SPECIAL ISSUE PAPER OPEN ACCESS

A Multi-Layered Analysis of Energy Consumption in Spark

Nestor D. O. Volpini[1,2] | Vinícius Dias[3] | Dorgival Guedes[2]

[1]Departamento de Eletroeletrônica e Computação, CEFET-MG, Belo Horizonte, Brazil | [2]Departamento de Ciência da Computação, UFMG, Belo Horizonte, Brazil | [3]Departamento de Ciência da Computação, UFLA, Lavras, Brazil

Correspondence: Nestor D. O. Volpini (nestor@cefetmg.br)

Received: 2 August 2025 | **Revised:** 2 December 2025 | **Accepted:** 8 January 2026

Keywords: big data | cloud allocation policies | energy consumption | experimental evaluation | monitored clouds | Spark | virtualized cloud

ABSTRACT

Although energy has become a major concern in data processing systems, it is usually hard to get a deep understanding of how performance and energy consumption relate to each other when planning how to configure a computing environment to execute a specific data-oriented workload. In this paper, we propose a multi-layered methodology to analyze the energy consumption of big data workloads executed using Apache Spark in virtualized cloud environments. The approach is structured into three layers: Resource provisioning, system-level resource utilization, and application-level resource utilization. Using direct energy measurements using a Power Distribution Unit (PDU) and detailed system monitoring, the study investigates how infrastructure choices and workload characteristics influence energy consumption. Results show that optimal virtual machine configurations depend on workload type and input size; while provisioning decisions affect energy consumption, system-level metrics such as CPU utilization and disk I/O offer a deeper understanding of the final performance versus energy consumption results. By applying our methodology, our results reveal the impact of task distribution and resource under-utilization on overall energy efficiency. The findings demonstrate that energy optimization in big data environments requires a comprehensive understanding of factors across infrastructure, system, and application layers. The proposed methodology serves as a practical guide for energy-aware design and decision-making in cloud-based data processing systems.

1 | Introduction

Considering the technological resources present in modern life, it is possible to observe a significant growth in data generation, which in turn demands processing to make it relevant. Sensor networks, personal wearable devices, and Internet of Things connected hardware are some examples of the high volume of data that has been generated. On the other hand, recommendation systems, machine learning algorithms, data mining, and other recent evolutions in artificial intelligence benefit from this data, making it relevant for some context. Cloud computing makes processing that data feasible and accessible, mainly by using resources with a cost proportional to their use, eliminating large initial investments for its users [1]. Cloud computing, by offering virtualized resources, reduces costs for both the provider and the user, making its use scalable and flexible, with high service availability. The use of virtualization to process large volumes of data is consolidated through the use of Virtual Machines (VMs), decoupling processing from the physical part, allowing the user to allocate a certain volume of resources (cores, memory, disks, etc.) in the way that suits them [2]. Nevertheless, to process such massive data sets, non-trivial big data infrastructures are often necessary, and they usually present non-trivial architectural tradeoffs that make it difficult to find

Abbreviations: HDFS, Hadoop File System; PDU, Power Distribution Unit; RAPL, Running Average Power Limit; RDD, Resilient Distributed Dataset; VM, virtual machine.

the best configurations for achieving the best cost-performance solutions.

The energy consumption demanded by that type of infrastructure is significant when processing large volumes of data. Its cost is relevant both for cloud providers and for the environment, as it often comes from an energy production matrix that is sustained by emitting worrying amounts of carbon into the atmosphere [3]. According to the IEA (International Energy Agency) [4], in 2022, datacenters consumed 1% of all energy produced worldwide, which is a considerable share, with a forecast of reaching 8% of the total by the end of the next decade [5]. It is important to highlight that properly measuring energy consumption requires a global view of computing processes, which is complex, especially for distributed systems [6]. As an example, data can be compressed to decrease network traffic volume, leading to lower consumption of its assets. However, that leads to an increase in energy consumed by processors when compressing and decompressing data, which needs to be considered. To determine if an intervention reduces energy consumption, it is necessary to measure the total energy consumed from end to end.

This work is built upon three main limitations of the literature of energy efficiency on big data workloads: (1) the lack of reliable and accurate energy monitoring approaches, since most works rely on partial or inaccurate measures (e.g., Intel RAPL [7]); (2) the fact that end-to-end energy consumption in real-world workloads can be intricate and non-trivial to characterize; and (3) the lack of an in-depth, interpretable methodology for studying energy consumption of representative big data workloads within modern and standard processing tools. Given such gaps, this work evaluated five big data algorithms in a small-scale, virtualized environment, using Apache Spark, a popular tool for distributed big data processing [8, 9]. We study each algorithm against three levels of load (light, standard, and heavy) and report two important measures: Runtime performance, translated into task completion time, and the total energy consumption for each task.

Our main contributions in this work are (*i*) a multi-layered methodology based on accurate PDU measurements for understanding the energy consumption behavior of big data workloads from system to application, and (*ii*) an instantiation of that methodology for some important big data algorithms covering many application domains and different scales in terms of data volume. For reproducibility of the results, all the artifacts, including code and data, are available in a public repository (https://github.com/dccspeed/pdu-spark-energy).

2 | Related Works

The work of [10] was one of the first to identify the challenges of controlling energy consumption in big data applications. It focused on the Apache Hadoop processing environment and considered the control to turn machines on and off in a cluster. Compared to Spark, Hadoop provides a simpler, more limited programming environment, based on the map-reduce abstractions. The simplicity of the model inspired others to study optimization models, like [11], and Works such as [12, 13], and [14] focused on scheduling map-reduced tasks to improve energy consumption. Our work considers Spark, which offers a much richer set of operators and internal optimization solutions.

Regarding the level where energy optimization is considered, some works considered saving energy by choosing an architecture [15] or by moving VMs to locations where there is renewable energy, as in GreenNebula [16], or even at the level of "computational grid federation" [17]. It is also possible to act on the datacenter topology and adapt it to consumption [18], and also find the best communication route [19]. On the other end of the spectrum, there are also studies that act on the voltage and frequency of processors (DVFS) [20, 21], or even consider adopting energy-saving hardware [22]. Our decision was to understand energy consumption at different levels of the system, but without changes to Spark/operating systems internals, to consider all possible system interactions.

Specifically, a paper by [21] proposes a scheduling framework that reduces energy consumption in Apache Spark clusters while maintaining application deadlines. By combining historical profiling of Spark tasks with Dynamic Voltage and Frequency Scaling (DVFS), the system predicts the energy and time requirements of incoming jobs and dynamically adjusts CPU frequencies per worker node. The paper reports 25%–40% energy savings, but the authors measured energy using Intel RAPL, which does not account for the total power consumption of the servers. As can be seen in Reference [7], the energy measured shows a high correlation with the wall plug measure; however, the loads used only stress the CPUs. Furthermore, the authors did not consider variations in cluster configuration to the extent presented here, where we measure the consumption of all the servers.

As previously mentioned, as a contribution, the option was made to measure end-to-end energy consumption to avoid biases that may arise from results that do not consider the system as a whole, as pointed out by [6], which proved the importance of the recommendations made by [23]. An interesting thesis by [24] compares RAPL and PDUs to highlight the discrepancy of measurements in cloud environments, demonstrating this effect through the presented data. More along that line, another study characterizes power measurement methods and shows that the accuracy of RAPL-based approaches is highly dependent on workload, with errors substantial enough to limit energy optimization when compared to external power measurements (such as PDU) [25]. In fact, it is shown that recent datacenters organized specifically to perform inference with Large Language Models (LLM) leverage rack-level PDUs to track power levels accurately and support energy optimization strategies on GPU servers [26]. Overall, compared to this work, the multi-layered nature of our results and the high-fidelity of PDU measurements set this study apart.

This paper is an extension of previously published work ([27]), largely expanded. Here, we improve our analysis with the definition of a multilayer methodology, with an extensive statistical analysis, and the inclusion of the dimension of workload level (adding light and heavy load configurations), where we consider a degree of flexibility on the amount of resources dedicated to each execution.

3 | Methodology

Inspired by those previous results, this work evaluated the determining factors for composing the execution time and energy consumption of big data scenarios, focusing on five popular massive data processing algorithms implemented using the Spark framework (Section 3.1). The evaluation was performed using a monitoring structure specifically designed to collect and integrate metrics at hardware, operating system, and application levels (Section 3.2). The collected execution data enabled a significantly deeper and more detailed analysis of application behavior related to energy consumption. (Sections 4, 5, and 6).

3.1 | Algorithms and Configurations

Five commonly used workloads in big data processing were selected for evaluation: Terasort, K-means, PageRank, Support Vector Machine (SVM), and Matrix Factorization. Together they cover a variety of input data models (graphs, matrices, vectors, text), algorithm domains (machine learning, information retrieval, data mining), and execution patterns (iterative, regular or irregular, coarse- or fine-grained tasks). Next, we present a detailed description of each algorithm.

1. Terasort [28]: An efficient benchmark implementation of a data sorting algorithm over the Spark RDD (*Resilient Distributed Datasets*) abstraction. TeraSort includes its own data generator, TeraGen, used to create the datasets used in each test, and an application to validate the execution results, TeraValidate. The interest in this application in our tests was due to its intensive use of shuffling, a communication pattern used in Spark to (re) distribute data among executing processors, which is one of the performance bottlenecks for this type of processing. The complexity of Terasort is dominated by the sorting phase, being on the order of $O(n \log(n))$, and can be significantly affected by the data partitioning in a distributed system. The load, with stable behavior, is frequently used to evaluate massive data processing environments.

2. K-means [29]: A widely used data mining clustering algorithm. Its performance is limited by the amount of main memory and CPU availability. It is iterative in nature, and each iteration is composed of two phases, where first the distances of each datapoint to each centroid are compared to identify the point's nearest centroid and to assign the point to the specific cluster. After that is done for all points, centroids are recomputed based on the elements assigned to the cluster, and the quality of the new clusters is assessed. Being a typically NP-hard problem, the convergence in the search for centroids is not guaranteed in polynomial time, so termination occurs when a predetermined threshold is reached, usually, after a certain number of iterations. In that case, the complexity of K-Means depends on the number of given points (N), the desired number of clusters (K), the number of dimensions (D) of the data, and the number of iterations (T) defined as the threshold. Based on that, the total complexity of K-Means is given by $O(NKDT)$.

3. PageRank [29]: Is an iterative algorithm that estimates the importance of nodes in a network (e.g., web pages). Originally developed by Larry Page and Sergey Brin to evaluate the importance of web pages, was initially used as part of Google's search engine. It can be used to evaluate the efficiency, performance, and scalability in cluster environments, helping to identify areas for optimization and compare performance across different configurations. Its complexity depends on calculating the importance of each node in a graph based on the importance of other nodes pointing to it. The time complexity for each iteration is $O(E)$, where E is the number of edges in the network/graph. The number of iterations required for PageRank to converge to a stable value varies. The number of iterations is independent of the number of nodes and edges, but in practice, it is usually set as a parameter, which can depend on the graph structure and desired precision.

4. Support Vector Machine (SVM) [30]: Is a supervised machine learning algorithm, primarily used for classification and regression tasks. The goal of SVM is to find the hyperplane that best separates the data, maximizing the distance between the closest points of each class. The inherent complexity of this algorithm comes from matrix inversion, reaching $O(n^3)$. Once the optimal hyperplane is found, new data points are classified based on which side of the hyperplane they fall.

5. Matrix Factorization (MatFact) [29]: Is an algorithm widely used in recommendation systems and data analysis, decomposing matrices into smaller factors that capture the underlying data structure. In a distributed processing context, matrix factorization is implemented to handle large datasets efficiently and scalably. The implementation used is available in the SparkBench benchmark, and it uses the *Alternating Least Squares* (ALS) algorithm. ALS is an iterative method that seeks the best decomposition of a matrix into two smaller factors, minimizing the quadratic error between the original and reconstructed matrix. At each iteration, it solves a least squares problem to update the factor matrices. The complexity per iteration is generally $O(MNK)$, where M and N are the dimensions of the original matrices and K is the number of factors, typically the dimensions of the decomposition. The number of iterations to converge varies depending on the desired precision and the characteristics of the dataset.

