Evaluating the Accuracy of Software Energy Consumption Models for Java Applications at Process and Transaction Levels

Andreas Brunnert
Munich University of Applied Sciences HM
Munich, Germany
brunnert@hm.edu

Abstract

The energy consumption of software systems is hard to quantify because software does not consume energy itself but rather the hardware that it runs on. To address this challenge, technologies such as the Intel Running Average Power Limit (RAPL) interface have been developed to measure the energy consumed by various hardware resources at runtime. Additionally, specialized tools like JoularJX have been created to attribute RAPL measurements to individual software systems and their components. However, these tools are typically limited to software running directly on a server, without virtualization or containerization. In virtualized or containerized environments, direct energy measurements are only possible if the hypervisor or container runtime forwards this information to the guests—a practice not widely adopted, particularly in cloud environments. In such cases, energy consumption models are used to estimate software energy usage based on other metrics, such as CPU, memory, storage, or network utilization. One recently released tool following this approach is the OpenTelemetry Java-Agent extension (OTJAE), making it suitable for environments where direct measurements are not feasible. This study aims to evaluate the accuracy of JoularJX and OTJAE, along with their underlying models, in calculating energy consumption at both the process and transaction levels of Java applications. Both models under examination are based on CPU demand measurements. We are able to show that the model predictions match actual measurements obtained from a hardware server with high accuracy in medium and high load scenarios (50% to 100% CPU utilization) but vary significantly for lower load levels.

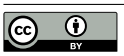
CCS Concepts

• **Software and its engineering → Software performance**.

Keywords

Green Software Engineering, Software Energy Consumption, Green IT

ACM Reference Format:
Andreas Brunnert. 2025. Evaluating the Accuracy of Software Energy Consumption Models for Java Applications at Process and Transaction Levels. In *33rd ACM International Conference on the Foundations of Software Engineering (FSE Companion '25), June 23–28, 2025, Trondheim, Norway*. ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3696630.3728709



FSE Companion '25, Trondheim, Norway

ACM ISBN 979-8-4007-1276-0/2025/06
https://doi.org/10.1145/3696630.3728709

1 Introduction

Assessing the environmental sustainability of software is challenging because software does not directly consume energy or emit carbon equivalent (CO_2-eq) emissions [10, 13]. Instead, these impacts are associated with the hardware resources that software utilizes [7], as illustrated in Figure 1. While the energy consumption of hardware can be measured directly, quantifying CO_2-eq emissions is more complex [9, 12]. These emissions are not a direct result of hardware operation but rather stem from the energy utilities that generate the energy consumed by the hardware.

This paper focuses on the relationship between software and energy consumption, as depicted in Figure 1. Several models and tools have been presented to describe this relationship [6, 7, 15, 16] but only few of them [2, 17, 20–22] provide insights into individual transactions of a software system. Two recent models that enable the analysis of software energy consumption at both the process and transaction levels have been presented in [17] and [1, 2]. These models are implemented in the tools JoularJX[1] and an OpenTelemetry[2] Java-Agent[3] extension[4] (OTJAE), respectively. Both tools estimate the energy consumption of Java-based software systems and their transactions, but their accuracy has not been compared. Therefore, the purpose of this study is to evaluate the accuracy of both tools and their underlying models by comparing their results with actual hardware measurements using a Java-based application. These tools were selected for their ability to capture energy consumption at the granularity of individual software transactions and their ease of integration with existing applications.

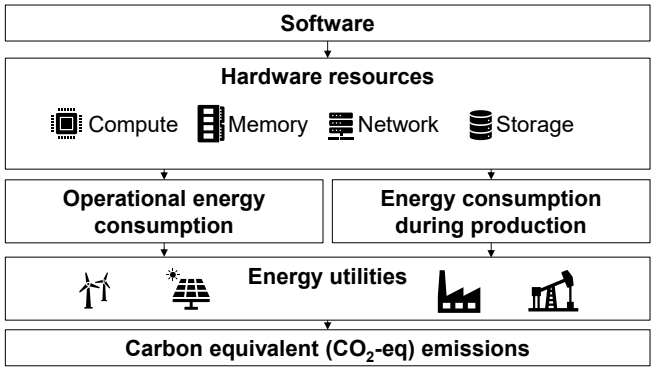


Figure 1: Software energy consumption overview

[1] https://github.com/joular/joularjx
[2] https://opentelemetry.io/
[3] https://github.com/open-telemetry/opentelemetry-java-instrumentation
[4] https://github.com/RETIT/opentelemetry-javaagent-extension

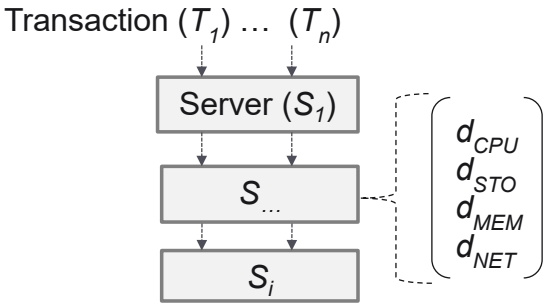


Figure 2: OTJAE metrics (adapted from [3])

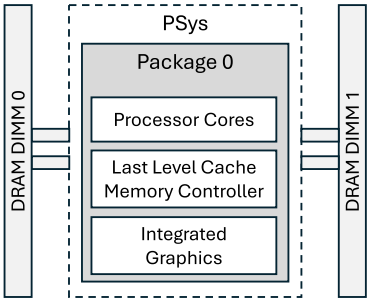


Figure 3: RAPL power domains (adapted from [14])

This paper is organized as follows: The subsequent section delves into the challenges associated with evaluating the energy consumption of software, detailing the metrics, measurement technologies, and calculation models employed in this study. Following this, we describe the experimental setup used to compare the two models and present the results of our experiments. The paper concludes with an outlook on future work in this area.

