://www.sistrix.com/blog/why-almost-everything-you-knew-about-google-ctr-is-no-longer-valid/.

[9] Brett Boston, Adrian Sampson, Dan Grossman, and Luis Ceze. 2015. Probability type inference for flexible approximate programming. In *Proceedings of the 2015 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications*. 470–487.

[10] Rajkumar Buyya, Anton Beloglazov, and Jemal Abawajy. 2010. Energy-efficient management of data center resources for cloud computing: A vision, architectural elements, and open challenges. *arXiv preprint arXiv:1006.0308* (2010).

[11] Anthony Canino and Yu David Liu. 2017. Proactive and adaptive energy-aware programming with mixed typechecking. *ACM SIGPLAN Notices* 52, 6 (2017), 217–232.

[12] Anthony Canino, Yu David Liu, and Hidehiko Masuhara. 2018. Stochastic energy optimization for mobile GPS applications. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 703–713.

[13] Yanpei Chen, Archana Ganapathi, and Randy H Katz. 2010. To compress or not to compress-compute vs. io tradeoffs for mapreduce energy efficiency. In *Proceedings of the first ACM SIGCOMM workshop on Green networking*. 23–28.

[14] Dazhao Cheng, Palden Lama, Changjun Jiang, and Xiaobo Zhou. 2015. Towards energy efficiency in heterogeneous Hadoop clusters by adaptive task assignment. In *2015 IEEE 35th International Conference on Distributed Computing Systems*. IEEE, 359–368.

[15] Michael Cohen, Haitao Steve Zhu, Emgin Ezgi Senem, and Yu David Liu. 2012. Energy types. In *Proceedings of the ACM international conference on Object oriented programming systems languages and applications*. 831–850.

[16] Georges Da Costa, Andreas Kopecki, Ariel Oleksiak, Jean-Marc Pierson, Tomasz Piontek, Eugen Volk, Stefan Wesner, et al. 2012. Modeling and simulation of data center energy-efficiency in CoolEmAll. In *International Workshop on Energy Efficient Data Centers*. Springer, 25–36.

[17] Howard David, Eugene Gorbatov, Ulf R Hanebutte, Rahul Khanna, and Christian Le. 2010. RAPL: Memory power estimation and capping. In *2010 ACM/IEEE International Symposium on Low-Power Electronics and Design (ISLPED)*. IEEE, 189–194.

[18] Miyuru Dayarathna, Yonggang Wen, and Rui Fan. 2015. Data center energy consumption modeling: A survey. *IEEE Communications Surveys & Tutorials* 18, 1 (2015), 732–794.

[19] Thanh Do, Suhib Rawshdeh, and Weisong Shi. 2009. ptop: A process-level power profiling tool.

[20] Prabal Dutta, Mark Feldmeier, Joseph Paradiso, and David Culler. 2008. Energy metering for free: Augmenting switching regulators for real-time monitoring. In *2008 International Conference on Information Processing in Sensor Networks (ipsn 2008)*. IEEE, 283–294.

[21] Y Shaya Fainman, Joseph Ford, William M Mellette, Shayan Mookherjea, George Porter, Alex C Snoeren, George Papen, Saman Saeedi, John Cunningham, Ashok

Krishnamoorthy, et al. 2019. Leed: A lightwave energy-efficient datacenter. In *Optical Fiber Communication Conference*. Optical Society of America, M4D–4.

[22] Eugen Feller, Louis Rilling, and Christine Morin. 2011. Energy-aware ant colony based workload placement in clouds. In *2011 IEEE/ACM 12th International Conference on Grid Computing*. IEEE, 26–33.

[23] Alexandre P. Ferreira, Daniel Mosse, and Jae C. Oh. 2007. Thermal Faults Modeling Using a RC Model with an Application to Web Farms. In *19th Euromicro Conference on Real-Time Systems (ECRTS'07)*. 113–124. https://doi.org/10.1109/ECRTS.2007.36

[24] Jason Flinn and Mahadev Satyanarayanan. 1999. Powerscope: A tool for profiling the energy usage of mobile applications. In *Proceedings WMCSA'99. Second IEEE Workshop on Mobile Computing Systems and Applications*. IEEE, 2–10.

