A Resource Utilization Score for Software Energy Consumption

Erik Jagroep, Jan Martijn E. M. van der Werf,
Jordy Broekman, Sjaak Brinkkemper
Utrecht University,
Department of Information and Computing Sciences
Princetonplein 5, 3584 CC Utrecht, The Netherlands
Email: {e.a.jagroep, j.m.e.m.vanderwerf, j.broekman, s.brinkkemper}@uu.nl

Leen Blom, Rob van Vliet
Centric Netherlands B.V.
P.O. Box 338,
2800 AH Gouda, The Netherlands
Email: {leen.blom, rob.van.vliet}@centric.eu

***Abstract*—Software as the true consumer of power and its potential contribution to reach sustainability goals is increasingly being acknowledged. Studies so far have presented successful results and methods to address the energy consumption of the software, indicating that different stakeholders striving for green software have different information needs with respect to their goals. However, currently there is no uniform manner to communicate measurements to the different stakeholders such that key findings are clearly identifiable and easy to understand, which is likely to hamper green software practices. In this paper we propose a metric that expresses a score for the resource utilization, such as power consumption, of a software product. The metric is designed to be a single score and is flexible to encompass those aspects that a stakeholder considers relevant in the context of software energy consumption. The metric was applied on two applications and allowed for objective comparison of application configurations and versions. Also the behavior of these applications across different hardware configurations could be analyzed. In addition to the metric we investigate means to visualize measurements which enhances communication and helped with highlighting the key findings.**

***Index Terms*—Software energy consumption, Resource utilization, Visualization, Sustainability.**

I. Introduction

The recent focus on the Energy Consumption (EC) of software has had a positive impact on the spectrum of sustainable, i.e. energy efficient [1], solutions in the ICT sector. Although hardware consumes energy, software directs the hardware on using the available resources [2] and numerous studies become available that report improvements on energy related aspects with the software itself as the central topic [3], [4]. In a recent study, Hindle [5] presents a method to analyze EC across releases of software products, which provides a basis for sustainable endeavors a software producing organization [6] might undertake. Despite this, organizations still struggle with addressing the software products in terms of their EC [7].

Key in this struggle is that addressing the EC of software confronts a software producing organization, specifically software developers, with a multifaceted issue. Depending on its deployment, measuring the EC of software can be done using relatively cheap hardware devices. However, apart from EC measurements, the software is also characterized using performance measurements which allows for analysis of the software's energy consuming behavior and resource usage. Performance measurements in our case refer to hardware resource performance and provide insight in how the hardware components are stressed when processing instructions. Developers should be able to use this information to address this relatively unknown, non-functional aspect [8] of the software.

Based on previous work (i.e. [9], [10]), however, we found that these measurements are not easy to communicate to stakeholders. Our experience is that in some cases deep knowledge is required to understand the measurements, i.e. how should a specific (performance) measurement or metric be interpreted, and that the key findings that require further investigation are difficult to identify. Issues that are strengthened by developer knowledge that is lacking in this area [7]. As a result, we had developers and software architects searching for the right information and identified a potential inhibiting factor to start addressing the EC of the software.

In this paper we investigate a means to effectively communicate Software Energy Consumption (SEC) related measurements to stakeholders wanting to address the sustainability of their software. Effectively in our case means that the information is easy to understand, is reported uniformly to enhance recognition, and clearly communicates any key findings. Ideally we are able to express the results in a metric that allows to objectively compare the software across different contexts (e.g. releases, installations).

Based on the above we formulate our main research question as follows:

RQ: *How can we effectively express the resource utilization for executing a software product in relation to the SEC?*

In the RQ, we refer to various resources as it is clear that energy is not the only involved resource. However, as this differs per study, a possible metric should be flexible to encompass those resources that are considered important in a given context. Translating the focus on resource usage to a metric, we contribute by providing a Resource Utilization Score (RUS) for the SEC. The RUS helps in the analysis of SEC related measurements and their visualization.

The remainder of this paper is structured as follows. We first present the related work (Sect. II) and continue with

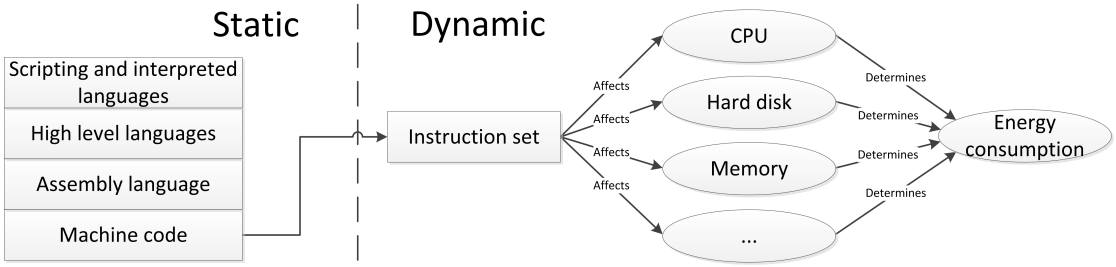

Figure 1. A translation from the language abstraction stack (static) to the energy consumers (dynamic).

the creation of the RUS (Sect. III). After constructing the score we apply the RUS in an experiment (Sect. IV) and evaluate the results (Sect. V). Finally, we discuss our findings (Sect. VI) and provide a conclusion including directions for future research (Sect. VII).