These algorithms are used as execution scenarios in this study. As a first step, the configuration parameters of each scenario were iteratively adjusted. During that calibration process, the best practices recommended by the Spark documentation were observed ([31]). For workloads where partitioning was not automatically configured, the number of partitions was determined according to the guidelines established in the benchmark's best practices. That defined a set of *standard* workloads, which gave us a baseline for each algorithm in terms of the volume of input data. Since data scientists do not always have all the information and time for such precise configurations, we decided to include two other scenarios, one considering that the configuration was used on a smaller dataset (hence resulting in a *ligher* workload) and another considering a dataset larger than the one used for the configuration process (hence a *heavier* workload). Both light and heavy workloads were executed under the same configuration parameters defined originally by the best practices for the standard workloads. The goal was to assess Spark behavior under misconfigured conditions, when compared to the standard. In general, input datasets for light and heavy loads were 50% smaller

TABLE 1 | Summary of algorithms and loads used in the experimental study. The output data sizes are dependent on the input sizes, but also depend on the specific data and the algorithms themselves.

Algorithm	Number of spark stages	Load	Input	Output	*Shuffle* Read	*Shuffle* write
Terasort	2	Light	14 GB	14 GB	7 GB	7 GB
		Standard	29 GB	29 GB	15 GB	16.5 GB
		Heavy	42 GB	42 GB	20.2 GB	24.7 GB
K-means	15	Light	9 GB	10 GB	2.2 MB	2.4 MB
		Standard	16 GB	19 GB	4 MB	4.3 MB
		Heavy	27 GB	32 GB	4.2 MB	5.5 MB
PageRank	98	Light	2.4 GB	34 MB	5.7 GB	6 GB
		Standard	2.6 GB	110 MB	7.8 GB	9.4 GB
		Heavy	6.3 GB	85.9 MB	14 GB	15.4 GB
Support Vector Machine (SVM)	28	Light	8.3 GB	15.2 MB	6.6 GB	7.4 GB
		Standard	16.7 GB	30.8 MB	13.2 GB	14.4 GB
		Heavy	25 GB	47 MB	17 GB	21.6 GB
Matrix Factorization (MatFact)	66	Light	1.4 GB	1.6 GB	2.5 GB	3.3 GB
		Standard	2.6 GB	2.8 GB	6.7 GB	7 GB
		Heavy	4.2 GB	5.4 GB	7.9 GB	9.8 GB

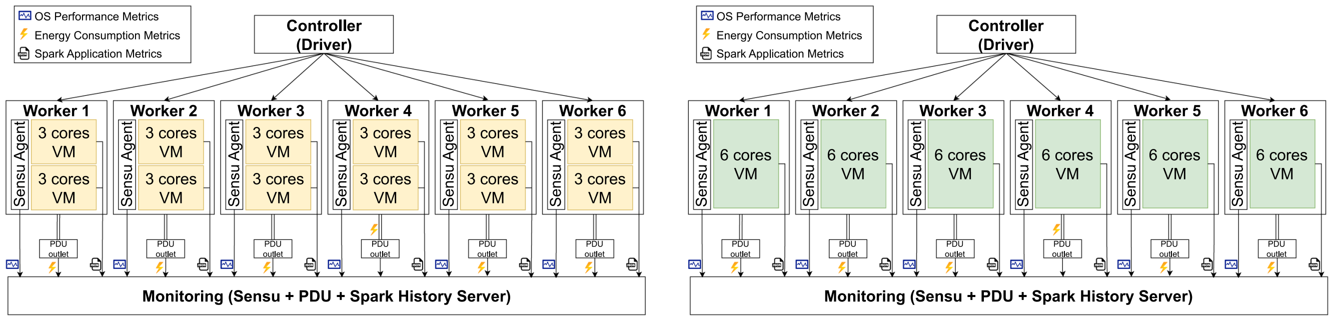

FIGURE 1 | Experimental setup. Left: 3 cores per VM (2 × 3). Right: 6 cores per VM (1 × 6).

or 50% larger than the standard load considered for parameter configuration. In a few cases, the larger dataset had to be trimmed down due to limitations of the execution environment. Table 1 summarizes the characteristics of the evaluated loads, including the number of stages in each and the volume of data effectively processed in critical tasks. The highlighted characteristics contributed to generating a diversity of behaviors capable of stressing the processing environment.

3.2 | Experimental Environment

Figure 1 illustrates the execution and monitoring environment used for our comprehensive performance and energy consumption analysis. The architecture comprises a centralized Controller (Driver), which orchestrates tasks across (up to) six individual Worker nodes. Each Worker is a physical server that hosts one or multiple VMs. Energy to all servers was provided through a Power Distribution Unit (PDU) capable of measuring energy consumption continuously during the experiments, to provide comprehensive energy numbers and not just that provided by CPU counters. A Sensu Agent is deployed on each Worker for collecting OS execution metrics. All Workers were configured to save data in their Spark History Servers to gather application run-time level metrics. The system incorporates a multi-layered System Monitoring infrastructure that integrates data from all Sensu Agents, the PDU outlets, and the Spark History Servers. This setup allows a detailed, end-to-end monitoring of system behavior, resource utilization, and energy consumption during big data processing tasks, enabling a comprehensive analysis of efficiency at different levels. Part of the physical machine resources was reserved for the hypervisors to monitor performance and consumption metrics, and another part was designated for application execution. This separation was made to minimize the impact of monitoring on the monitored executions.

One of the goals of this analysis was to consider the impact of different cluster configurations under the same budget restrictions: In cloud environments, users can not only choose how many workers they want to use, but also the internal configuration of each machine. For the same budget, it is possible, for example, to deploy many small workers or a few larger ones. To include some of that in our experiments, we maintained the same total number of physical Workers (six) and the same total number of cores allocated per Worker (six), but considered one case where one large VM was used per worker (with six cores in it), and another case where the six cores of each worker were divided into two smaller VMs with three cores each. During experiments, when using different VM sizes, we kept the total number of cores allocated equal to allow us to compare different configurations for basically the same resource budget. These two configurations allow a direct comparison of the impact of VM granularity (i.e., using more smaller VMs versus fewer larger VMs) on big data application performance, resource utilization, and energy consumption. This setup was crucial for understanding how the virtualized environment itself, and not just the raw number of cores, influences the efficiency of distributed computing workloads.

3.2.1 | Execution Environment

Applications were executed on Apache Spark v2.2.0, using the Hadoop/Hadoop File System (HDFS) v2.9.1 file system with replicas to offer minimal redundancy and provide agility in reading/writing from discs and the network. We adopted a replication factor of two to ensure that most input data were read locally, in the same VM as the computing tasks. Our cluster did not span multiple racks, so the use of three replicas to span more than one rack and still have at least two replicas in one rack was not necessary. In particular, a replication factor of two in our experiments was enough to guarantee that only 4.26% and 4.4% of Spark tasks, for heavy and light loads respectively, were not scheduled locally with data across all the experiments and configurations. The allocation unit size used was 128 MB. The resource manager used was Yarn, already provided with Spark. For provisioning and orchestrating the virtual machine environment, we used OpenStack Pike version on an Intel Core-i7 2600 3.4 GHz physical server with 16 GB of RAM. For the VMs, 6 Intel Xeon E5-2620v4 2.1 GHz servers were used, each with 8 cores (16 *threads*) and 32 GB of RAM, two Gigabit Ethernet network interfaces, and a 2 TB SATA disk, running on the QEMU-KVM 2.10.1 hypervisor over GNU/Linux Ubuntu 16.04, Kernel 4.4.0.

3.2.2 | Resource Provisioning Configuration

For the tests, not all processing cores nor all RAM in each server were made available, to establish a reserve for the hypervisor [32]. Concerning resource provisioning, from 3 to 6 physical servers were offered, adding one at a time, which corresponded to offering from 18 to 36 cores in increments of 6 cores at a time. From a consumption perspective, machines not taking part in a scenario were powered down so as not to affect the results. However, it is important to consider that every time a physical machine is added, there is a significant increase in the latent energy consumption P_0 even when the machine is under no processing load. Since consumption is given by the integration of power over time, the question to be answered was: Does the acceleration obtained for task completion justify the increase in total consumption? As previously mentioned, the tests performed included either one VM per physical server with 6 cores and 30 GB of memory (1×6), or two VMs with 3 cores on each physical server, each with 15 GB of RAM (2×3). This allowed us to evaluate the impact of using more (2×3) or fewer (1×6) VMs while maintaining the same total resource quantity. The resource provisioning at the application level in Spark was a one-to-one mapping between VMs and executors, that is, in one VM with 3 cores, we instantiate one single Spark executor also with 3 cores. The same reasoning is adopted for VMs with 6 cores. The parallelism level of Spark is set to the default (total number of cores on all executor nodes). Such a design choice mitigates experimental noise since one virtual core (VM and executor level) is always mapped to one physical core. Unless otherwise specified, all the other resource provisioning factors were left unchanged to enforce a realistic setting—including disk optimization among VMs, processor affinity, Spark scheduling policies, and other configurations.

3.2.3 | Monitoring Environment

To monitor active power, voltage, current, and the consumption of each server, we used the Raritan PX-2 Series 5000 PDU, with 1% accuracy (ISO/IEC 62053-21 standard). To record data on processor utilization, disk usage, memory occupation, machine loads, network traffic, and energy consumption, as well as Spark execution logs for capturing metrics at the application, Yarn, and HDFS levels, a container-based monitoring cluster [33] was set up, consisting of 3 Intel Core-i7 2600 3.4 GHz physical servers with 16 GB of RAM. All physical servers, including those of the workers where the executor VMs are launched, were independently connected to the PDU outlets. The active power consumed by the servers and assets was recorded along with infrastructure metrics. The monitoring environment used open-source tools: Collectd for interfacing with the PDU to collect energy, Sensu for monitoring the processor, memory, disk, and network. RabbitMQ was used for queue management. InfluxDB was used for time series storage, and Grafana for data visualization through dashboards. We estimated the energy consumed by each execution as $E = (\sum_{i=1}^{N} P_i)/N * (t/3600)$, that is, the energy consumed in watt-hours is given by the average power measured on equally spaced intervals, multiplied by the duration of the application.

Overhead due to monitoring (Table 2). Concerning the deployment with 3-core VMs (more VMs), we observe a median (50th percentile) relative increment of 2.71% and 1.54% to runtime and energy, respectively, when the monitoring is turned on (11.49% and 9.62% at the 90th percentile). Concerning the deployment with 6-core VMs (fewer VMs), under the same circumstances, a 5.67% increase for runtime and 5.82% increase for energy at the median (9.96% and 14.51% at the 90th percentile). Hence, our experimental evidence indicates that overall our monitoring overhead is negligible and lightweight across most configurations and scenarios considered.

3.2.4 | Experiments

The experiments conducted in this study are summarized in Table 3. For each experimental configuration, we collected evaluation metrics concerning execution time, resource utilization performance (CPU, memory, disk, etc.), energy consumption via PDU, and application-level metrics provided by the Spark stack and collected via History Server. The measurements presented throughout this study are average values of at least 10 replications, with their respective 95% confidence intervals. Also, before each execution, the caches were properly cleared to avoid biased results.

The experiments were designed with the following objectives: **[OBJ-1]** to define a better set of resources for each load, based on the measurements; **[OBJ-2]** to verify if the size of the VMs used to build the processing cluster, given a fixed budged, impacts the results; **[OBJ-3]** to propose models that explain execution times and energy consumption with a minimum set of factors necessary to characterize consumption behavior; and **[OBJ-4]** to comprehend the factors at the application level that correlate well with the behaviors identified using system-level performance metrics.

TABLE 2 | Overhead added by the monitoring solution.

	VM with 3 cores		VM with 6 cores	
Percentile	**Energy (%)**	**Runtime (%)**	**Energy (%)**	**Runtime (%)**
50th	2.71	1.54	1.25	0.58
75th	7.76	6.68	5.67	5.82
90th	11.49	9.62	9.96	8.41
max	27.68	29.13	14.76	14.51

TABLE 3 | Experiment design space.

Parameter	Parameter variation
Algorithm	Terasort, K-means, PageRank, SVM, MatFact
Load level	Standard (default input data), Light, Heavy
Servers	3, 4, 5, 6
VM config.*	3 cores per VM (3×6), 6 cores per VM (1×6)

Note: * in a VM with k cores is instantiated a Spark executor with k cores.

3.2.5 | Multi-Layered Analysis

To provide a comprehensive view of these experimental results, we propose in this work a multi-layered methodology for characterizing big data workloads with respect to energy consumption. The idea is to start with high-level aggregated measurements and gradually drill down into low-level measurements that are closer to the application. For such we define three layers for workload evaluation (Figure 2).

