Extending the OpenTelemetry Java Auto-Instrumentation Agent to Publish Green Software Metrics

Andreas Brunnert (brunnert@hm.edu)
Ferdinand Gutzy (ferdinand.gutzy@hm.edu)
Munich University of Applied Sciences HM, Munich, Germany

Abstract

The increasing focus on the carbon emissions of the IT sector has led to the creation of green software metrics that attempt to quantify the environmental impact of software in terms of carbon emissions. One of the most prominent examples is the Software Carbon Intensity (SCI) metric, recently standardized as ISO/IEC 21031:2024, which is defined as the rate of carbon emissions for a software system. Despite the merits of these metrics, development teams often lack the tools to capture them for their software systems. This paper attempts to address this challenge by presenting an extension to the OpenTelemetry Java Auto-Instrumentation Agent to collect not only traditional performance metrics such as response time, throughput, and resource utilization, but also carbon emission data for an instrumented software system.

1 Introduction

At the time of writing there is no commonly agreed upon standard for quantifying the carbon emissions of software systems[2]. Researchers and industry practitioners alike have proposed green software metrics to fill this gap[3]. One of the green software metrics that is already standardized as ISO/IEC 21031:2024[1] is the Software Carbon Intensity (SCI) metric[2]. SCI is defined as follows:

$$SCI = ((E * I) + M) per R$$

E: energy consumed by a software system (kWh)

I: carbon intensity of electricity (gCO2eq/kWh)

M: embodied carbon, the amount of carbon emitted during the creation and disposal of a hardware device (gCO2eq)

R: functional unit that describes how the application scales (e.g, transactions per minute or simultaneous users).

This paper describes an extension of the OpenTelemetry (OTel) Java Auto-Instrumentation Agent[3] to collect the relevant data to calculate the SCI metric for individual transactions of an instrumented software system. The OTel Java Auto-Instrumentation Agent can be attached to any Java application running on a Java version higher than 8 and automatically collects traces, metrics and logs for commonly used libraries and frameworks automatically. It does this by injecting measurement probes before and after each method invocation to capture so-called spans. In this work we extend the probes of this agent to collect the data to calculate the SCI value for all transactions (e.g., API calls) executed by the instrumented application.

[1]`https://www.iso.org/standard/86612.html`
[2]`https://sci.greensoftware.foundation`
[3]`https://github.com/open-telemetry/opentelemetry-java-instrumentation`

2 Architectural Overview

The general architecture of such an extension has been outlined in our previous work [2] and is shown in Figure 1. We use OpenTelemetry[4] as standard for capturing and transferring the metrics from the application to Prometheus[5] as time series database. Grafana[6] is used to visualize and calculate the carbon emissions per transaction as well as per server, container, or virtual machine involved in processing a transaction over time.

Compared to our previous work in [2], we have simplified the overall architecture by eliminating the need to integrate with an external service to capture carbon emission data. In the current implementation the application agent sends the carbon emission data for each transaction directly along with the other transaction data such as resource demands, response time and throughput.

The reason for this change is that we are now using freely available data[7] from the Cloud Carbon Footprint (CCF)[8] project to calculate the emissions, which are bundled with the application and do not change over time.

[4]`https://opentelemetry.io`
[5]`https://prometheus.io`
[6]`https://grafana.com`
[7]`https://github.com/cloud-carbon-footprint/ccf-coefficients`
[8]`https://www.cloudcarbonfootprint.org/`

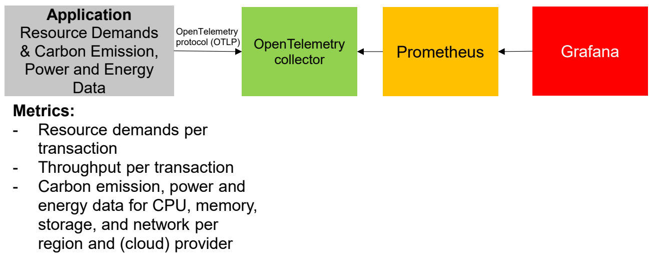
Application
Resource Demands & Carbon Emission, Power and Energy Data
OpenTelemetry protocol (OTLP)
OpenTelemetry collector
Prometheus
Grafana
Metrics:
- Resource demands per transaction
- Throughput per transaction
- Carbon emission, power and energy data for CPU, memory, storage, and network per region and (cloud) provider

Figure 1: Architectural Overview [2]