2 Software Energy Consumption

As previously mentioned, software itself does not consume energy; rather, it is the hardware on which the software runs that does, as illustrated in Figure 1. When discussing software energy consumption, it is crucial to distinguish between operational energy consumption and the energy consumed during production [8, 12]. The latter includes energy expended in the manufacturing of hardware [12] or the development of the software itself [6]. This paper focuses on the operational energy consumption of software, which refers to the energy required to run the software in a production environment.

When examining the operational energy consumption of software, the primary focus is on the hardware resources utilized by the software, as their usage directly influences energy consumption [11]. Consequently, software energy models typically measure hardware resource utilization and calculate software energy consumption based on this data [5, 7].

In the following sections, we will detail the metrics employed by the two tools evaluated in this paper, along with the measurement technologies used to capture these metrics. Subsequently, we will describe the calculation models implemented by JoularJX and OTJAE.

2.1 Metrics

To distinguish the energy consumption of software from that of the underlying hardware, software energy models typically measure the total resource utilization of a system and the specific resource demand (d) of the software itself [5, 18]. This can be achieved by monitoring the resource utilization of the operating system processes associated with the running software or by measuring the resource demand of specific software components, such as individual transactions. The two tools evaluated in this paper support both measurement approaches for Java-based applications.

In JoularJX [17] the CPU demand (d_{CPU}), defined as the time spent on the CPU excluding wait time, is measured periodically for both the overall process and individual threads within the running application. This data is then statistically allocated to each method according to their invocation count.

OTJAE [2] not only focuses on CPU demand (d_{CPU}) but also measures memory (d_{MEM}), storage (d_{STO}), and network (d_{NET}) demand of an application to determine energy consumption. Each of these metrics is captured for every transaction and server involved in request processing, as illustrated in Figure 2. Unlike JoularJX, this tool does not employ a sampling-based approach but rather measures every transaction invocation.

2.2 Measurement Technologies

As detailed in the previous section, both JoularJX and OTJAE capture resource demand metrics of a running Java process. To utilize these tools, they must be attached as agent to a Java application at startup, as shown in Listings 1 and 2.

Listing 1: JoularJX agent start parameter

```
java
    -javaagent:joularjx-$version.jar
    -jar yourProgram.jar
```

Listing 2: OTJAE agent start parameter

```
java
    -javaagent:opentelemetry-javaagent.jar
    -Dotel.javaagent.extensions=
    io.retit.opentelemetry.javaagent.extension.jar
    -jar yourProgram.jar
```

While an application is running, both agents periodically capture the aforementioned metrics. The methods used to achieve this are detailed in their respective papers [2, 3, 17–19] and vary depending on the operating system. For instance, on Linux, such measurements can be obtained using the proc file system[5], which provides data on the resource usage of specific threads and processes.

A notable feature of JoularJX is its ability to directly measure the energy consumption of the underlying hardware using the Intel Running Average Power Limit (RAPL) interface [14]. This technology provides direct access to the energy consumption of the processor and other components, such as random access memory (RAM), as illustrated in Figure 3. RAPL employs different domains to represent the components of a processor. Each processor socket

[5] https://www.man7.org/linux/man-pages/man5/proc.5.html

is represented as a package (see the grey box in the center of Figure 3), encompassing the energy consumption of all processor cores, integrated graphics, and certain uncore components, such as last-level caches and the memory controller [14]. Another crucial RAPL domain is the DRAM domain, which measures the energy consumption of RAM attached to the integrated memory controller [14]. JoularJX periodically reads the RAPL data and distributes the energy readings to the instrumented process, its threads and individual methods / transactions based on the CPU demand measurements using the models outlined in the following section.

The ability to directly read RAPL data allows JoularJX to collect accurate measurements, but it restricts its use to environments where such energy readings are available. This typically includes bare-metal deployments or virtualized environments that propagate RAPL readings to virtual machines or containers[6]. Unfortunately, cloud environments often do not support this feature or dropped support due to changes in the hypervisor[7] or by transitioning to CPU types that do not support RAPL.

In contrast, OTJAE employs a purely model-based approach to calculate energy consumption from the collected metrics. This method enables its application in environments where direct energy readings using RAPL are unavailable.

The software energy models used by JoularJX and OTJAE to allocate power consumption to individual processes, threads, and transactions over time are detailed in the following section. Since JoularJX relies solely on CPU demand measurements, only CPU time is considered for OTJAE in this discussion, despite the tool's capability to model the energy consumption of other resources [2].

2.3 Calculation Models

To determine the energy consumption of individual software processes and their transactions, JoularJX and OTJAE employ different calculation approaches to calculate power consumption over time.