[25] Rodrigo Fonseca, Prabal Dutta, Philip Levis, and Ion Stoica. 2008. Quanto: Tracking Energy in Networked Embedded Systems.. In *OSDI*, Vol. 8. 323–338.

[26] Rodrigo Fonseca, George Porter, Randy H. Katz, and Scott Shenker. 2007. X-Trace: A Pervasive Network Tracing Framework. In *4th USENIX Symposium on Networked Systems Design & Implementation (NSDI 07)*. USENIX Association, Cambridge, MA. https://www.usenix.org/conference/nsdi-07/x-trace-pervasive-network-tracing-framework

[27] Benjamin Gaudette, Carole-Jean Wu, and Sarma Vrudhula. 2016. Improving smartphone user experience by balancing performance and energy with probabilistic QoS guarantee. In *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 52–63.

[28] Alexander Gilgur, Brian Coutinho, Iyswarya Narayanan, and Parth Malani. 2021. Transitive Power Modeling for Improving Resource Efficiency in a Hyperscale Datacenter. In *Companion Proceedings of the Web Conference 2021*. 182–191.

[29] Íñigo Goiri, William Katsak, Kien Le, Thu D Nguyen, and Ricardo Bianchini. 2013. Parasol and GreenSwitch: Managing datacenters powered by renewable energy. *ACM SIGPLAN Notices* 48, 4 (2013), 51–64.

[30] Marcus Hähnel, Björn Döbel, Marcus Völp, and Hermann Härtig. 2012. Measuring energy consumption for short code paths using RAPL. *ACM SIGMETRICS Performance Evaluation Review* 40, 3 (2012), 13–17.

[31] Matthew Halpern, Behzad Boroujerdian, Todd Mummert, Evelyn Duesterwald, and Vijay Janapa Reddi. 2019. One size does not fit all: Quantifying and exposing the accuracy-latency trade-off in machine learning cloud service apis via tolerance tiers. *arXiv preprint arXiv:1906.11307* (2019).

[32] Mingzhe Hao, Levent Toksoz, Nanqinqin Li, Edward Edberg Halim, Henry Hoffmann, and Haryadi S Gunawi. 2020. LinnOS: Predictability on Unpredictable Flash Storage with a Light Neural Network. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. 173–190.

[33] Brandon Heller, Srinivasan Seetharaman, Priya Mahadevan, Yiannis Yiakoumis, Puneet Sharma, Sujata Banerjee, and Nick McKeown. 2010. Elastictree: Saving energy in data center networks.. In *Nsdi*, Vol. 10. 249–264.

[34] Henry Hoffmann, Stelios Sidiroglou, Michael Carbin, Sasa Misailovic, Anant Agarwal, and Martin Rinard. 2011. Dynamic knobs for responsive power-aware computing. *ACM SIGARCH computer architecture news* 39, 1 (2011), 199–212.

[35] Chang-Hong Hsu, Qingyuan Deng, Jason Mars, and Lingjia Tang. 2018. Smoothoperator: Reducing power fragmentation and improving power utilization in large-scale datacenters. In *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*. 535–548.

[36] Chung-Hsing Hsu and Ulrich Kremer. 2003. The design, implementation, and evaluation of a compiler algorithm for CPU energy reduction. In *Proceedings of the ACM SIGPLAN 2003 conference on Programming language design and implementation*. 38–48.

[37] hubblo-org. 2021. Scaphandre. https://github.com/hubblo-org/scaphandre.

[38] hubblo-org. 2022. How scaphandre computes per process power consumption. https://hubblo-org.github.io/scaphandre-documentation/explanations/how-scaph-computes-per-process-power-consumption.html.

[39] Intel. 2022. Combined Volume Set of Intel 64 and IA-32 Architectures Software Developer's Manual. https://www.intel.com/content/www/us/en/developer/articles/technical/intel-sdm.html.

[40] Intel. 2022. Running Average Power Limit Energy Reporting / CVE-2020-8694 , CVE-2020-8695 / INTEL-SA-00389. https://www.intel.com/content/www/us/en/developer/articles/technical/software-security-guidance/advisory-guidance/running-average-power-limit-energy-reporting.html.