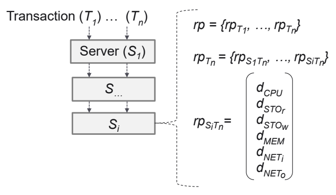
Transaction (T_1) … (T_n)
Server (S_1)
$S_{...}$
S_i
$rp = \{rp_{T_1}, \ldots, rp_{T_n}\}$
$rp_{T_n} = \{rp_{S_1T_n}, \ldots, rp_{S_iT_n}\}$
$rp_{S_iT_n} = \begin{pmatrix} d_{CPU} \\ d_{STO_r} \\ d_{STO_w} \\ d_{MEM} \\ d_{NET_i} \\ d_{NET_o} \end{pmatrix}$

Figure 2: Resource Profile [1]

3 Collecting Resource Demands

The basis for assessing the carbon emissions of a software system is the amount of hardware resources (i.e., CPU, memory, storage, network) used during the transaction processing as this drives the energy consumption and helps us to calculate the E value of the SCI formula.

In [1] and [2] we have already shown and evaluated how to measure resource demand for a software system. We use the same concepts to capture the resource demand (d) values for CPU ($\mathrm{d_{CPU}}$), memory ($\mathrm{d_{MEM}}$), storage ($\mathrm{d_{STO}}$), and network($\mathrm{d_{MEM}}$) for each method (or transaction) by extending the existing measurement probes injected by the OTel Java Auto-Instrumentation Agent using a SpanProcessor[9]. The resource demand for CPU is measured as CPU time in ms, while the other demands for memory, storage and network are measured as bytes allocated in memory or read (r) from respectively written (w) to storage or network. We collect this data in so-called resource profiles as shown in Figure 2 in which we store a vector of resource demands per transaction of the software system.

This resource demand data now needs to be translated into carbon emission data as explained in the following section.

4 Calculating Carbon Emissions

Since we cannot use the resource demand data as a direct input for the SCI calculation, we need to translate it into energy consumption and carbon emission data. In order to do this, we follow the recommendations provided in the CCF methodology[10].

For CPU resources, the CCF methodology and our implementation rely on a simple model originally published by Etsy as part of their Cloud Jewels conversion factors[11], which uses data on the minimum (idle, $\mathrm{P_{CPUmin}}$) and maximum (100% utilization, $\mathrm{P_{CPUmax}}$) power consumption of a processor (e.g., taken from SPECpower results[12]) to calculate the actual power consumption ($\mathrm{P_{CPU}}$) based on the current CPU utilization ($\mathrm{CPU_{util}}$) as follows:

$$P_{CPU} = P_{CPUmin} + (CPU_{util} * (P_{CPUmax} - P_{CPUmin}))$$

The CPU power model also takes into account the data center's Power Usage Effectiveness (PUE), a metric that describes how much power is consumed by the computing equipment in a data center relative to the total power (including cooling and other energy consumers). PUE is not directly measured, but is configured based on the cloud provider or data center in which the software is running.

$$P_{CPUinclPUE} = P_{CPU} * PUE$$

As the total CPU utilization ($\mathrm{CPU_{util}}$) is not directly related to the CPU demands of a single transaction of a system, we need to measure $\mathrm{CPU_{util}}$ in order to calculate the current power consumption of the CPU, and then we can allocate the power consumption proportionally to single transactions of the system. To do this, the CPU demand data needs to be converted into a utilization value ($\mathrm{CPU_{utilT}}$) by summing up all CPU demand values (measured in ms) of a transaction in a given time period (where $\mathrm{t_0}$ is the start timestamp of the time period in ms and $\mathrm{t_n}$ is the end of the end timestamp of the time period in ms) and dividing this sum by the total time the CPU was busy in that time period as follows:

$$CPU_{utilT} = (\sum_{t_0}^{t_n} d_{CPU}) \div (CPU_{util} * (t_n - t_0))$$

Using this data we can calculate the power usage of the transaction ($\mathrm{P_{TCPU}}$) as follows:

$$P_{TCPU} = P_{CPUinclPUE} * CPU_{utilT}$$

Using this power consumption value, we can calculate the energy (E) in kilowatt-hours (kWh) by considering the time interval ($\mathrm{t_0}$ to $\mathrm{t_n}$) in hours as follows:

$$E_{CPU} = (P_{TCPU} * (t_n - t_0))/1000$$

To determine the carbon emissions ($\mathrm{c_{CPU}}$ in gCO2eq) required to produce the energy, we need to know the grid emissions factor (GEF). The GEF denotes the carbon intensity of producing one kWh in a given region and is typically expressed in grams of CO2 equivalents per kWh (gCO2eq/kWh). The GEF