JoularJX periodically (by default in intervals of 1s) allocates the total system energy consumption in Joule (J) measured using RAPL to individual processes and threads based on their CPU demand in the same interval, as detailed in the publication of its predecessor, Jalen [18, 19], and on the JoularJX website[8]. This method involves distributing the system power consumption (P_{system}) in watts (W) in a given time interval (derived from the energy measurements) equally among processes, based on their CPU utilization ($CPU_{UTILprocess}$) as a percentage of the total system utilization (CPU_{UTIL}). The calculation is performed as follows:

$$P_{process} = \left(\frac{CPU_{UTILprocess}}{CPU_{UTIL}}\right) \times P_{system} \quad (1)$$

JoularJX extends this approach to threads and specific methods executed by a thread within a given time interval. Consequently,the overall process power ($P_{process}$), calculated in the previous step, is then attributed to the threads (P_{thread}) of a process by examining the CPU utilization of a given thread ($CPU_{UTILthread}$) within that interval [18]:

$$P_{thread} = \left(\frac{CPU_{UTILthread}}{CPU_{UTILprocess}}\right) \times P_{process} \quad (2)$$

In a final step, JoularJX tracks the ratio of method executions within a single thread over a given time interval. This ratio is used to determine the CPU utilization attributed to each method invocation ($CPU_{UTILmethod}$). This method-specific utilization value is then used to calculate the power consumption of a method:

$$P_{method} = \left(\frac{CPU_{UTILmethod}}{CPU_{UTILthread}}\right) \times P_{thread} \quad (3)$$

The power consumption of a complete transaction can be calculated using JoularJX by summing all method power values involved in a single transaction, as shown in Equation 4. This calculation needs to be done manually and is only necessary when a transaction spans multiple threads. For transactions that only use a single thread, the power consumption of the top-level method includes the entire call tree.

$$P_{transaction} = \sum_{i=0}^{n} P_{method_i} \quad (4)$$

OTJAE follows a methodology published by the Cloud Carbon Footprint (CCF)[9] project to calculate power consumption based on resource demand measurements for different resource types (CPU, memory, storage, network). To determine the power consumption for CPU demands, the CCF methodology uses a simple model originally published by Etsy as part of their Cloud Jewels conversion factors[10]. This model utilizes data on the minimum (idle, P_{CPUmin}) and maximum (100% utilization, P_{CPUmax}) power consumption of a processor (e.g., obtained from SPECpower results[11]) to calculate the actual power consumption (P_{CPU}) based on the current CPU utilization (CPU_{UTIL}) as follows:

$$P_{CPU} = P_{CPUmin} + (CPU_{UTIL} \times (P_{CPUmax} - P_{CPUmin})) \quad (5)$$

This calculated CPU power value can then be attributed to individual processes based on their CPU utilization, similar to the model used by JoularJX shown in Equation 1:

$$P_{CPU_{process}} = \left(\frac{CPU_{UTILprocess}}{CPU_{UTIL}}\right) \times P_{CPU} \quad (6)$$

In OTJAE this model has been extended for individual transactions by splitting the overall power consumption based on the CPU demand of a single transaction. To achieve this, the CPU demand data is converted into a utilization value for a single transaction ($CPU_{UTILtransaction}$) by summing all CPU demand values (d_{CPU}, measured in milliseconds (ms)) of a transaction in a given time interval (from t_0 to t_n in seconds). This sum is then divided by the duration of the time interval (converted to milliseconds by multiplying it with 1000) multiplied with the overall CPU core count (CPU_{cores}), as follows:

$$CPU_{UTILtransaction} = \frac{\sum_{t=t_0}^{t_n} d_{CPU_t}}{CPU_{cores} \times (t_n - t_0) \times 1000} \quad (7)$$

[6] https://joular.github.io/joularjx/ref/vm.html
[7] https://medium.com/teads-engineering/estimating-aws-ec2-instances-power-consumption-c9745e347959
[8] https://joular.github.io/joularjx/ref/how_it_works.html
[9] https://www.cloudcarbonfootprint.org/docs/methodology/
[10] https://www.etsy.com/codeascraft/cloud-jewels-estimating-kwh-in-the-cloud
[11] https://www.spec.org/power_ssj2008/results/power_ssj2008.html

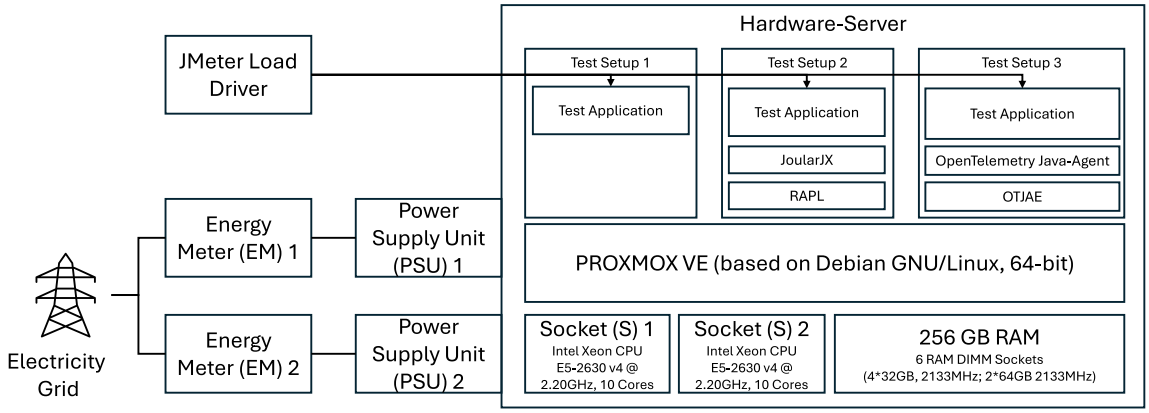


Figure 4: Experiment setup overview

Using this data, the CPU power consumption of a transaction ($P_{CPU_{transaction}}$) can be calculated as follows:

$$P_{CPU_{transaction}} = \left(\frac{CPU_{UTILtransaction}}{CPU_{UTILprocess}} \right) \times P_{CPU_{process}} \quad (8)$$

A similar approach is taken for other resources by OTJAE [2] to calculate the total power consumption of a transaction but in the following experiment, we will focus on the CPU demand models of JoularJX and OTJAE.