[41] Mohammad A Islam, Shaolei Ren, and Gang Quan. 2013. Online energy budgeting for virtualized data centers. In *2013 IEEE 21st International Symposium on Modelling, Analysis and Simulation of Computer and Telecommunication Systems*. IEEE, 424–433.

[42] Nicola Jones. 2018. How to stop data centres from gobbling up the world's electricity. *Nature* 561, 7722 (2018), 163–167.

[43] Kostis Kaffes, Timothy Chong, Jack Tigar Humphries, Adam Belay, David Mazières, and Christos Kozyrakis. 2019. Shinjuku: Preemptive Scheduling for Microsecond-scale Tail Latency. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*. USENIX Association, Boston, MA, 345–360. https://www.usenix.org/conference/nsdi19/presentation/kaffes

[44] Jonathan Kaldor, Jonathan Mace, Michał Bejda, Edison Gao, Wiktor Kuropatwa, Joe O'Neill, Kian Win Ong, Bill Schaller, Pingjia Shan, Brendan Viscomi, Vinod Venkataraman, Kaushik Veeraraghavan, and Yee Jiun Song. 2017. Canopy: An End-to-End Performance Tracing And Analysis System. In *Proceedings of the 26th Symposium on Operating Systems Principles* (Shanghai, China) *(SOSP '17)*. Association for Computing Machinery, New York, NY, USA, 34–50. https://doi.org/10.1145/3132747.3132749

[45] Aman Kansal, Scott Saponas, AJ Bernheim Brush, Kathryn S McKinley, Todd Mytkowicz, and Ryder Ziola. 2013. The latency, accuracy, and battery (lab) abstraction: programmer productivity and energy efficiency for continuous mobile context sensing. *ACM SIGPLAN Notices* 48, 10 (2013), 661–676.

[46] Aman Kansal and Feng Zhao. 2008. Fine-grained energy profiling for power-aware application design. *ACM SIGMETRICS Performance Evaluation Review* 36, 2 (2008), 26–31.

[47] Philip Levis, Samuel Madden, Joseph Polastre, Robert Szewczyk, Kamin Whitehouse, Alec Woo, David Gay, Jason Hill, Matt Welsh, Eric Brewer, et al. 2005. TinyOS: An operating system for sensor networks. In *Ambient intelligence*. Springer, 115–148.

[48] Adam Wade Lewis, Soumik Ghosh, and Nian-Feng Tzeng. 2008. Run-time Energy Consumption Estimation Based on Workload in Server Systems. *HotPower* 8 (2008), 17–21.

[49] Chao Li, Rui Wang, Tao Li, Depei Qian, and Jingling Yuan. 2014. Managing green datacenters powered by hybrid renewable energy systems. In *11th International Conference on Autonomic Computing (ICAC 14)*. 261–272.

[50] Dan Li, Yunfei Shang, and Congjie Chen. 2014. Software defined green data center network with exclusive routing. In *IEEE INFOCOM 2014-IEEE Conference on Computer Communications*. IEEE, 1743–1751.

[51] Ze Li, Qian Cheng, Ken Hsieh, Yingnong Dang, Peng Huang, Pankaj Singh, Xinsheng Yang, Qingwei Lin, Youjiang Wu, Sebastien Levy, and Murali Chintalapati. 2020. Gandalf: An Intelligent, End-To-End Analytics Service for Safe Deployment in Large-Scale Cloud Infrastructure. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*. USENIX Association, Santa Clara, CA, 389–402. https://www.usenix.org/conference/nsdi20/presentation/li

[52] Andreas Litke, Kostas Zotos, Alexander Chatzigeorgiou, and George Stephanides. 2005. Energy consumption analysis of design patterns. In *Proceedings of the International Conference on Machine Learning and Software Engineering*. 86–90.

[53] Kenan Liu, Gustavo Pinto, and Yu David Liu. 2015. Data-oriented characterization of application-level energy optimization. In *International Conference on Fundamental Approaches to Software Engineering*. Springer, 316–331.

[54] Yung-Hsiang Lu, Qinru Qiu, Ali R Butt, and Kirk W Cameron. 2011. End-to-end energy management. *Computer* 44, 11 (2011), 75–77.

[55] Jonathan Mace and Rodrigo Fonseca. 2018. Universal context propagation for distributed system instrumentation. In *Proceedings of the thirteenth EuroSys conference*. 1–18.