[9]`https://opentelemetry.io/docs/specs/otel/trace/sdk/#span-processor`

[10]`https://www.cloudcarbonfootprint.org/docs/methodology`

[11]`https://www.etsy.com/codeascraft/cloud-jewels-estimating-kwh-in-the-cloud`

[12]`https://www.spec.org/power_ssj2008/results/power_ssj2008.html`

is derived by the region in which the software is operated and this region needs to be configured for the agent. For cloud providers there is a list of grid emissions for their regions provided by CCF[10]. Alternative sources for this value would be ElectricityMaps[13] or Carbon Aware Computing[14].

$$c_{CPU} = E_{CPU} * GEF$$

For storage, memory, and network the CCF methodology[10] provides a power consumption per GB (P_{GB}) depending on the type (RAM/HDD/SSD), again following the initial publication by Etsy[11] for storage and network. Therefore, it is important to know how much data is consumed in a given time period, which can be calculated based on the resource demand measurements. Since the resource demand measurement values are in bytes, we need to multiply them by 10^{-9} to convert them to GB values. Therefore, the calculation of energy consumption (E) and carbon emissions (c) is the same for memory, network, and storage:

$$E_{GB} = (P_{GB} * PUE * (t_n - t_0))/1000$$

$$c_{MEM} = \sum_{t_0}^{t_n} d_{MEM} * 10^{-9} * E_{GB} * GEF$$

$$c_{NET} = \sum_{t_0}^{t_n} d_{NET} * 10^{-9} * E_{GB} * GEF$$

$$c_{STO} = \sum_{t_0}^{t_n} d_{STO} * 10^{-9} * E_{GB} * GEF$$

Based on the previous calculations we can now calculate the total carbon emissions of a transaction as shown in the following section.

5 SCI Calculation

For any given transaction we can now use the the sum of the carbon emissions (c_T) of each resource as the E*I part of the SCI specification.

$$E * I = c_T = c_{CPU} + c_{MEM} + c_{STO} + c_{NET}$$

What is still missing is the embodied carbon (M) part of the SCI calculation. Since the total embodied emissions (TEE) of a hardware server cannot be used for modern runtime environments such as virtual machines (VMs) or containers, CCF[10] suggests using the relative amount of resources used by a VM or container as M (e.g., CPU count of the instance (CPU_{count_i}) divided by the CPU count of the entire server (CPU_{count_s})). In addition, the time that the server is in use must be taken into account. To do this, the coefficient 0.0289[15] is used to scale down the TEE value, typically given in kgCO2eq, to gCO2eq/h based on four years of usage. Using the above ratio and our time interval the value for M is calculated as follows:

$$M = TEE * 0.0289 * (t_n - t_0) * (CPU_{count_i} / CPU_{count_s})$$

As this value is not yet scaled to the individual transaction, we cannot add it directly to the SCI value of a single transaction (SCI_T). We use the previously introduced CPU_{utilT} to properly scale this value to the various transactions in a given time frame as follows:

$$M_T = M * CPU_{utilT}$$

Finally, we can calculate the SCI value for any given transaction (SCI_T) by adding c_T and M_T. The R value can be omitted because all values are already scoped to a single functional unit, namely the transaction for which the resource demand values were measured.

$$SCI_T = c_T + M_T$$

6 Conclusion and Future Work

This work presented an extension to the OpenTelemetry Java Auto-Instrumentation agent that collects the relevant data to automatically calculate an SCI score per transaction. The current implementation state of the emission calculation relies on many data sources that are based on assumptions and configurations. In addition, the data sources are static which is not realistic considering the GEF differences during day and night for renewable energy sources. Therefore, we are working on integrating other data sources that are more up-to-date with the current grid system on the one hand and the actual energy consumption of the system on the other hand. In addition, we plan to perform evaluations comparing our calculations with real measurements to see how well they represent the reality.

References

[1] A. Brunnert and H. Krcmar. "Continuous performance evaluation and capacity planning using resource profiles for enterprise applications". In: *Journal of Systems and Software* 123 (2017), pp. 239–262.

[2] A. Brunnert. "Green Software Metrics". In: *Companion of the 15th ACM/SPEC International Conference on Performance Engineering.* ICPE '24 Companion. London, United Kingdom: Association for Computing Machinery, 2024, pp. 287–288.

[3] A. Guldner et al. "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)". In: *Future Generation Computer Systems* 155 (2024), pp. 402–418.

[13] `https://www.electricitymaps.com/`

[14] `https://www.carbon-aware-computing.com/`

[15] 0.0289 = 1000 (kg to g) ÷ 4 (years of server usage) ÷ 12 (months per year) ÷ 30 (days per month) ÷ 24 (hours per day)