3 Experiment Setup

To evaluate the accuracy of the measurement and software energy modeling approaches of both tools, we use the experiment setup shown in Figure 4. For our test, we use a bare metal server with two CPU sockets, each equipped with the same Intel Xeon E5-2630 v4[12] processor. Each processor has 10 cores that run with a base frequency of 2.20Ghz and a max turbo frequency of 3.10Ghz, hyper-threading has been disabled for our experiment. The server is equipped with 256GB RAM, distributed across six RAM sockets containing four 32GB and two 64GB dual inline memory modules (DIMMs). In addition, the server has two redundant power supply units (PSU), whose power consumption can be measured using external energy meters (EM)[13].

This server is typically used to run virtual machines using the Proxmox Virtual Environment (VE)[14] as operating system. However, in this setup, all virtual machines are stopped, and Proxmox VE (based on Debian GNU/Linux) is used directly as a regular Debian-based Linux operating system. This approach is taken to avoid any indirection caused by virtualization, which would complicate the measurement of the actual power demand of the test processes.

As test application, we use a simple REST service implemented with the Spring framework, provided as an example application for OTJAE[15]. This service receives load via the HTTP methods GET, POST, and DELETE, and generates load on the CPU, memory, disk, and network. The application includes an Apache JMeter[16] load test script that executes this load using an open workload, distributing an equal number of requests across each of the HTTP methods at any given time. We use this load test script to generate different load levels, denoted by transactions per second (T/s), which are sent by an external JMeter load test driver to the system under test. The load is applied to the application in three different test setups:

Test setup 1: To evaluate the system and application behavior without any instrumentation, we execute the test application without attaching any instrumentation agent. We use this setup for the evaluations of the overall system and at process level. The other two test setups are used to evaluate the system at transaction level.

Test setup 2: In this setup, the application is started with the JoularJX Java agent (version 3.0.1) attached as shown in Listing 1, which continuously measures the CPU demand of the Java process and threads, as well as the energy consumption of the overall system using RAPL.

Test setup 3: Here the test application is started using the OpenTelemetry Java-Agent (version 2.12.0) and OTJAE (version v0.0.15-alpha) to capture the resource demand as shown in Listing 2.

While the application is running under load, several metrics are measured using different tools:

(1) External power consumption (P_{EM}) using the EM
(2) Power consumption per socket (P_S) and DRAM (P_{DRAM}) derived from RAPL energy readings using the powercapping framework[17]
(3) CPU utilization for the whole system (CPU_{UTIL}) and each CPU socket (CPU_{UTIL_S}) using *sar*[18]
(4) Process CPU utilization ($CPU_{UTILprocess}$) using */proc/[pid]/stat*[19]
(5) Energy consumption per transaction in Joule (J) using JoularJX (test setup 2)
(6) CPU resource demand (d_{CPU}) per transaction using OTJAE (test setup 3)

[12] https://www.intel.com/content/www/us/en/products/sku/92981/intel-xeon-processor-e52630-v4-25m-cache-2-20-ghz/specifications.html
[13] we use Shelly Plug S devices as EM, for specifications see https://www.shelly.com/de/products/shelly-plug-s-gen3
[14] https://www.proxmox.com/
[15] https://github.com/RETIT/opentelemetry-javaagent-extension/tree/v0.0.15-alpha/examples/spring-rest-service
[16] https://jmeter.apache.org
[17] https://www.kernel.org/doc/html/v6.14-rc3/power/powercap/powercap.html
[18] https://www.man7.org/linux/man-pages/man1/sar.1.html
[19] https://www.man7.org/linux/man-pages//man5/proc_pid_stat.5.html

Table 1: Mean idle power consumption

P_{EM1}	P_{EM2}	P_{S1}	P_{DRAM1}	P_{S2}	P_{DRAM2}
55.10W	48.44W	10.31W	2.91W	10.54W	1.26W

4 Experimental Results

As a baseline for the evaluation and to determine the system idle power consumption, we measured the power consumption (P) of the system using two external energy meters (EMs) and RAPL for the two CPU sockets (S1, S2) over a 10-minute interval while the system was idle, except for the measurements. The results are shown in Table 1.

It is evident that the RAPL measurements capture only a small fraction of the overall power consumption. Comparing the RAPL measurements of S1 (P_{S1}) and EM1 (P_{EM1}) reveals a difference of 44.79W, and comparing S2 (P_{S2}) and EM2 (P_{EM2}) shows a smaller difference of 37.9W. These discrepancies can be attributed to several factors. Firstly, the power supply units have an 80 PLUS Gold[20] efficiency rating, meaning approximately 12% of the power is lost during conversion. Accounting for this loss reduces the difference between the external (P_{EM}) and internal (P_S) measurements to 38.18W for P_{EM1} vs. P_{S1} and 32.09W for P_{EM2} vs. P_{S2}.