In the first layer concerning ***Resource Provisioning*** (concerning **[OBJ-1]** and **[OBJ-2]**), we characterize the optimal resource provisioning and energy saving potential of big data loads by considering only higher-level aggregated measures such as application execution time and end-to-end energy consumption. This first layer is independent of the computing framework or stack, since it depends solely on the resource provisioning from VMs and the underlying energy monitoring method via PDUs.

In the second layer concerning ***System-level Resource Utilization*** (concerning **[OBJ-3]**), we employ linear regression models to describe the impact of performance metrics collected from O.S. monitoring on energy consumption, and hence, we provide insights into how diverse energy consumption patterns can be and which general conclusions are possible within our evaluated workload. This second layer can also be applied to other computing environments as is, since it only depends on the energy monitoring via PDUs and the internal built-in OS performance metrics.

Finally, in the third layer concerning ***Application-level Resource Utilization*** (concerning **[OBJ-4]**), we discuss the impact of the inherent application characteristics on the energy consumption. The third layer is framework-dependent, and hence, to leverage the methodology to other computing stacks, one would need to first define and collect application-level data.

4 | Layer 1: Resource Provisioning

The results presented in this section represent a general overview of the executions. The graphs include 95% confidence intervals

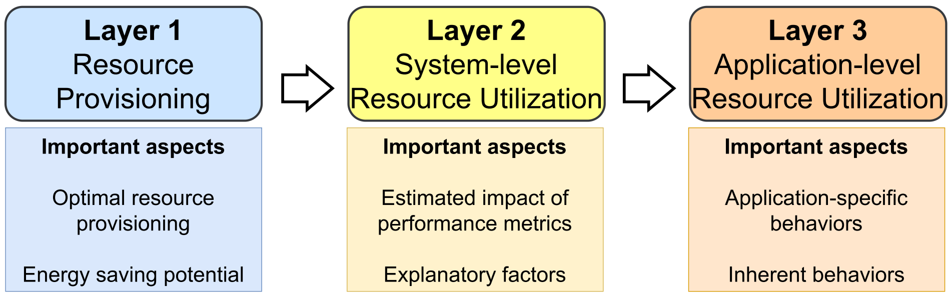
Layer 1
Resource Provisioning
Important aspects
Optimal resource provisioning
Energy saving potential
Layer 2
System-level Resource Utilization
Important aspects
Estimated impact of performance metrics
Explanatory factors
Layer 3
Application-level Resource Utilization
Important aspects
Application-specific behaviors
Inherent behaviors

FIGURE 2 | Multi-layered analysis overview.

from at least 10 executions, for two values: Execution time (seconds) on the left and energy consumption (watt-hours) on the right. In all loads, it is expected that as more resources are offered, the execution time will be reduced. However, as will be shown, energy consumption does not always suffer a proportional reduction. When adding a new server, it adds a power value P_0 that is present even under no application load, and it is added to the other factors that compose the consumption during execution time. All factors that contribute to the consumption depend on the time taken to complete the application. For this reason, the energy consumption of the scenarios behaves in various ways. The results obtained from processing each scenario are presented as two bar graphs, containing Execution Time and Energy consumption, showing the performance of workloads under a standard, light, and heavy load, respectively. All graphs plot metrics against "VM size" on the x-axis, in two categories: "2×3" (two small VMs per worker) and "1×6" (one large VM per worker). The number of servers used is presented as 3 in blue, 4 in orange, 5 in green, and 6 in red.

4.1 | Terasort (Figure 3)

For Terasort, the increase in the number of servers reduces execution time as would be expected, with better performance observed with 2 VMs of 3 cores. For each server included, the power (not included in the graphs) increases similarly for both VM sizes. Since Terasort's energy consumption is highly dependent on disk reads combined with CPU usage (see shuffle dimensions in Table 1), having more VMs contributed to the interposition of processing and I/O, providing concurrent HDFS accesses and accelerating executions. Regarding the variation in the number of servers, we observed that *despite the time acceleration obtained with more servers, a reduction in total energy consumption is not always achieved*—which leads us to conclude that optimizing consumption in big data applications is not trivial and often does not proportionally follow the gains obtained in time.

Comparing with light and heavy loads, we can note that Terasort's large workload requires more time and energy, as expected. Increasing the number of servers generally reduces execution time, but for light loads, this reduction can be marginal and even inefficient in terms of energy at certain points. The relationship between execution time and energy consumption is not linear or always proportional. In scenarios like the heavy load with 1×6 VMs in 4 servers, a shorter execution time does not mean proportionally lower energy consumption. This highlights the complexity of jointly optimizing performance and energy efficiency. The choice of VM size between 2×3 or 1×6 and the number of servers impacts performance and energy consumption differently depending on the load. For Terasort, the 2×3 VM size showed to be more consistently energy-efficient across all loads.

4.2 | K-Means (Figure 4)

Increasing the number of servers reduces execution time, but only up to a certain point. Unlike Terasort, K-means performance was higher with 1×6 VMs than with 2×3 VMs. However, performance does not consistently improve with an increased number of servers. For example, there is a minimum point in time and consumption observed with 5 servers in 6-core VMs. This behavior suggests uneven work distribution in these servers, an issue that is only exacerbated with more servers and can only be verified via application-level metrics, as we will see in Section 6.

Next, we contrast these results against light and heavy loads. For K-means with light loads, the 2×3 VM size generally shows increasing energy consumption with more servers (from 42 Wh at 3 servers to 56 Wh at 4 servers, then drop to 50 Wh at 5 servers and a slight drop to 48 Wh at 6 servers), while the 1×6 VM size is notably more energy-efficient, consistently lower (20 Wh for 3–5 servers, rising to 32 Wh at 6 servers), with its lowest consumption observed at 3, 4, or 5 servers. For K-means with heavy loads, the 2×3 VM size shows energy consumption increasing from 170 Wh with 3 servers to a peak of 200 Wh with 5 servers, before dropping back to 170 Wh with 6 servers. Conversely, the 1×6 VM size begins with high energy consumption (300 Wh at 3 servers), but significantly decreases to 125 Wh with 6 servers, eventually becoming lower than 2×3 at 6 servers.

Overall, energy consumption is lowest for light loads, moderate for standard loads, and highest for heavy loads. The 1×6 configuration is more energy-efficient for light loads, whereas for heavy loads, 1×6 demonstrates a substantial energy reduction with an increasing number of servers, indicating that more resources contribute to greater efficiency over time. For K-means, the 1×6 VM size generally offers better performance (lower execution time) and often better energy efficiency compared to 2×3 across all load types, especially as the number of servers increases for heavy loads. This contrasts with Terasort, where 2×3 was often a good performer for energy. The light and heavy loads corroborate that anomalies in intra-application and in data partitioning can lead to unpredictable energy consumption.

4.3 | PageRank (Figure 5)[1]

The 1×6 VM size consistently achieves lower execution times than 2×3 for PageRank under standard load, with the fastest execution observed at 5 servers for 1×6 (200s), while adding a 6th server leads to an increase in time for both VM sizes. For PageRank's standard load, the 1×6 VM size shows significantly lower energy consumption across all server counts compared to 2×3, with its lowest consumption at 3 servers (21 Wh), whereas 2×3 experiences an increase in energy as more servers are added, peaking at 55 Wh with 6 servers. Thus, for PageRank, reducing execution time by provisioning additional servers does not directly implies in proportional reduction in energy consumption.

When considering light load, the 2×3 VM size has the highest execution time with 3 servers, decreasing with 4 servers, then it rises with 5 servers and drops with 6 servers. The pattern is somewhat erratic, but still suggests fewer than 3 servers. In the case of the 1×6 VM size, execution time is lower than for 2×3 VMs, with the fastest suggested execution time with 3 servers. The confidence interval in the group of 1×6 VM does not allow strong affirmations for 4, 5, and 6 servers. For a heavy load is interesting note that for 3 servers in 2×3 VMs size runs faster than equivalent budget of servers with 1×6 VMs. Both configurations have their execution times shortened from 4 servers to 5 servers, and

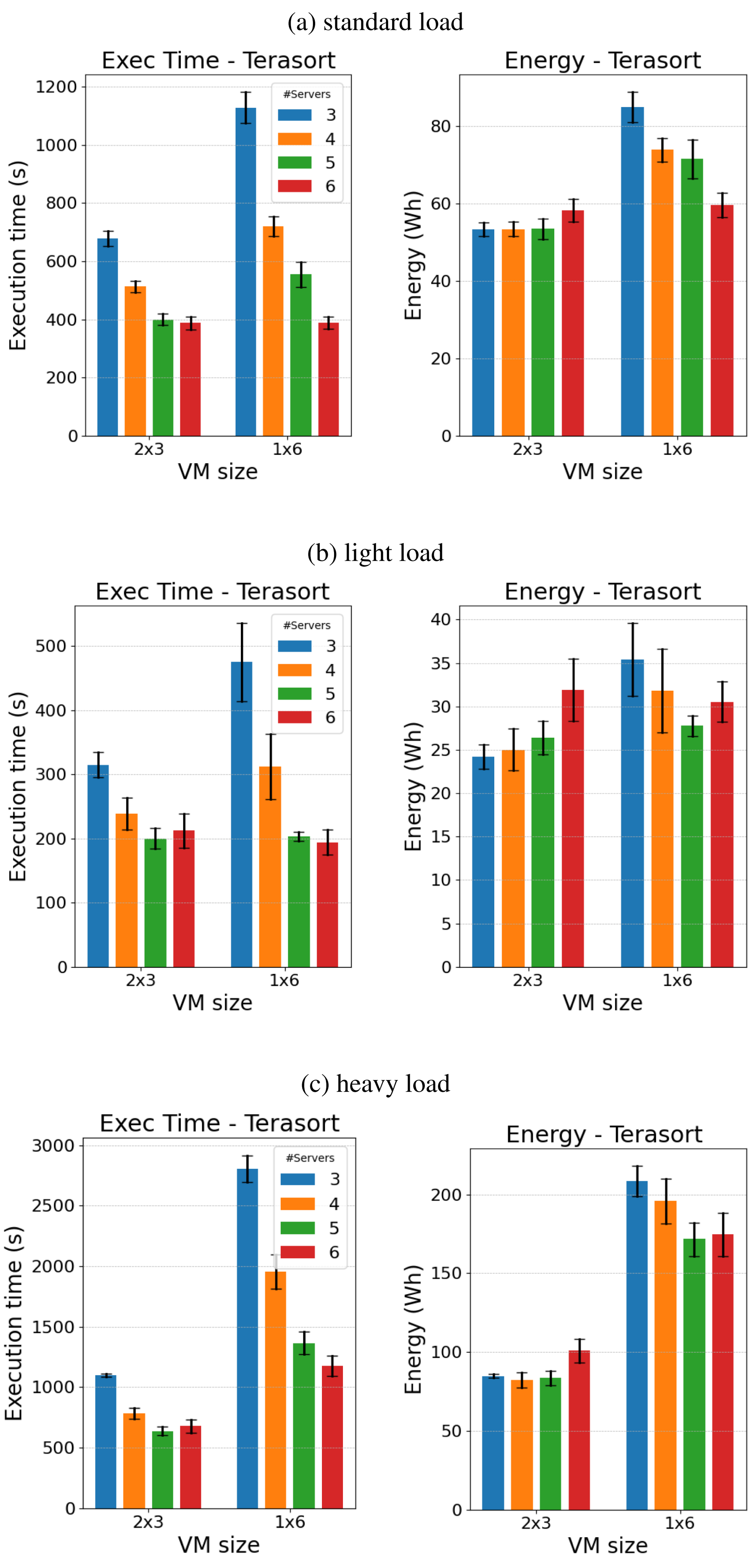

FIGURE 3 | Execution time and energy consumption of Terasort in standard, light, and heavy load.