[56] Jonathan Mace, Ryan Roelke, and Rodrigo Fonseca. 2015. Pivot Tracing: Dynamic Causal Monitoring for Distributed Systems. In *Proceedings of the 25th Symposium on Operating Systems Principles* (Monterey, California) *(SOSP '15)*. Association for Computing Machinery, New York, NY, USA, 378–393. https://doi.org/10.1145/2815400.2815415

[57] A Hasan Mahmud and Shaolei Ren. 2013. Online capacity provisioning for carbon-neutral data center with demand-responsive electricity prices. *ACM SIGMETRICS Performance Evaluation Review* 41, 2 (2013), 26–37.

[58] Irene Manotas, Lori Pollock, and James Clause. 2014. Seeds: A software engineer's energy-optimization decision support framework. In *Proceedings of the 36th International Conference on Software Engineering*. 503–514.

[59] Hongzi Mao, Malte Schwarzkopf, Shaileshh Bojja Venkatakrishnan, Zili Meng, and Mohammad Alizadeh. 2019. Learning scheduling algorithms for data processing clusters. In *Proceedings of the ACM special interest group on data communication*. 270–288.

[60] Leverage Marketing. 2017. How Far Down the Search Engine Results Page Will Most People Go? https://www.theleverageway.com/blog/how-far-down-the-search-engine-results-page-will-most-people-go/.

[61] Eric R Masanet, Richard E Brown, Arman Shehabi, Jonathan G Koomey, and Bruce Nordman. 2011. Estimating the energy use and efficiency potential of US data centers. *Proc. IEEE* 99, 8 (2011), 1440–1453.

[62] Microsoft. 2010. Joulemeter. http://research.microsoft.com/en-us/projects/joulemeter/.

[63] Goldman on Greener Software Microsoft Teams Up With Accenture. 2021. https://www.bloomberg.com/news/articles/2021-05-25/microsoft-teams-up-with-accenture-goldman-on-greener-software. Bass, Dina.

[64] Jeffrey C. Mogul and Ramana Rao Kompella. 2015. Inferring the Network Latency Requirements of Cloud Tenants. In *15th Workshop on Hot Topics in Operating Systems (HotOS XV)*. USENIX Association, Kartause Ittingen, Switzerland. https://www.usenix.org/conference/hotos15/workshop-program/presentation/mogul

[65] Adel Noureddine, Aurelien Bourdon, Romain Rouvoy, and Lionel Seinturier. 2012. Runtime monitoring of software energy hotspots. In *2012 Proceedings of the 27th*

IEEE/ACM International Conference on Automated Software Engineering. IEEE, 160–169.

[66] Chandrakant D Patel, Cullen E Bash, Ratnesh Sharma, Monem Beitelmal, and Rich Friedrich. 2003. Smart cooling of data centers. In *International Electronic Packaging Technical Conference and Exhibition*, Vol. 36908. 129–137.

[67] Rui Pereira, Marco Couto, Francisco Ribeiro, Rui Rua, Jácome Cunha, João Paulo Fernandes, and João Saraiva. 2017. Energy efficiency across programming languages: how do energy, time, and memory relate?. In *Proceedings of the 10th ACM SIGPLAN International Conference on Software Language Engineering*. 256–267.

[68] Mark Pesce. 2021. Cloud Computing's Coming Energy Crisis: The cloud's electricity needs are growing unsustainably. https://spectrum.ieee.org/cloud-computings-coming-energy-crisis.

[69] Philip Petrescu. 2014. Google Organic Click-Through Rates in 2014. https://moz.com/blog/google-organic-click-through-rates-in-2014.

[70] Gustavo Pinto, Fernando Castor, and Yu David Liu. 2014. Understanding energy behaviors of thread management constructs. In *Proceedings of the 2014 ACM International Conference on Object Oriented Programming Systems Languages & Applications*. 345–360.

[71] Gustavo Pinto, Kenan Liu, Fernando Castor, and Yu David Liu. 2016. A comprehensive study on the energy efficiency of java's thread-safe collections. In *2016 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 20–31.