This additional power is likely consumed by hardware components not measured by RAPL, such as disks, the RAID controller, and the Baseboard Management Controller (BMC). Although we included the DRAM domain in our measurements for each socket (see Figure 3), we can not attribute the difference to the RAM, as the DRAM measurements are very low (about 2W out of the total). However, these measurements might not be fully representative of the 256GB RAM in the system.

4.1 System Level Results

To further understand how the system behaves under different load levels, we used the sample application to create four distinct load levels, as shown in Table 2. We also conducted an idle measurement while the application was running but not receiving any load, to account for background activities such as garbage collection. As shown in Table 2, the application did not consume any significant resources when it was not receiving requests. For the other load levels, the load was applied for 10 minutes at each level after a one minute warm-up period. Between all tests we let the system rest for at least 5 minutes to reduce the temperature to a comparable level for each test execution [4]. Comparing the results in Tables 1 and 2, the difference in power consumption measured by the external meters (P_{EM1} and P_{EM2}) between 0% and ~100% CPU utilization is 51.22W for EM1 (55.1W idle vs. 106.32W at ~100% utilization) and 52.86W for EM2 (48.44W idle vs. 101.3W at ~100% utilization). In contrast, the difference in RAPL measurements (P_{S1},P_{DRAM1} and P_{S2},P_{DRAM2}) is 45.32W for S1 (10.31W + 2.91W idle vs. 49.28W + 9.26W at ~100% utilization) and 46.57W for S2 (10.54W + 1.26W idle vs. 49.42W + 8.95W at ~100% utilization). The discrepancy between the external and internal measurements can be explained by the inefficiency of the power supply unit. Approximately 12% of the power is lost during conversion, which amounts to 6.15W and 6.34W for the EM1 and EM2 growth rates, respectively. Therefore, the growth in power consumption is similar when measured externally compared to the internal measurements.

With these measurements, we can now determine the minimum (idle) and maximum power consumption of the CPUs used in this experiment as shown in Equations 9 and 10. These values are important for the calculations in Equation 5, as outlined in section 2.3. For the following calculations, we combine the minimum and maximum power values of both sockets (S1 and S2), as JoularJX and OTJAE do not differentiate between the sockets in their calculations.

$$P_{CPUmins1s2} = 10.31W + 2.91W + 10.54W + 1.26W = 25.02W \quad (9)$$

$$P_{CPUmaxs1s2} = 49.28W + 9.26W + 49.42W + 8.95W = 116.91W \quad (10)$$

4.2 Process Level Results

In the next step, we evaluate the accuracy of the process power consumption calculations using the models employed by JoularJX ($P_{process}$, see Equation 1) and OTJAE ($P_{CPU_{process}}$, see Equation 6). The results of this evaluation are presented in Table 3. Please note that we do not differentiate between the utilization of the two CPU sockets and use the overall system CPU utilizaton (CPU_{UTIL}) instead; therefore, the power calculations are for both sockets combined.

For this evaluation, we also measured the CPU time consumed by the Java process using the proc filesystem during the measurements presented in the previous section. With the overall runtime of 10 minutes in a steady state, we were able to calculate the process utilization ($CPU_{UTILprocess}$) to differentiate it from the overall system utilization (CPU_{UTIL}). Additionally, we summed the RAPL measurements of the entire system from Table 2 to specify the system power consumption for the JoularJX power model (Equation 1) as shown in Equation 11.

$$P_{system} = P_{S1} + P_{DRAM1} + P_{S2} + P_{DRAM2} \quad (11)$$

Using both utilization values ($CPU_{UTILprocess}$ and CPU_{UTIL}), the system power consumption (P_{system}), as well as the minimum and maximum power values ($P_{CPUmins1s2}$ and $P_{CPUmaxs1s2}$) we applied the process power models (see Equations 1 and 6) explained in section 2.3 to calculate the power consumption of the process. As you can see in Table 3 the process ($CPU_{UTILprocess}$) and system CPU utilization (CPU_{UTIL}) values are very close to each other, so the calculated power consumption of the process (see column $P_{process}$ for JoularJX) is close to the power consumption of the overall system (P_{system}) when employing the model of JoularJX (Equation 1).

As shown in Table 3, the process power model of OTJAE ($P_{CPU_{process}}$, see Equation 6), based on CPU utilization, aligns well with the power demand of the RAPL measurements (P_{system}) for higher CPU utilization rates. However, the linear nature of the model results in error rates of up to 40.56% for lower utilization levels.

In contrast, the RAPL-based calculations of JoularJX (Equation 1) more closely match the actual consumption, as they do not assume a linear model and are derived from real values. The minor differences observed are equivalent to the discrepancies between the process and system CPU utilization values.

[20] https://www.clearesult.com/80plus/program-details#program-details-table

Table 2: Mean system power consumption by load level

Load	Throughput	$CPU_{UTIL_{S1}}$	$CPU_{UTIL_{S2}}$	P_{EM1}	P_{EM2}	P_{S1}	P_{DRAM1}	P_{S2}	P_{DRAM2}
0T/s	0T/s	0.35%	0.30%	55.21W	48.51W	9.33W	2.24W	10.14W	1.08W
150T/s	150T/s	26.55%	23.54%	83.74W	78.40W	30.03W	8.63W	31.68W	8.21W
300T/s	300T/s	54.34%	50.68%	92.88W	87.83W	37.95W	9.08W	38.55W	8.66W
450T/s	450T/s	81.22%	78.51%	99.88W	95.21W	44.52W	9.21W	44.49W	8.95W
600T/s	537.4 T/s	98.13%	97.98%	106.32W	101.30W	49.28W	9.26W	49.42W	8.95W