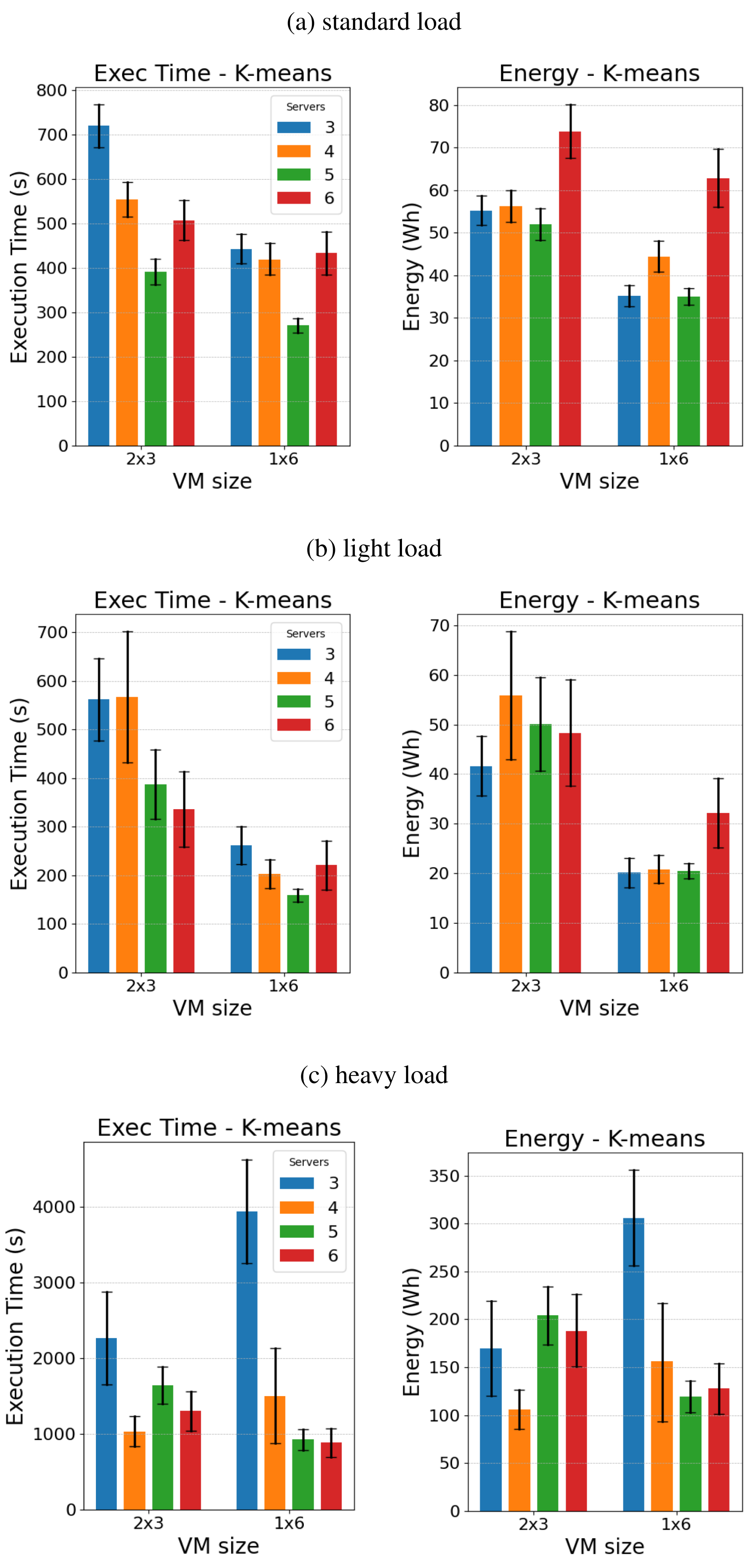

FIGURE 4 | Execution time and energy consumption of K-means in standard, light, and heavy load.

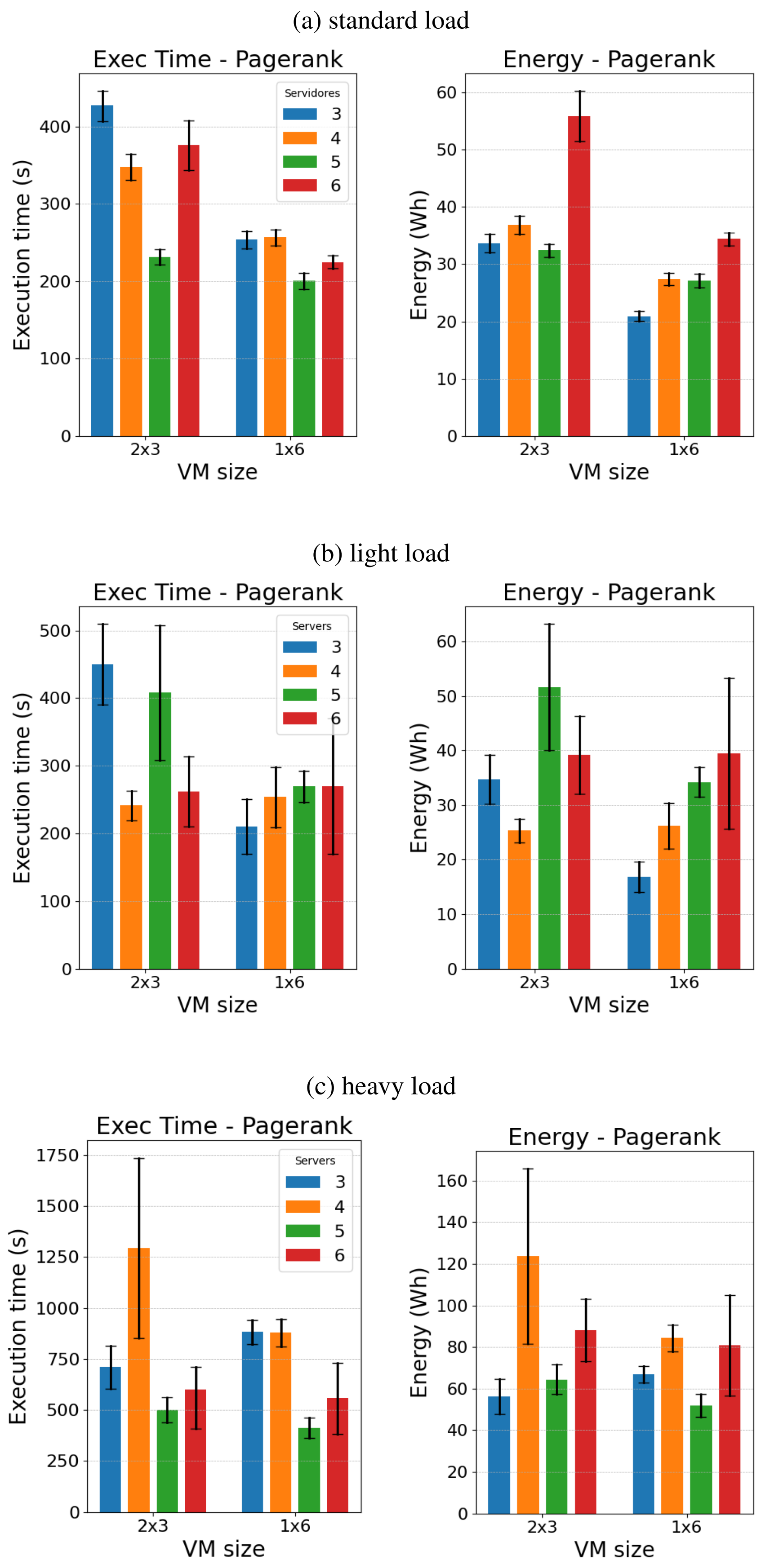

FIGURE 5 | Execution time and energy consumption of PageRank in standard, light, and heavy load (scenarios for 6 servers under heavy load did not execute).

it seems that 4 servers is the worst case for both VM sizes (hardware problems prevented us from executing the scenarios with 6 servers in this case). The overall trend for execution times is lowest for light loads, moderate for standard loads, and highest for heavy loads. For light loads, a 1×6 VMs configuration is faster. Considering the energy consumption under light load, the erratic pattern in the execution time impacts the results for the 2×3 VMs configuration. For the 1×6 configuration, the impact on total power of including servers is more significant than the acceleration in execution time. In other way for a heavy load of PageRank, 2×3 VMs suggests a minimum similar consumption for even 3, 4, or 5 servers. One interesting observation is that for a heavy load in 2×3 VMs and 1×6 VMs, unlike the light load, the reduction in execution times with 5 servers becomes significant enough to decrease energy consumption when adding more servers. Again is still clear that are complex time-energy relationship, since the graphs consistently show that reducing execution time doesn't always directly lead to proportional energy savings. The optimal configuration for speed might not be the most energy-efficient, and vice-versa.

4.4 | SVM (Figure 6)

For the standard SVM workload, the 1×6 VM size delivered superior performance with lower execution times and reduced energy consumption compared to 2×3 across all server configurations. While increasing the number of servers generally decreased execution time for both VM sizes, reaching optimal speeds at 5 or 6 servers for 1×6, the most energy-efficient point was observed at 4 servers in the 1×6 configuration. Conversely, the 2×3 VM size typically experienced increased energy consumption with more servers, underscoring a trade-off where greater resources do not always equate to energy savings, particularly for certain VM configurations and load types.

The SVM performance under light and heavy loads reveals distinct patterns for execution time and energy consumption. For light loads, the 1×6 VM size generally yields faster execution, achieving its quickest run at 5 servers, while the 2×3 configuration is slower and its execution time can increase with more servers. In terms of energy for light loads, 1×6 is markedly more efficient, reaching its lowest consumption with 3 to 5 servers, whereas 2×3 typically shows an increase in energy as server count rises. Under heavy loads, the 1×6 VM size initially experiences very high execution times and energy consumption with fewer servers, but significantly improves both metrics as more servers are added, becoming fastest and most energy-efficient at 5 servers. Conversely, the 2×3 configuration under heavy load starts faster with fewer servers but shows less significant improvements in time and higher, more stable energy consumption compared to the optimized 1×6 setup.

Comparing these results with the standard SVM load, all load types, the 1×6 VM size consistently outperforms 2×3 in terms of both lower execution times and generally superior energy efficiency. While light loads often exhibit diminishing returns or even performance degradation when adding too many servers beyond an optimal point, heavy loads consistently benefit from increased server allocation, where significant reductions in execution time directly translate into lower total energy consumption. This highlights that for demanding workloads, allocating more resources to the 1×6 VMs can lead to substantial efficiency gains that outweigh the initial overhead.

4.5 | Matrix Factorization (Figure 7)

In Matrix Factorization for standard load (Figure 7), in the 2×3 VM size, the execution time generally decreases as the number of servers increases from 3 to 5, but then rises notably with 6 servers. Energy consumption for 2×3 follows a less consistent pattern, starting at 3 servers, peaking at 6 servers, and with a slight dip at 5 servers. Conversely, the 1×6 VM size generally shows lower execution times, decreasing from 3 servers to 5 servers, then increasing slightly at 6 servers. Its energy consumption pattern for 1×6 is also lower overall, starting at 3 servers, increasing to 4 servers, then slightly decreasing to 5 servers before rising to 6 servers. Notably, the 1×6 VM size generally achieves faster execution and lower energy consumption compared to 2×3 for this load.

The Matrix Factorization (MatFact) performance under light and heavy loads reveals varied behaviors across different VM sizes and server counts. For light loads, the 2×3 VM size exhibits variable execution times, with a notable peak at 3 servers and a low at 5 servers, while its energy consumption is highest with 6 servers and lowest with 5 servers. The 1×6 VM configuration for light loads shows lower execution times, particularly at 4 and 5 servers, though it experiences an energy increase with 6 servers. Under heavy load, the 2×3 VM size demonstrates execution times that decrease with more servers, from 3 to 5 servers, and its energy consumption also decreases, reaching a lower consumption with 4 servers. Meanwhile, the 1×6 VM size under heavy load has significantly longer execution times with fewer servers (e.g., 3 servers), and its energy consumption generally increases with more servers, reaching a peak with 6 servers.

Comparing these with the standard load, several key patterns emerge. Across all load types, the 1×6 VM size achieves lower execution times than 2×3 for standard load, but under heavy load, 2×3 showed be faster with fewer servers. In terms of energy, the 1×6 VMs size is often more energy-efficient for light and standard loads, showing lower consumption at optimal server counts. However, for the heavy load, the 1×6 VMs size, despite its reduced execution time with more servers, paradoxically shows a significant increase in total energy consumed as more resources are added. This contrasts sharply with the 2×3 VM size under heavy load, which, while not always the fastest, often exhibits lower and sometimes decreasing energy consumption with additional servers, indicating that the most performant configuration is not always the most energy-efficient across different workloads and VM sizes.