[72] Cristina Bianca Pop, Ionut Anghel, Tudor Cioara, Ioan Salomie, and Iulia Vartic. 2012. A swarm-inspired data center consolidation methodology. In *Proceedings of the 2nd International Conference on Web Intelligence, Mining and Semantics*. 1–7.

[73] Haoran Qiu, Subho S. Banerjee, Saurabh Jha, Zbigniew T. Kalbarczyk, and Ravishankar K. Iyer. 2020. FIRM: An Intelligent Fine-grained Resource Management Framework for SLO-Oriented Microservices. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. USENIX Association, 805–825. https://www.usenix.org/conference/osdi20/presentation/qiu

[74] Ramya Raghavendra, Parthasarathy Ranganathan, Vanish Talwar, Zhikui Wang, and Xiaoyun Zhu. 2008. No "power" struggles: coordinated multi-level power management for the data center. In *Proceedings of the 13th international conference on Architectural support for programming languages and operating systems*. 48–59.

[75] Suzanne Marion Rivoire. 2008. *Models and metrics for energy-efficient computer systems*. Stanford University.

[76] Raja R Sambasivan, Ilari Shafer, Jonathan Mace, Benjamin H Sigelman, Rodrigo Fonseca, and Gregory R Ganger. 2016. Principled workflow-centric tracing of distributed systems. In *Proceedings of the Seventh ACM Symposium on Cloud Computing*. 401–414.

[77] Adrian Sampson, Werner Dietl, Emily Fortuna, Danushen Gnanapragasam, Luis Ceze, and Dan Grossman. 2011. EnerJ: Approximate data types for safe and general low-power computation. *ACM SIGPLAN Notices* 46, 6 (2011), 164–174.

[78] 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. *ACM SIGARCH Computer Architecture News* 41, 1 (2013), 65–76.

[79] Benjamin H Sigelman, Luiz Andre Barroso, Mike Burrows, Pat Stephenson, Manoj Plakal, Donald Beaver, Saul Jaspan, and Chandan Shanbhag. 2010. Dapper, a large-scale distributed systems tracing infrastructure. (2010).

[80] Indra Widjaja, Anwar Walid, Yanbin Luo, Yang Xu, and H Jonathan Chao. 2013. Small versus large: Switch sizing in topology design of energy-efficient data centers. In *2013 IEEE/ACM 21st International Symposium on Quality of Service (IWQoS)*. IEEE, 1–6.

[81] Wikipedia. 2022. Observer effect(physics). https://en.wikipedia.org/wiki/Observer_effect_(physics).

[82] Yuan Yao, Longbo Huang, Abhishek B Sharma, Leana Golubchik, and Michael J Neely. 2012. Power cost reduction in distributed data centers: A two-time-scale approach for delay tolerant workloads. *IEEE Transactions on Parallel and Distributed Systems* 25, 1 (2012), 200–211.

[83] Ivan Zecena, Ziliang Zong, Rong Ge, Tongdan Jin, Zizhong Chen, and Meikang Qiu. 2012. Energy consumption analysis of parallel sorting algorithms running on multicore systems. In *2012 International Green Computing Conference (IGCC)*. IEEE, 1–6.

[84] Jeff Zhang, Sameh Elnikety, Shuayb Zarar, Atul Gupta, and Siddharth Garg. 2020. Model-Switching: Dealing with Fluctuating Workloads in Machine-Learning-as-a-Service Systems. In *12th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 20)*.

[85] Yumin Zhang, Xiaobo Hu, and Danny Z Chen. 2002. Efficient global register allocation for minimizing energy consumption. *ACM SIGPLAN Notices* 37, 4 (2002), 42–53.

[86] Yikai Zhao, Kaicheng Yang, Zirui Liu, Tong Yang, Li Chen, Shiyi Liu, Naiqian Zheng, Ruixin Wang, Hanbo Wu, Yi Wang, and Nicholas Zhang. 2021. LightGuardian: A Full-Visibility, Lightweight, In-band Telemetry System Using Sketchlets. In *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*. USENIX Association, 991–1010. https://www.usenix.org/conference/nsdi21/presentation/zhao

[87] Haitao Steve Zhu, Chaoren Lin, and Yu David Liu. 2015. A programming model for sustainable software. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, Vol. 1. IEEE, 767–777.