Table 3: Mean process power consumption by load level

Load	P_{system}	CPU_{UTIL}	$CPU_{UTILprocess}$	OTJAE		JoularJX	
				$P_{CPU_{process}}$	Accuracy = $\frac{P_{CPUprocess}}{P_{system}}$	$P_{process}$	Accuracy = $\frac{P_{process}}{P_{system}}$
150T/s	78.55W	25.04%	24.34%	46.69W	59.44%	76.35W	97.20%
300T/s	94.24W	52.51%	51.19%	71.43W	75.80%	91.87W	97.49%
450T/s	107.17W	79.86%	78.14%	96.28W	89.84%	104.86W	97.85%
600T/s	116.91W	98.05%	97.71%	114.72W	98.13%	116.50W	99.65%

Table 4: Mean transaction power consumption by load level

Load	Transaction	OTJAE			JoularJX		
		$\sum_{t=0}^{600} d_{CPU_t}$	$P_{CPU_{transaction}}$	$\sum P_{CPU_{transaction}}$-$P_{CPU_{process}}$	Energy (J) = $\sum_{t=0}^{600} P_{method_t}$	$P_{transaction}$	$\sum P_{transaction}$-$P_{process}$
50T/s	GET	418870ms	6.70W	-1.11W	5744.40J	9.57W	-4.76W
50T/s	POST	737083ms	11.78W		10450.33J	17.42W	
50T/s	DELETE	1695310ms	27.10W		26757.78J	44.60W	
100T/s	GET	893394ms	10.39W	-0.37W	6633,83J	11.06W	-3.62W
100T/s	POST	1582312ms	18.40W		12519.60J	20.87W	
100T/s	DELETE	3635195ms	42.27W		33789.42J	56.32W	
150T/s	GET	1300417ms	13.35W	-2.47W	6662.42J	11.10W	-7.58W
150T/s	POST	2351304ms	24.14W		13383.31J	22.31W	
150T/s	DELETE	5485072ms	56.32W		38323.82J	63.87W	

4.3 Transaction Level Results

As a final step in this comparison, we examine the differences at the transaction level. To do this, we use the data that both tools measure for each transaction of the system (GET, POST, DELETE for our test application). We use only three of the previous four load levels, as the highest load level in the previous evaluation already utilized the system close to 100%, and the overhead of both tools renders the system unusable at that level.

For JoularJX, we are using test setup 2 (see Figure 4), with a stack-monitoring-sample-rate[21] of 10ms in order to achieve a good compromise between accuracy and overhead. We did this as the highest stack-monitoring-sample-rate of 1ms caused a huge overhead and as such would lead to incorrect results. JoularJX provides the total energy consumption in joules (JoularJX Energy (J)) for each method after each test run which is calculated based on the the sum of all P_{method} calculations in Equation 3 during the test duration of 600s ($\sum_{t=0}^{600} P_{method_t}$), as shown in Table 4. Given that the transactions only use a single thread during their execution, the method energy consumption is equivalent to the transaction energy consumption. Therefore, the mean transaction power consumption ($P_{transaction}$) in watts (W) can be calculated by dividing the energy consumption value by the measurement timeframe, in seconds.

For OTJAE, we use test setup 3 (see Figure 4), storing only the spans[22] captured by the OpenTelemetry agent locally in the logs. Additionally, we have activated only CPU demand measurements on the agent to focus our comparisons on CPU-based calculation techniques. Using the sum of CPU time per transaction (OTJAE $\sum_{t=0}^{600} d_{CPU_t}$) during the test duration (600s), we calculate the CPU utilization per transaction as shown in Equation 7. With this value, we can allocate the overall process power consumption ($P_{CPU_{process}}$ in Table 3) to each individual transaction using Equation 8. The results are presented in Table 4 for the transaction power calculation models of both tools (OTJAE $P_{CPU_{transaction}}$ based on Equation 8 and JoularJX $P_{transaction}$ based on Equation 4).

For the OTJAE results, it is again noticeable that the power consumption is underestimated by the models at lower utilization rates. However, starting from 50% utilization, the results begin to align with those of JoularJX.

[21] https://github.com/joular/joularjx/blob/3.0.1/config.properties#L66

[22] https://opentelemetry.io/docs/concepts/signals/traces/#spans

Table 5: Mean CPU utilization and power consumption of measurement runs on different test setups

Load	$CPU_{UTIL\text{-}none}$	$CPU_{UTIL\text{-}OTJAE}$	$CPU_{UTIL\text{-}JoularJX}$	$P_{system\text{-}none}$	$P_{system\text{-}OTJAE}$	$P_{system\text{-}JoularJX}$
150T/s	25.04%	24.70%	26.95%	78.55W	78.53W	78.08W
300T/s	52.51%	52.69%	60.79%	94.24W	95.12W	91.50W
450T/s	79.86%	78.32%	80.44%	107.17W	107.08W	103.6W

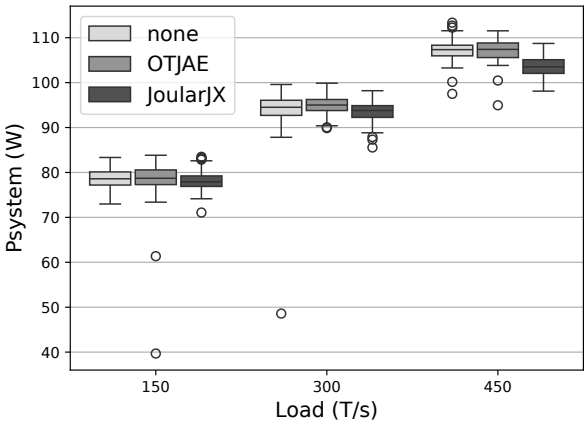


Figure 5: Power consumption distribution of measurement runs on different test setups