4.6 | Discussion and Takeaways From Layer 1

Considering the results shown, some of those algorithms get benefits from increased servers up to a point, after which overhead becomes noticeable. Each algorithm demonstrated unique sensitivities to VM size and server count. For instance, PageRank consistently favored 1×6 VMs for both time and energy,

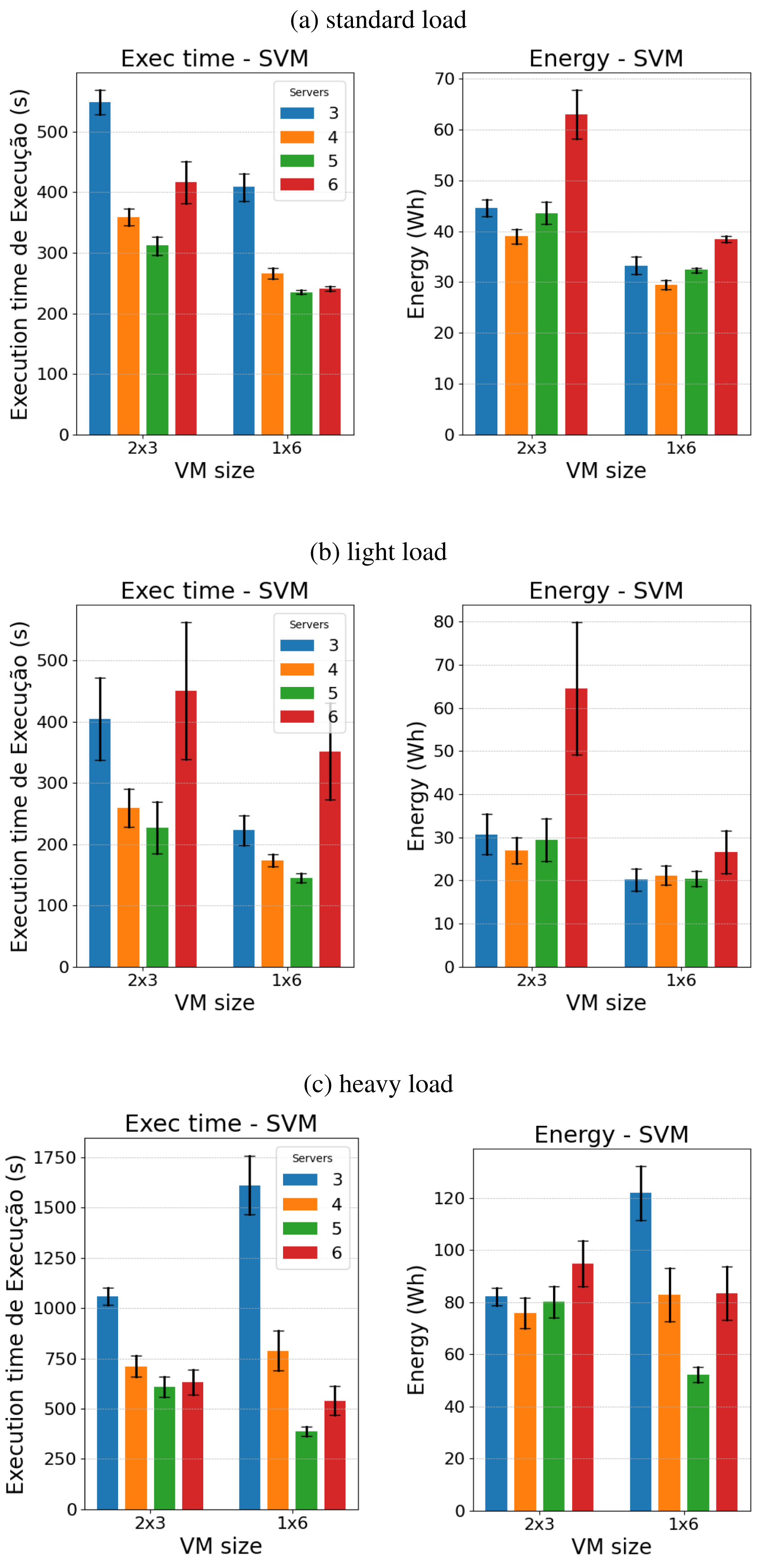

FIGURE 6 | Execution time and energy consumption of SVM in standard, light, and heavy load.

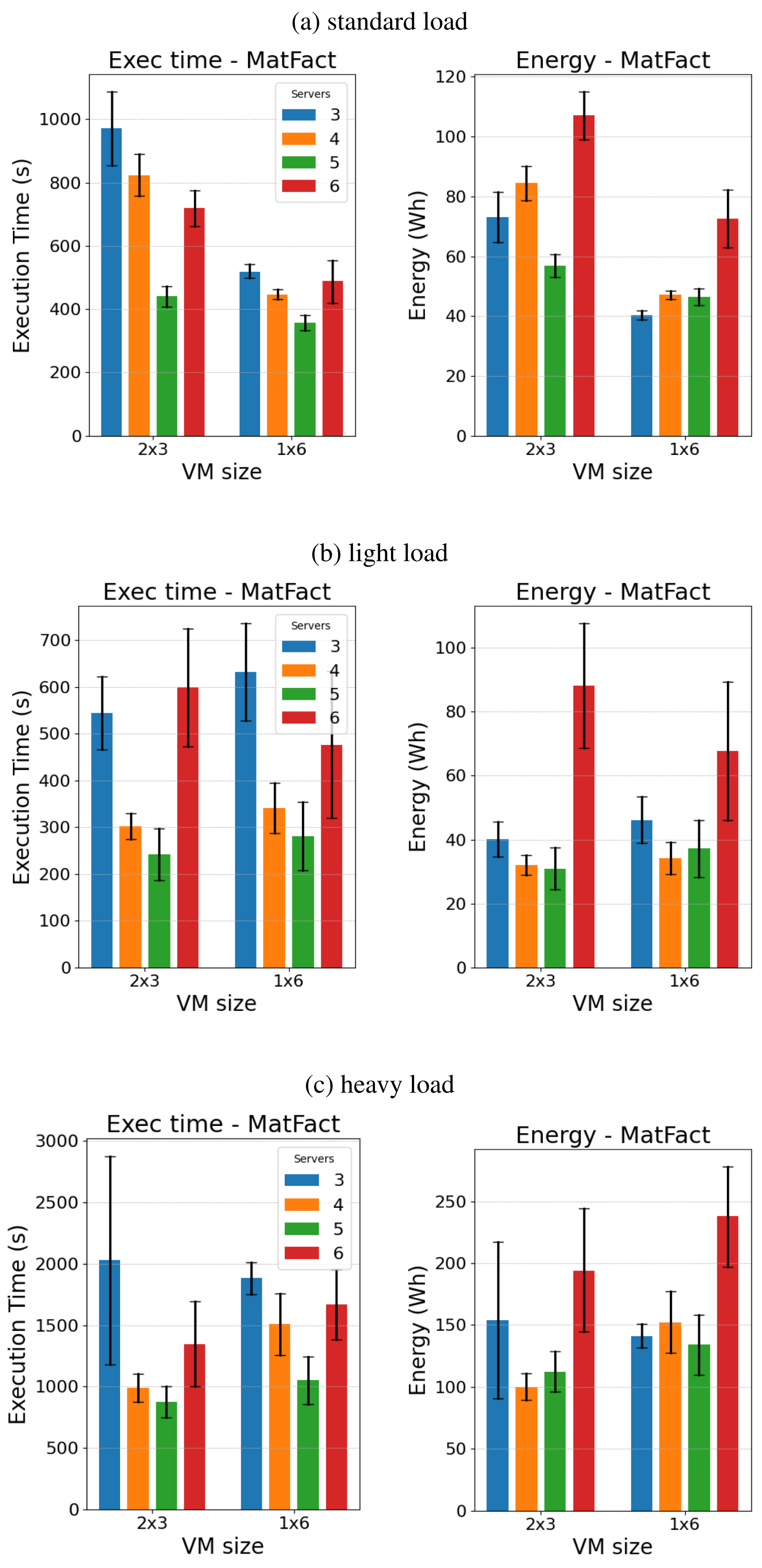

FIGURE 7 | Execution time and energy consumption of Matrix Factorization in standard, light, and heavy load.

while SVM showed dramatic energy efficiency gains in 1×6 VMs for heavy loads as servers increased. K-means also often preferred 1×6 VMs but could exhibit variable, erratic energy spikes. This indicates that a one-size-fits-all approach to resource allocation and energy optimization is unlikely to be effective. For light workloads, adding servers beyond an optimal point (often 4 or 5) can prove inefficient, as execution times may stagnate or even increase, which makes energy consumption rise due to overhead. Conversely, heavy workloads consistently benefit from an increased number of servers, particularly with the 1×6 VMs, leading to substantial reductions in execution time. In some instances, such as SVM with 1×6 VMs on heavy load, the time reduction also results in lower total energy consumption, demonstrating that a well-resourced setup is crucial for efficient handling of tasks. The behavior observed with standard loads typically falls between that of light and heavy loads, where certain algorithms show benefits from additional servers up to a specific count before overhead effects become noticeable.

The findings in this *Resource Provisioning* Layer demonstrate that there are setups capable of being more efficient in both time and consumption, but results clearly indicated that *the speedup obtained with more servers does not always translate into equivalent energy savings*. Also, VM size can influence consumption because it directly impacts how the parallel processing framework, in this case, Spark, sees the available parallelism. Another important takeaway is that the energy saving potential, that is, the difference between the lowest and the highest energy consumption for a given workload, varies substantially among applications and provisioning configurations. Such an observation indicates that *energy saving strategies in datacenters could benefit from tracking the behavior of the workloads and prioritizing the most prominent in terms of energy efficiency*, given that tuning applications for energy may be non-trivial, as we see in these results.

The variety of trends observed across the studied workloads led to questioning which factors could explain such behaviors. With each increase in server count, more disks are engaged in HDFS, enhancing parallel data processing that can accelerate read/write operations. Concurrently, the augmented number of network interfaces provides a larger capacity for network traffic, potentially speeding up inter-node communication and overall task completion. However, this expansion also carries the inherent challenge of increased management complexity and overhead, which might, in some instances, hinder performance gains or introduce unexpected delays in workload execution. Similarly, the presented experimental results do not provide sufficient data to evaluate average CPU utilization or which workloads benefit from or effectively occupy additional CPU resources. Consequently, it remains unclear at this point, for instance, which specific workload benefits most from the resources made available by adding a new server, or if the impact of resource allocation differs across the three load levels: Standard, light, and heavy. To a better understanding of what factors impact more or less in performance and consumption, in the following section, we go deeper into our analysis and study energy consumption with its relation to resource utilization metrics (CPU, memory, disk, and network) captured by monitoring.

5 | Layer 2: System-Level Resource Utilization

To determine how the collected metrics by monitoring impact the obtained results, multifactor linear regressions were performed for each algorithm in all load levels. At this point, it's important to differentiate between the two types of factors present in the regressions. There are factors (1) composed only of system metrics, *collected through monitoring*, and inherent to each execution, and there are factors (2) that the *user can control*. For the collected metrics, the following factors were considered: The average percentage of processor usage during executions (CPU-AVG), the sum of data written (D-WRITE) and read from disk (D-READ) in GB, and the sum of data transmitted over the network during execution (NET) in GB. The user-configurable factors considered in this study are the allocation of offered resources in different VM sizes (2 VMs with 3 cores each or 1 VM with 6 cores) and the number of servers (3 to 6). Here, we use linear regression as a method for estimating the impact of these factors, and hence, the purpose is not for prediction tasks.

5.1 | The Impact of Collected Performance Metrics

To characterize the impact of the collected metrics, factorial regressions that included just observed system factors, without differentiating the user-modifiable factors, are presented in Tables 4, 5, and 6. All models across all load levels exhibit very high R^2 values (ranging from 0.689 to 0.988), indicating that the selected metrics (CPU-AVG, D-READ, D-WRITE, NET) explain, not trivially, a large proportion of the variance in energy consumption. This suggests these factors are strong in explaining energy consumption in these big data workloads. The significance of factors in the majority of cases, p-values are lower than 0.001, indicating that the coefficients for CPU-AVG, D-READ, D-WRITE, and NET are statistically significant to compose energy consumption for most applications and loads. Notable exceptions where a p-value is higher, meaning less significance, can be interesting for deeper analysis, such as K-means D-READ at standard load, Matfact D-READ, and D-WRITE at standard load.