Although the power results per transaction for JoularJX appear valid compared to the OTJAE measurements, we observed that JoularJX assumes that the RAPL domain called intel-rapl:1 is always the PSys domain[23]. This special domain was introduced with Intel Skylake [14], and our processor does not support PSys. Therefore, the power calculations are primarily based on the power data of one CPU socket. Despite this, the overall calculation still seems to work, as the CPU demand of the threads and transactions measured by JoularJX includes the processing time on both CPU sockets.

We also compared the results at the transaction level with the totals at the process level in Table 3 to assess the differences (see column "$\sum P_{CPU_{transaction}}$ - $P_{CPU_{process}}$" for OTJAE and "$\sum P_{transaction}$ - $P_{process}$" for JoularJX, respectively). To do this, we summed the power consumption of all transactions at each load level from Table 4 and compared it with the corresponding value at the process level in Table 3. It is evident that the process totals align very well with the results at the transaction level. However, when using the power values from the power models at the transaction and process levels to calculate the energy consumption for the test duration (600s), the results diverge over time. This is especially visible for JoularJX as the sum of the transaction power values underestimates the process power values by 7.58W at the highest load level (450T/s), leading to a difference of 4548J ($7.58W \times 600s$) for the test duration.

4.4 Overhead Evaluation

We also measured the overall system utilization (CPU_{UTIL}) and power consumption (P_{system}) while executing the measurements with JoularJX (see $CPU_{UTIL\text{-}JoularJX}$ and $P_{system\text{-}JoularJX}$) and OTJAE (see $CPU_{UTIL\text{-}OTJAE}$ and $P_{system\text{-}OTJAE}$). The results for each run are shown in Table 5 and Figure 5.

Neither tool introduced significant overhead in terms of CPU utilization or power consumption when compared to measurements without instrumentation ($CPU_{UTIL\text{-}none}$ and $P_{system\text{-}none}$). However, the CPU utilization values were slightly higher when JoularJX was used. Specifically, at the load level of 300 transactions per second (T/s), the CPU utilization was 8.28% higher compared to the measurements without instrumentation (see Table 5). Interestingly, this did not have the same impact on the system's power consumption. We have not yet determined why this is the case, but the lower power consumption when using JoularJX is a pattern observable in all measurements across all load levels as shown in Figure 5.

5 Conclusion and Future Work

This work evaluated the accuracy of two software energy consumption models by comparing their results at both the process and transaction levels of a REST service written in Java. We demonstrated that JoularJX provides more accurate process-based power estimations than OTJAE at lower utilization levels, although both models offer comparable accuracy at higher CPU utilization levels. A similar pattern is observed in transaction-based calculations: JoularJX delivers more precise results at lower utilization levels, while OTJAE's accuracy improves starting at around 50% CPU utilization. The higher accuracy of JoularJX is expected, as it relies on direct energy measurements for its calculations, whereas OTJAE uses a simple linear CPU-based calculation model. However, our findings indicate that OTJAE's calculation model is sufficiently accurate at CPU utilization levels beyond 50%. This suggests that using OTJAE is a viable option in environments where direct RAPL-based measurements are not accessible, provided the achievable accuracy meets the requirements.

This study has several limitations. We examined only two of the many available software energy consumption models [5]. Future work should include evaluations of additional models and tools [7] in this area. Notably, OTJAE can also measure and calculate the power consumption of memory, storage, and network demands of software transactions. Future research will compare these results with other tools offering similar capabilities. It would be particularly interesting to assess how accurately the energy consumption of different application types, which stress hardware resources in varying ways, can be estimated.

Another limitation is that our test environment utilized an older CPU from the Intel Broadwell generation (14nm lithography). Newer Intel processors, which include the RAPL domain PSys that covers the thermal and power specifications of the entire system on a chip [14], should be evaluated similarly. The same applies to AMD-based CPUs that support RAPL measurements.

[23] https://github.com/joular/joularjx/blob/3.0.1/src/main/java/org/noureddine/joularjx/cpu/RaplLinux.java#L30C38-L30C91

References

[1] Andreas Brunnert. 2024. Green Software Metrics. In *Companion of the 15th ACM/SPEC International Conference on Performance Engineering* (London, United Kingdom) *(ICPE '24 Companion)*. Association for Computing Machinery, New York, NY, USA, 287–288. doi:10.1145/3629527.3652883

[2] Andreas Brunnert and Ferdinand Gutzy. 2024. Extending the OpenTelemetry Java Auto-Instrumentation Agent to Publish Green Software Metrics. In *15th Symposium on Software Performance 2024*. Linz, Austria.

[3] Andreas Brunnert and Helmut Krcmar. 2017. Continuous performance evaluation and capacity planning using resource profiles for enterprise applications. *Journal of Systems and Software* 123 (2017), 239–262. doi:10.1016/j.jss.2015.08.030

[4] Luís Cruz. 2021. Green Software Engineering Done Right: a Scientific Guide to Set Up Energy Efficiency Experiments. http://luiscruz.github.io/2021/10/10/scientific-guide.html. doi:10.6084/m9.figshare.22067846.v1 Blog post..