Focusing on specific metric and load levels, the β_1 (CPU-AVG) coefficient is negative across all applications and all load levels. This is a counterintuitive and interesting finding. Conventionally, higher CPU usage would be expected to lead to higher energy consumption. A negative coefficient suggests that, within these models, three possible things, or a combination of those: (i) increased CPU utilization is correlated with decreased total energy consumption. This could imply efficiency gains, if higher CPU utilization mean that tasks complete faster, thus reducing the total time for which power is drawn, leading to lower overall energy; (ii) it could be under-utilization overhead, when it might indicate that even when the CPU is less utilized, the base power consumption of the server (idle or low-power state) is still significant, and effectively utilizing the CPU leads to more work done for a given amount of active power; and (iii) workload-specific patterns, since negative correlation might also be due to the nature of the workloads where some CPU-intensive phases lead to quicker completion, or perhaps other components (disk,

TABLE 4 | Model and results of linear regressions per application at standard load.

Energy = $\beta_0 + \beta_1 \cdot$ CPU-AVG + $\beta_2 \cdot$ D-READ + $\beta_3 \cdot$ D-WRITE + $\beta_4 \cdot$ NET						
		β_0 Intercept	**β_1(%)**	**β_2(GB)**	**β_3(GB)**	**β_4(TB)**
Terasort	Coef.	74.90 ± 1.81	-0.39 ± 0.04	6.77 ± 1.10	-238.10 ± 48.48	0.926 ± 0.191
$R^2 = 0.818$	p	< 0.001	< 0.001	< 0.001	< 0.001	< 0.001
PageRank	Coef.	52.46 ± 3.03	-0.36 ± 0.03	-46.46 ± 5.79	318.53 ± 48.58	-0.289 ± 0.092
$R^2 = 0.835$	p	< 0.001	< 0.001	< 0.001	< 0.001	0.002
K-means	Coef.	60.62 ± 5.98	-0.65 ± 0.07	-2.39 ± 4.44	335.91 ± 48.05	-0.00120 ± 0.00011
$R^2 = 0.689$	p	< 0.001	< 0.001	0.591	< 0.001	< 0.001
SVM	Coef.	47.50 ± 0.87	-0.27 ± 0.01	-71.71 ± 5.12	1316.81 ± 69.58	-3.586 ± 0.153
$R^2 = 0.948$	p	< 0.001	< 0.001	< 0.001	< 0.001	< 0.001
Matfact	Coef.	101.70 ± 5.05	-1.08 ± 0.07	12.29 ± 9.36	-0.13 ± 0.09	0.386 ± 0.178
$R^2 = 0.804$	p	< 0.001	< 0.001	0.191	0.126	0.032

TABLE 5 | Model and results of linear regressions for each light load.

Energy = $\beta_0 + \beta_1 \cdot$ CPU-AVG + $\beta_2 \cdot$ D-READ + $\beta_3 \cdot$ D-WRITE + $\beta_4 \cdot$ NET						
		β_0 Intercept	**β_1(%)**	**β_2(GB)**	**β_3(GB)**	**β_4(TB)**
Terasort	Coef.	40.65 ± 1.78	-0.89 ± 0.06	17.09 ± 1.54	-10.58 ± 1.02	-0.0369 ± 0.0035
$R^2 = 0.939$	p	< 0.001	< 0.001	< 0.001	< 0.001	< 0.001
PageRank	Coef.	27.88 ± 4.31	-0.78 ± 0.14	211.13 ± 27.57	-143.74 ± 18.99	-0.432 ± 0.0562
$R^2 = 0.940$	p	< 0.001	< 0.001	< 0.001	< 0.001	< 0.001
K-means	Coef.	39.25 ± 2.30	-1.30 ± 0.08	54.94 ± 4.12	-32.82 ± 2.53	-0.145 ± 0.0113
$R^2 = 0.988$	p	< 0.001	< 0.001	< 0.001	< 0.001	< 0.001
SVM	Coef.	44.83 ± 2.71	-1.28 ± 0.06	-106.79 ± 15.10	68.10 ± 9.31	0.284 ± 0.0411
$R^2 = 0.979$	p	< 0.001	< 0.001	< 0.001	< 0.001	< 0.001
Matfact	Coef.	66.73 ± 4.39	-2.73 ± 0.18	-74.66 ± 33.12	55.03 ± 23.74	0.140 ± 0.0621
$R^2 = 0.914$	p	< 0.001	< 0.001	0.027	0.023	0.027

TABLE 6 | Model and results of linear regressions for each heavy load.

Energy = $\beta_0 + \beta_1 \cdot$ CPU-AVG + $\beta_2 \cdot$ D-READ + $\beta_3 \cdot$ D-WRITE + $\beta_4 \cdot$ NET						
		β_0 Intercept	**β_1(%)**	**β_2(GB)**	**β_3(GB)**	**β_4(TB)**
Terasort	Coef.	76.73 ± 19.52	-0.45 ± 0.42	-0.46 ± 0.52	9.00 ± 1.63	-0.0222 ± 0.0030
$R^2 = 0.945$	p	< 0.001	0.289	0.370	< 0.001	< 0.001
PageRank	Coef.	60.57 ± 16.71	-1.19 ± 0.45	-11.6 ± 3.84	19.39 ± 11.63	0.0011 ± 0.0012
$R^2 = 0.925$	p	< 0.001	< 0.001	0.002	0.001	0.690
K-means	Coef.	108.26 ± 8.75	-2.93 ± 0.36	3.07 ± 0.38	7.64 ± 0.58	-0.0397 ± 0.0042
$R^2 = 0.974$	p	< 0.001	< 0.001	< 0.001	< 0.001	< 0.001
SVM	Coef.	89.80 ± 7.71	-1.47 ± 0.22	3.39 ± 1.48	7.35 ± 1.95	-0.0434 ± 0.016
$R^2 = 0.908$	p	< 0.001	< 0.001	0.025	< 0.001	0.008
Matfact	Coef.	75.99 ± 10.59	-2.83 ± 0.36	-25.07 ± 1.60	103.92 ± 6.64	-0.145 ± 0.0095
$R^2 = 0.980$	p	< 0.001	< 0.001	< 0.001	< 0.001	< 0.001

network) become bottlenecks when CPU is less active, extending overall time and thus energy.

The magnitude of the negative coefficient often increases from light to heavy loads for some applications (e.g., K-means, SVM, Matfact), suggesting that the "efficiency gain" or "overhead reduction" from higher CPU utilization is even more pronounced for demanding tasks.

For the coefficient β_2 (D-READ), the sign of β_2 changes significantly across applications and load levels. For Terasort, β_2 has a positive coefficient at standard and light loads, but a negative one at heavy loads. PageRank has a consistently negative β_2. K-means has a negative β_2 at standard, but positive at light and heavy loads. SVM has a negative β_2 at standard and light, but positive at heavy. Matfact is highly variable, with a positive at standard, a very large negative coefficient at light, and a negative at heavy. This variability suggests that the energy cost of disk reads is highly dependent on the specific application's I/O patterns and the load level. A negative coefficient might imply that workloads that are bottlenecked by disk reads might spend more time overall, leading to higher total energy, while efficient, high-volume reads might be optimized.

The coefficient β_3 (D-WRITE) is generally positive for most applications and loads. D-WRITE has a positive coefficient, indicating that writing more data to disk consumes more energy. This is an intuitive result. Notable exceptions in magnitude are PageRank, which has a positive and relatively large β_3 in standard load, and Matfact shows a near-zero or slightly negative coefficient at standard load, but a very large positive coefficient at heavy load, suggesting a drastically different energy profile for disk writes under extreme conditions. At this point is important to emphasize that under heavy load, MatFact lost and recovered executors to complete required tasks.

For coefficients β_4 (NET) for network traffic are generally much smaller in magnitude compared to D-READ or D-WRITE, and their signs vary. Terasort has positive β_4 at standard and light, but negative at heavy. PageRank is negative at standard and light, but near zero at heavy. K-means and SVM are consistently negative. Matfact has mixed small values. This mixed behavior suggests that network traffic's direct energy impact might be less dominant than CPU or disk I/O, or that high network traffic is often correlated with efficient data movement that reduces overall execution time. A negative coefficient could, similar to CPU, imply that efficient network usage leads to faster completion, reducing total energy.

5.2 | Discussion and Takeaways From Layer 2 — Performance Metrics

In this *System-level Resource Utilization* Layer, concerning performance metrics, linear models showed that a consistently negative coefficient for CPU-AVG is a strong indicator that optimizing for CPU utilization, thereby reducing overall execution time, is a key strategy for energy saving in these types of workloads. Even if the instantaneous power draw is higher, the shorter duration of the task leads to lower total energy. Furthermore, the highly variable coefficients for D-READ and D-WRITE underscore that the energy cost of disk I/O is not universal; it's intricately linked to the application's specific data access patterns and how those patterns interact with the underlying storage system under different loads. For the load level on factor impact is possible observe that the change in magnitude and even sign of some coefficients across light, standard, and heavy loads indicates that the relative importance and energy implications of different resource metrics are not static, but depend on the intensity of the workload. This is crucial for dynamic resource management. Therefore, aside from CPU-AVG, we observe no overall pattern for factors' coefficients across applications and load levels, neither in terms of magnitude nor sign.

5.3 | The Incremental Impact of User-Defined Configurations and Collected Performance Metrics

Next, we evaluated the incremental impact of factors on the model through a hierarchical linear regression. The results are presented in the heatmap in Figure 8, which illustrates the

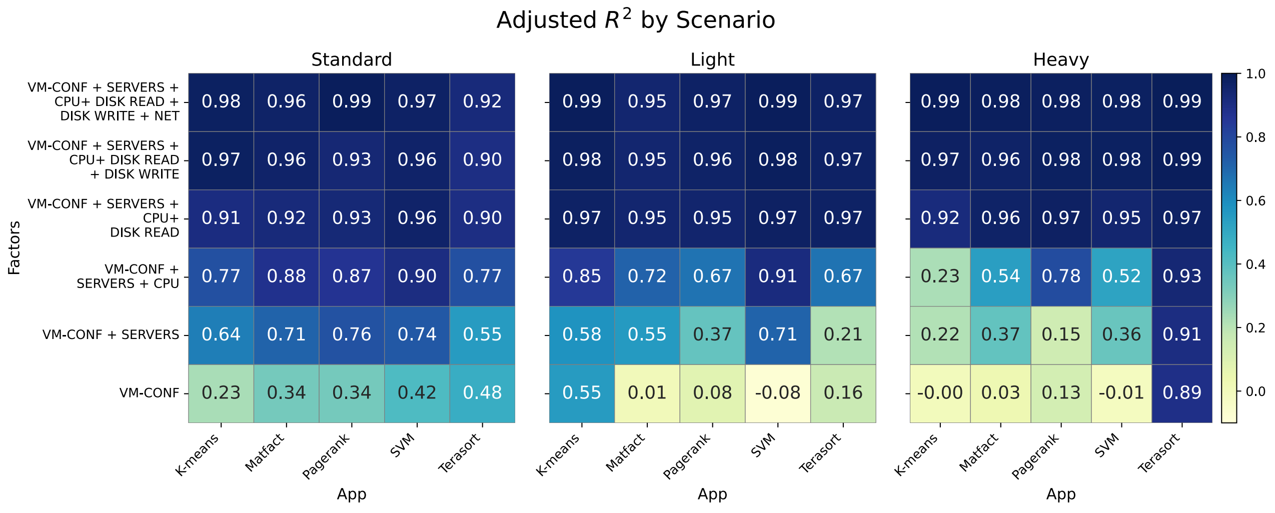

FIGURE 8 | Hierarchical linear regression: Understanding the incremental explainability of factors.

Adjusted R^2 values for hierarchical linear regressions modeling energy consumption across different applications (K-means, Matfact, PageRank, SVM, Terasort) and load scenarios (Light, Standard, Heavy). The analysis incrementally adds factors to the model, beginning with user-available parameters and then incorporating system-level metrics. The starting point for user intervention involves VM configuration (VM-CONF) and the number of servers (SERVERS).

Under standard loads, VM-CONF alone provides slightly better, yet still insufficient, explanatory power (0.23 to 0.48). However, the combination of VM-CONF + SERVERS yields a significantly stronger baseline (R^2 from 0.55 to 0.76), demonstrating that these user-controlled parameters become much more effective factors in energy consumption. Further incorporation of CPU, DISK READ, DISK WRITE, and NET consistently pushes R^2 values into the over 0.90 (0.92–0.98), completing their comprehensive role. For light loads, VM-CONF alone has a very limited impact (R^2 ranging from −0.08 to 0.55). Adding the number of servers improves R^2 notably for SVM (0.71), K-means (0.58), and Matfact (0.55), but still falls short for PageRank (0.37) and SVM (0.21). Subsequent inclusion of CPU, DISK READ, and DISK WRITE factors dramatically increases R^2 across all applications, frequently exceeding 0.95. The final addition of NET consistently brings R^2 to very high levels (0.95–0.99), indicating these system metrics are crucial for explaining energy at light loads. For heavy loads, VM-CONF alone is almost entirely ineffective as a factor (R^2 often near zero or negative), with Terasort being a significant exception (0.89). Even VM-CONF + SERVERS provides very low R^2 values for most applications (0.15 to 0.37), highlighting that for demanding tasks, simply configuring VMs and server count offers little insight into energy consumption without internal system metrics. It's the addition of CPU and D-READ that causes a dramatic surge in R^2 for all algorithms (all reaching around 0.92 to 0.97), demonstrating its critical role in explaining energy under heavy pressure. With the full set of factors, including DISK READ, DISK WRITE, and NET, R^2 values consistently reach near-perfect levels (0.98–0.99).

5.4 | Discussion and Takeaways From Layer 2—Incremental Impact of Factors

In the *System-level Resource Utilization* Layer, concerning the incremental impact of factors, we observe a clear hierarchy of explanatory power: User-controlled parameters (VM-CONF, SERVERS) offer a foundational level of explanation, which becomes more robust at standard loads than at light or heavy loads. *Such an observation leads us to conclude that user-defined factors (number of servers and VM configuration) are able to explain energy behavior up to a certain point. To have a truly reliable understanding of energy efficiency, one must account for monitored performance metrics that only can only be extracted at runtime*. This reinforces the idea that energy consumption is heavily impacted by load level—which can be dependent on the application (see Section 6). For a comprehensive understanding and accurate modeling of energy consumption, particularly at the extremes of workload intensity, detailed system-level metrics (CPU, disk I/O, network) are indispensable, consistently leading to extremely high R^2 values across all applications and scenarios. Notably, CPU utilization emerges as a particularly critical factor for explaining energy behavior, especially under heavy loads. Notice that our methodology at this layer does not include application-level metrics, and the results shown here enforce the idea that some energy consumption behaviors can indeed be explained by system-level observations, and moreover, that system-level metrics can supersede some application-level metrics that are more intricate to obtain. In our experiments, this is the case of metrics that track the Garbage Collection (GC) overhead in JVM-based systems such as Spark. In preliminary analysis considering GC as a factor to explain energy consumption (not shown), we concluded that such a factor did not provide significant improvement to the regression's R^2 in any scenario and hence, did not provide any *new* information that had not been captured by system-level metrics.

6 | Layer 3: Application-Level Resource Utilization

Even if the most important factors for a load's energy consumption are understood, it is also important to identify the specific execution characteristics that led to the observed behavior. Moreover, two application-level aspects may be too context-specific and exhibit an inherent energy consumption behavior. The first aspect concerns characteristics of the execution environment and model, in our case, algorithms evaluated are subject to the parallelism approach of Spark, which may or may not be optimized. The second aspect concerns characteristics and data-dependency patterns specific to each algorithm. We argue that the impact of these aspects can be observed via an application-level resource utilization metric designed to estimate *how effectively an execution timeline is filled with useful work—in our context, Spark tasks*. Hence, we evaluated the impact of Spark's task allocation on energy consumption. On a Spark server with the capacity to execute T_{max} Spark tasks in parallel (i.e., a server with T_{max} available cores per Spark executor), the maximum possible core utilization for an execution that lasted D seconds would be given by $T_{max} * D$ (all cores executed tasks during all the time). With this, we define the Spark Core Utilization metric SCU of a server as the fraction of that maximum (given by $T_{max} * D$) actually filled with Spark tasks ($0 \leq SCU \leq 1$). Table 7 shows the SCU for the VM configurations that exhibited the most energy-efficient results for each load level.

Concerning the standard load, when the *Spark Core Utilization* is contrasted with the energy consumption (Layer 1), it is clear that higher energy consumption is generally associated with a suboptimal utilization of available resources by tasks (lower SCU). In this case, it is possible to identify, beyond a certain point, loads that do not benefit from more servers, which, when active, unnecessarily increase energy consumption. Furthermore, the imbalance in resource utilization by servers plays an important role. Taking the Matrix Factorization algorithm with standard load on 6 servers as an example, one can see that some servers are very well utilized (Max $SCU = 0.81$), while most other servers remain practically idle (Min. $SCU = 0.13$ and Avg. $SCU = 0.28$). PageRank and K-means show progressive imbalance in utilization, similar to Matrix Factorization. At the other extreme, we can observe that Terasort and SVM had the lowest degrees of imbalance in SCU. Indeed, loads with more regular and dense processing (Terasort and SVM) allowed for better utilization compared to

TABLE 7 | *SCU (Spark Core Utilization):* Core utilization as the number of servers increases, for the best VM configuration identified for each load (scenarios for PageRank on 6 servers under heavy load did not execute).

		VM	3 servers			4 servers			5 servers			6 servers		
		Config.	min.	avg.	max.	min.	avg.	max.	min.	avg.	max.	min.	avg.	max.
Standard load	Terasort	2x3	0.87	0.87	0.88	0.77	0.80	0.86	0.68	0.72	0.79	0.63	0.67	0.78
	Kmeans	1x6	0.36	0.55	0.69	0.49	0.58	0.69	0.34	0.47	0.70	0.38	0.44	0.51
	Pagerank	1x6	0.53	0.62	0.79	0.36	0.47	0.65	0.40	0.48	0.56	0.28	0.35	0.63
	SVM	1x6	0.67	0.77	0.83	0.68	0.71	0.76	0.60	0.68	0.74	0.57	0.63	0.70
	MatFact	1x6	0.40	0.56	0.87	0.30	0.45	0.86	0.29	0.42	0.82	0.13	0.28	0.81
Light load	Terasort	2x3	0.64	0.70	0.78	0.54	0.62	0.73	0.58	0.62	0.70	0.38	0.47	0.62
	Kmeans	1x6	0.44	0.54	0.65	0.34	0.45	0.64	0.27	0.41	0.64	0.27	0.35	0.57
	Pagerank	1x6	0.25	0.51	0.76	0.19	0.44	0.63	0.40	0.45	0.50	0.21	0.37	0.72
	SVM	1x6	0.60	0.64	0.69	0.47	0.53	0.63	0.43	0.50	0.68	0.38	0.46	0.62
	MatFact	2x3	0.12	0.37	0.85	0.15	0.30	0.72	0.07	0.16	0.57	0.06	0.16	0.57
Heavy load	Terasort	2x3	0.90	0.91	0.93	0.73	0.76	0.86	0.77	0.79	0.85	0.38	0.47	0.62
	Kmeans	1x6	0.73	0.78	0.88	0.47	0.53	0.61	0.30	0.43	0.72	0.32	0.44	0.76
	Pagerank	1x6	0.66	0.71	0.79	0.65	0.67	0.68	0.59	0.65	0.68	0.44	0.50	0.61
	SVM	1x6	0.90	0.92	0.94	0.80	0.82	0.85	0.59	0.65	0.80	0.62	0.67	0.77
	MatFact	2x3	0.28	0.38	0.52	0.31	0.42	0.71	0.27	0.40	0.82	0.14	0.29	0.51

more sparse and irregular processing, typical of loads that process graphs and spatial data, such as PageRank and K-means.

Concerning the resource utilization under light loads, SCU values are consistently lower than in the standard or heavy scenarios. This is particularly pronounced for Matrix Factorization, which shows critical underutilization on six servers (Min. SCU = 0.06, avg. SCU = 0.16), and for PageRank, where SCU remains low even as the number of servers increases (e.g., avg. SCU = 0.37 on 6 servers). These results indicate that light loads tend to aggravate the inefficiencies of task distribution, especially in irregular or graph-oriented applications like PageRank and K-means. Even Terasort and SVM, which are more regular algorithms, exhibit moderate declines in SCU under light loads as the server count increases. Such a trend can be observed in Figure 9.

Concerning the behavior under heavy loads reveals consistently higher and more balanced resource utilization. Terasort and SVM stand out with SCU values remaining high across all server counts (e.g., Terasort avg. SCU = 0.91 with 3 servers, and still 0.47 with 6 servers; SVM avg. SCU = 0.92 and 0.67, with 3 and 6 servers, respectively). K-means also benefits from heavier workloads, achieving significantly better utilization than under light loads (e.g., avg. SCU = 0.78 with 3 servers, compared to 0.44 in Table 6). PageRank likewise shows improved balance, considering the data points available.

These comparisons reinforce key observations and are shown in Figure 9: Light loads tend to amplify imbalance, particularly in irregular workloads, making additional servers less effective and often wasteful. In contrast, heavy loads enable better parallelization, leading to higher SCU values across the board, especially in workloads with dense and regular processing patterns. However, the degradation in SCU as the number of servers increases is highly workload-dependent: While Terasort and SVM maintain robust utilization even at larger scales, Matrix Factorization, K-means, and PageRank exhibit increasing imbalance and diminishing returns, particularly when the workload is insufficient to saturate the available computational resources.

6.1 | Discussion and Takeaways From Layer 3

In the *Application-level Resource Utilization* layer, new perspectives on energy consumption for big data workloads are introduced via the *SCU* metric. First, our methodology confirms that, overall, it is challenging to prevent resource idleness in distributed execution engines as the number of servers increases. This becomes more evident when we observe lower and more skewed SCU values as the number of servers increases. This trend can be visualized in our scenario in Figure 9. Second, we identify inherent application behaviors in which the suboptimal utilization comes from algorithm-specific data dependencies, and in this case, the amount of work or the number of servers does not properly mitigate such an imbalance. This application-level analysis suggests that, for some cases, algorithm redesign or execution environment change may be the only effective alternative for saving energy.

7 | Conclusion and Future Work

This work proposed and applied a **multi-layered methodology** to analyze the energy consumption behavior of big data

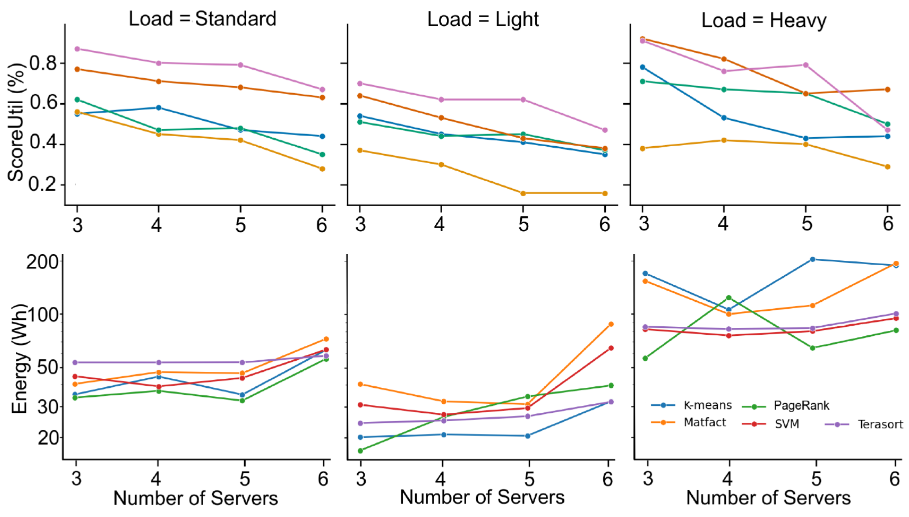

FIGURE 9 | Trends in application-level resource utilization: Comparing Spark Core Utilization (SCU) versus Energy Consumption for the configurations in Table 7.

workloads executed on Apache Spark in virtualized cloud environments. By decomposing the analysis into three layers—Resource Provisioning, System-Level Resource Utilization, and Application-Level Resource Utilization—we offered a structured methodology that provides a deeper understanding of the underlying factors influencing energy consumption.

At the first layer, resource provisioning, we observed that while increasing the number of servers often reduces execution time, it does not necessarily lead to proportional energy savings. In some cases, especially with light workloads, additional resources introduced unnecessary overhead, resulting in higher energy usage. The choice of VM configuration (2×3 vs. 1×6) also demonstrated importance: While 2×3 (more small VMs) performed better for workloads like Terasort, 1×6 (fewer larger VMs) configurations were generally more energy-efficient for SVM, K-means, and PageRank, especially under heavy loads. These observations reinforce the notion that there is no universally optimal configuration, and workload-specific resource provisioning is challenging, but can be achieved.