[5] Miyuru Dayarathna, Yonggang Wen, and Rui Fan. 2016. Data Center Energy Consumption Modeling: A Survey. *IEEE Communications Surveys & Tutorials* 18, 1 (2016), 732–794. doi:10.1109/COMST.2015.2481183

[6] Stefanos Georgiou, Stamatia Rizou, and Diomidis Spinellis. 2019. Software Development Lifecycle for Energy Efficiency: Techniques and Tools. *ACM Comput. Surv.* 52, 4, Article 81 (Aug. 2019), 33 pages. doi:10.1145/3337773

[7] Achim Guldner, Rabea Bender, Coral Calero, Giovanni S. Fernando, Markus Funke, Jens Gröger, Lorenz M. Hilty, Julian Hörnschemeyer, Geerd-Dietger Hoffmann, Dennis Junger, Tom Kennes, Sandro Kreten, Patricia Lago, Franziska Mai, Ivano Malavolta, Julien Murach, Kira Obergöker, Benno Schmidt, Arne Tarara, Joseph P. De Veaugh-Geiss, Sebastian Weber, Max Westing, Volker Wohlgemuth, and Stefan Naumann. 2024. Development and evaluation of a reference measurement model for assessing the resource and energy efficiency of software products and components—Green Software Measurement Model (GSMM). *Future Generation Computer Systems* 155 (2024), 402–418. doi:10.1016/j.future.2024.01.033

[8] Udit Gupta, Mariam Elgamal, Gage Hills, Gu-Yeon Wei, Hsien-Hsin S. Lee, David Brooks, and Carole-Jean Wu. 2022. ACT: designing sustainable computer systems with an architectural carbon modeling tool. In *Proceedings of the 49th Annual International Symposium on Computer Architecture* (New York, New York) *(ISCA '22)*. Association for Computing Machinery, New York, NY, USA, 784–799. doi:10.1145/3470496.3527408

[9] Peter Henderson, Jieru Hu, Joshua Romoff, Emma Brunskill, Dan Jurafsky, and Joelle Pineau. 2020. Towards the systematic reporting of the energy and carbon footprints of machine learning. *J. Mach. Learn. Res.* 21, 1, Article 248 (Jan. 2020).

[10] Abram Hindle. 2016. Green Software Engineering: The Curse of Methodology. In *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, Vol. 5. 46–55. doi:10.1109/SANER.2016.60

[11] Avita Katal, Susheela Dahiya, and Tanupriya Choudhury. 2023. Energy efficiency in cloud computing data centers: a survey on software technologies. *Cluster Computing* 26, 3 (June 2023), 1845–1875. doi:10.1007/s10586-022-03713-0

[12] Tom Kennes. 2023. Measuring IT Carbon Footprint: What is the Current Status Actually? arXiv:2306.10049 [cs.SE] https://arxiv.org/abs/2306.10049

[13] Eva Kern, Lorenz M. Hilty, Achim Guldner, Yuliyan V. Maksimov, Andreas Filler, Jens Gröger, and Stefan Naumann. 2018. Sustainable software products—Towards assessment criteria for resource and energy efficiency. *Future Generation Computer Systems* 86 (2018), 199–210. doi:10.1016/j.future.2018.02.044

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

[15] Patricia Lago, Qing Gu, and Paolo Bozzelli. 2014. *A systematic literature review of green software metrics.* VU Technical Report.

[16] Javier Mancebo, Félix García, and Coral Calero. 2021. A process for analysing the energy efficiency of software. *Information and Software Technology* 134 (2021), 106560. doi:10.1016/j.infsof.2021.106560

[17] Adel Noureddine. 2022. PowerJoular and JoularJX: Multi-Platform Software Power Monitoring Tools. In *18th International Conference on Intelligent Environments (IE2022)*. Biarritz, France.

[18] Adel Noureddine, Aurelien Bourdon, Romain Rouvoy, and Lionel Seinturier. 2012. Runtime monitoring of software energy hotspots. In *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering* (Essen, Germany) *(ASE '12)*. Association for Computing Machinery, New York, NY, USA, 160–169. doi:10.1145/2351676.2351699

[19] Adel Noureddine, Romain Rouvoy, and Lionel Seinturier. 2015. Monitoring energy hotspots in software. *Automated Software Engineering* 22, 3 (Sept. 2015), 291–332. doi:10.1007/s10515-014-0171-1

[20] Abhinav Pathak, Y. Charlie Hu, and Ming Zhang. 2012. Where is the energy spent inside my app? fine grained energy accounting on smartphones with Eprof. In *Proceedings of the 7th ACM European Conference on Computer Systems* (Bern, Switzerland) *(EuroSys '12)*. Association for Computing Machinery, New York, NY, USA, 29–42. doi:10.1145/2168836.2168841

[21] Shinan Wang, Hui Chen, and Weisong Shi. 2011. SPAN: A software power analyzer for multicore computer systems. *Sustainable Computing: Informatics and Systems* 1, 1 (2011), 23–34. doi:10.1016/j.suscom.2010.10.002

[22] Shinan Wang, Youhuizi Li, Weisong Shi, Lingjun Fan, and Abhishek Agrawal. 2012. Safari: Function-level power analysis using automatic instrumentation. In *2012 International Conference on Energy Aware Computing*. 1–6. doi:10.1109/ICEAC.2012.6471014