The second layer of analysis, based on monitored system-level performance metrics (CPU, disk I/O, and network), revealed increased ability to explain energy behavior. The consistently negative correlation between CPU utilization and total energy consumption suggests that higher CPU use can result in shorter execution durations and thus lower energy use. Disk and network metrics exhibited application-specific behaviors, indicating that a generalized interpretation is insufficient, and detailed analysis is necessary. Moreover, the hierarchical regressions performed show that while user-defined parameters (like server count and VM size) offer some explanatory capability, it is the inclusion of system-level runtime metrics that provides a more accurate understanding of energy consumption (i.e., the user may not be able to select the configuration with better performance-energy consumption just by setting server/VM level parameters).

At the application layer, the proposed SCU metric (Spark Core Utilization) proved valuable in assessing how effectively each workload utilized available processing capacity. We found that workloads with sparse or irregular execution patterns (such as PageRank and Matrix Factorization) often suffered from imbalanced task distribution, leading to under-utilization of resources and increased energy usage. In contrast, more regular workloads (like Terasort and SVM) tended to achieve higher and more balanced utilization across servers. These insights reveal how execution dynamics at the task scheduling level contribute directly to energy efficiency, an important consideration for future scheduling and orchestration strategies.

The objective of the present work was to provide explanatory insights and practical guidance for system designers and cloud architects seeking to balance performance and energy efficiency. Overall, this study highlights that optimizing energy consumption in big data environments requires a multi-dimensional understanding that goes beyond raw infrastructure metrics. The multi-layered methodology proposed here is one of the main contributions of this work, enabling a structured and interpretable approach to analyze how different factors across provisioning, system, and application layers impact energy consumption behavior.

For future work, first, we plan to consider new workloads (algorithms) in the same environment using the same methodology to strengthen its usability. Second, we find it important to extend the developed methodology by applying it to deep learning workloads that heavily utilize GPUs, such as LLMs. Last, a crucial step involves developing a lightweight and modular system designed to ensure the transparent integration of PDU power monitoring data with the existing performance monitoring stack, such as the *Spark's History Server*.

Acknowledgments

This work was partially funded by Fapemig, CAPES, CNPq, and the projects MCT/CNPq-InWeb (573871/2008-6), FAPEMIG-PRONEX-MASWeb (APQ-01400-14), H2020-EUBR-2015 EUBra-BIGSEA, H2020-EUBR-2017 Atmosphere, and FAPEMIG Universal APQ-00202-24. The authors confirm that Artificial Intelligence (AI) was used

exclusively as an accessory and tool to enhance the presentation and efficiency of this work. Specifically, AI-powered writing assistance tools, including GPT, Gemini, Copilot, and Perplexity, were used solely to refine the clarity, flow, and grammatical correctness of the original text in English, acting as a professional copyeditor to ensure a high standard of written communication. Furthermore, those AI tools were employed to correct and optimize the scripts and code used for the composition and rendering of graphs and visual elements within the article, significantly accelerating the preparation of visual data. No generative AI was used for the conception, research, data analysis, interpretation of results, or drawing of conclusions, which remain the exclusive and original intellectual contribution of the human authors.

Funding

This work was supported by the Fundação de Amparo à Pesquisa do Estado de Minas Gerais (Grant Nos. APQ-01400-14 and APQ-00202-24), Conselho Nacional de Desenvolvimento Científico e Tecnológico (Grant No. 573871/2008-6), Coordenação de Aperfeiçoamento de Pessoal de Nível Superior, EUBra-BIGSEA, and H2020-EUBR-2017 Atmosphere.

Conflicts of Interest

The authors declare no conflicts of interest.

Data Availability Statement

The data that support the findings of this study are openly available in pdu-spark-energy at https://github.com/dccspeed/pdu-spark-energy.

Endnotes

[1]When working on the PageRank algorithm, we ran into hardware problems that prevented us from getting execution results for the heavy load scenarios using 6 servers. For that reason, the analysis in this section does not include data points of time and energy for PageRank under heavy load with 6 servers.

References

1. M. Armbrust, A. Fox, R. Griffith, et al., “A View of Cloud Computing,” *Communications of the ACM* 53, no. 4 (2010): 50–58, https://doi.org/10.1145/1721654.1721672.

2. M. D. Assunção, R. N. Calheiros, S. Bianchi, M. A. Netto, and R. Buyya, “Big Data Computing and Clouds: Trends and Future Directions,” *Journal of Parallel and Distributed Computing* 79 (2015): 3–15, https://doi.org/10.1016/j.jpdc.2014.08.003. special Issue on Scalable Systems for Big Data Management and Analytics.

3. J. Whitney and P. Delforge, “Scaling up Energy Efficiency Across the Data Center Industry: Evaluating Key Drivers and Barriers,” (2014), https://www.nrdc.org/energy/files/data-center-efficiency-assessment-IP.pdf.

4. IEA, “Data Centres and Data Transmission Networks,” (2022), https://www.iea.org/reports/data-centres-and-data-transmission-networks.

5. M. Pesce, “Cloud Computing’s Coming Energy Crisis-the Cloud’s Electricity Needs Are Growing Unsustainably,” *IEEE Spectrum* (2021).

6. V. Anand, Z. Xie, M. Stolet, et al., “The Odd One Out: Energy Is Not Like Other Metrics,” in *HotCarbon 2022: 1st Workshop on Sustainable Computer Systems Design and Implementation* (2022).

7. K. N. Khan, M. Hirki, T. Niemi, J. K. Nurminen, and Z. Ou, “Rapl in Action: Experiences in Using Rapl for Power Measurements,” *ACM Transactions on Modeling and Performance Evaluation of Computing Systems* 3, no. 2 (2018): 1–26.

8. Foundation, T.A.S, “Apache Spark Lightning-Fast Custer Computing,” (2015), http://spark.apache.org/.

9. M. M. Saeed, Z. Al Aghbari, and M. Alsharidah, “Big Data Clustering Techniques Based on Spark: A Literature Review,” *PeerJ Computer Science* 6 (2020): e321.

10. J. Leverich and C. Kozyrakis, “On the Energy (In) Efficiency of Hadoop Clusters,” *ACM SIGOPS Operating Systems Review* 44, no. 1 (2010): 61–65.

11. L. Mashayekhy, M. M. Nejad, D. Grosu, Q. Zhang, and W. Shi, “Energy-Aware Scheduling of Mapreduce Jobs for Big Data Applications,” *IEEE Transactions on Parallel and Distributed Systems* 26, no. 10 (2015): 2720–2733, https://doi.org/10.1109/TPDS.2014.2358556.

12. I. N. Goiri, W. Katsak, K. Le, T. D. Nguyen, and R. Bianchini, “Parasol and Greenswitch: Managing Datacenters Powered by Renewable Energy,” in *Proceedings of the Eighteenth International Conference on Architectural Support for Programming Languages and Operating Systems. ASPLOS ‘13* (ACM, 2013), 51–64, https://doi.org/10.1145/2451116.2451123.

13. H. Li, H. Wang, S. Fang, Y. Zou, and W. Tian, “An Energy-Aware Scheduling Algorithm for Big Data Applications in Spark,” *Cluster Computing* 23 (2020): 593–609.

14. T. D. S. Gonçalves, A. C. S. Beck, and A. F. Lorenzon, “Explorando a variabilidade de processo para otimizar a eficiência energética em servidores de nuvem,” in *Anais do XXIV Simpósio em Sistemas Computacionais de Alto Desempenho* (SBC, 2023), 229–240, https://doi.org/10.5753/wscad.2023.235799.

15. X. Gu, R. Hou, K. Zhang, L. Zhang, and W. Wang, “Application-Driven Energy-Efficient Architecture Explorations for Big Data,” in *Proceedings of the 1st Workshop on Architectures and Systems for Big Data* (ACM, 2011), 34–40.

16. J. L. Berral, Í. Goiri, T. D. Nguyen, R. Gavalda, J. Torres, and R. Bianchini, “Building Green Cloud Services at Low Cost,” in *Proceedings of the 2014 IEEE 34th International Conference on Distributed Computing Systems (ICDCS)* (IEEE, 2014), 449–460.

17. C. H. Forte, A. Manacero, R. S. Lobato, and R. Spolon, “An Energy-Aware Task Scheduler Based in Ownership Fairness Applied to Federated Grids,” in *Proceedings of the 2018 IEEE Symposium on Computers and Communications (ISCC)* (IEEE, 2018), 30–33.

18. D. Abts, M. R. Marty, P. M. Wells, P. Klausler, and H. Liu, “Energy Proportional Datacenter Networks,” *ACM SIGARCH Computer Architecture News* 38, no. 3 (2010): 338–347, https://doi.org/10.1145/1816038.1816004.

19. T. Baker, B. Al-Dawsari, H. Tawfik, D. Reid, and Y. Ngoko, “Greedi: An Energy Efficient Routing Algorithm for Big Data on Cloud,” *Ad Hoc Networks* 35 (2015): 83–96.

20. K. H. Kim, A. Beloglazov, and R. Buyya, “Power-Aware Provisioning of Cloud Resources for Real-Time Services,” in *Proceedings of the 7th International Workshop on Middleware for Grids, Clouds and e-Science. MGC ‘09* (ACM, 2009), 1:1–1:6, https://doi.org/10.1145/1657120.1657121.

21. N. Zacheilas, S. Maroulis, and V. Kalogeraki, “A Framework for Efficient Energy Scheduling of Spark Workloads,” in *Proceedings of the 2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)* (IEEE, 2017), 1132–1142.

22. F. Bernardo, A. Yokoyama, B. Schulze, and M. Ferro, “Avaliação do consumo de energia para o treinamento de aprendizado de máquina utilizando single-board computers baseadas em arm,” in *Simpósio em Sistemas Computacionais de Alto Desempenho (SSCAD)* (SBC, 2021), 60–71.

23. J. Ousterhout, “Always Measure One Level Deeper,” *Communications of the ACM* 61, no. 7 (2018): 74–83.

24. T. Wuttge, “Benchframe: A Framework for Benchmarking Power Monitoring Tools,” (Ph.D. Thesis, Vrije Universiteit Amsterdam) (2025).

25. M. Fahad, A. Shahid, R. R. Manumachu, and A. Lastovetsky, "A Comparative Study of Methods for Measurement of Energy of Computing," *Energies* 12, no. 11 (2019), https://doi.org/10.3390/en12112204.

26. P. Patel, E. Choukse, C. Zhang, et al., "Characterizing Power Management Opportunities for Llms in the Cloud," in *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3. ASPLOS '24* (Association for Computing Machinery, 2024), 207–222, https://doi.org/10.1145/3620666.3651329.

27. N. D. O. Volpini, V. Dias, and D. Guedes, "Uma análise multicamadas do consumo de energia em cargas big data," in *Simpósio em Sistemas Computacionais de Alto Desempenho (SSCAD)* (SBC, 2024), 324–335, https://doi.org/10.5753/sscad.2024.244769.

28. E. Higgs, "Ehiggs/Spark-Terasort," (2018), https://github.com/ehiggs/spark-terasort.

29. M. Li, J. Tan, Y. Wang, L. Zhang, and V. Salapura, "Sparkbench: A Spark Benchmarking Suite Characterizing Large-Scale In-Memory Data Analytics," *Cluster Computing* 20, no. 3 (2017): 2575–2589.

30. M. Li, J. Tan, Y. Wang, L. Zhang, and V. Salapura, "Sparkbench: A Comprehensive Benchmarking Suite for in Memory Data Analytic Platform Spark," in *Proceedings of the 12th ACM International Conference on Computing Frontiers* (ACM, 2015), 53.

31. A. Spark, "Tuning Spark," (2015), https://spark.apache.org/docs/2.2.0/tuning.html.

32. E. Asyabi, M. Sharifi, and A. Bestavros, "ppxen: A Hypervisor Cpu Scheduler for Mitigating Performance Variability in Virtualized Clouds," *Future Generation Computer Systems* 83 (2018): 75–84, https://doi.org/10.1016/j.future.2018.01.015.

33. V. S. Conceição, N. D. O. Volpini, and D. Guedes, "Seshat: uma arquitetura de monitoração escalável para ambientes em nuvem," in *Anais do XVII Workshop em Desempenho de Sistemas Computacionais e de Comunicaç ao, Natal-RN* (Sociedade Brasileira de Computaçao (SBC), 2018), https://doi.org/10.5753/wperformance.2018.3336.