II. Related Work

The term sustainability in the ICT domain is aimed at controlling ecological, economical and social dimensions (of ICT) to the extent that future stakeholders are not compromised in the ability to meet their needs [11]. Sustainable ICT, also coined 'Green IT' [1], helps to improve energy efficiency, lower greenhouse gas emissions and promotes reuse and recycling. Recently a technical dimension has been added for software intensive systems [12], addressing the aspects of the constantly changing environment in which software is executed. Our focus on optimizing the EC of software, i.e. 'green software' [13], is mainly concerned with the ecological dimension, however economic, social and technical goals could also be addressed using a RUS.

A. Addressing Software Sustainability

A popular approach towards creating green software is to consider sustainability [13], or energy efficiency [14], [15], as a quality aspect for the software. Doing so allows software producing organizations to consider sustainability aspects during the design of the software, i.e. with its software architecture, and make trade-offs with other quality aspects (e.g [16]). However, analogous to the other quality attributes [17], working on software quality could require significant investments in terms of time, specialized knowledge to address specific issues and analysis across architectural views [18]. An EC perspective [9] could help guide any efforts in this regard, but the complexity of the matter could still pose difficulties.

If we look at the language abstraction stack (Fig. 1), we can explain the complexity. After a blueprint for the software is made in the form of its software architecture [8], the actual software development can take place. Through several layers of abstraction an instruction set is acquired that is executed by the hardware. This brings us from the static to the dynamic aspect of software. Performing the sequences of instructions affects the hardware components and available (virtualized) resources, which in turn determines the EC induced by the software. Hence, as developers have limited control over the instruction set, they can only await what effect changes in the source code might have. Some tools are available though, e.g. Big-O notation [19], but again complexity issues rise due to the large software systems that organizations produce.

To exert control, apart from EC measurements, studies in the area of green software report a variety of metrics depending on the context in which a study was performed and the stakeholders that are involved. For example, performance [9], [20] and software [5] metrics are used to characterize a software product, which is typically input for developers and architects. These two stakeholders could also benefit from knowing the EC on process level [4]. As the developer is responsible for writing the code, these insights could stimulate to, for example, minimize the number of invocations for a specific, high energy consuming method.

On the other hand we find metrics on infrastructure and organizational level, that are useful for higher level sustainability goals (e.g. by product management [21]). Green performance indicators [22] can, among others, be used to monitor infrastructure facilities and are of interest when EC needs to be considered on datacenter level. In their work, Lundfall et al. [23] present a tool to make the economic impact of green practices explicit with the purpose of justifying green practices on management level. The relation with green software is apparent though as the figures often still stem from low level computing and application measurements.

B. Resource Utilization

Attributing the EC to the software itself requires monitoring the usage of the available hardware resource; i.e. performance measurements. Performance measurements provide insight in how the hardware components are stressed when processing instructions and specific performance metrics can be identified for each individual component [9]. For example, [24] monitors the overall system throughput, CPU, memory and hard disk through the 'number of instructions', 'CPU utilization', 'memory utilization', and 'disk transactions per second' performance metrics. A different approach is to assume theoretical EC figures, e.g. based on the specifications provided by the manufacturer [25], however this approach fails to account for the dynamic behavior of the software.

Important in this field of research is to select the relevant hardware components to monitor and the right instructions to process. Traditionally the CPU has been identified as the

most decisive component for the EC by a system [26], [27]. However, CPU based energy models do not capture all the power drawn by a system [28]. In [29] the contribution of each laptop component to the energy consumed is identified, e.g. the optical drive and LCD-backlight, and shows only 20% can be attributed to the CPU. On the other hand, in large-scale infrastructures the EC of network equipment is argued not to fluctuate heavily with increased traffic [30]. With regard to the instructions to process, a workload model should be made to reflect realistic conditions [24].

Resource monitoring is relevant on different levels related to green software. While investigating the EC of a server, a static and dynamic component can be identified [28]; static is the EC while the system is idle, i.e. minimum resources are used, and the dynamic EC fluctuates with the usage of the resources. As the static component is a large part of the EC, minimizing the absolute number of physical servers could significantly contribute to achieving the desired EC savings. In a cloud architecture the load of resources can be monitored (CPU, disk storage and network interface) and nodes switched on or off to minimize the overall power consumption [27]. This potential to scale up or down, based on performance monitoring, could help in achieving cost-effective scalability [31]. The dynamic part is relevant for green software practices and is determined by the resource utilization of the software.

C. Labeling Software Products

In [32] work has been done towards creating eco-labels for software in terms of a definition, criteria, form of representation, target groups and stakeholders. Sustainability is considered in the broadest sense of the word and, for example, also includes the sustainability aspects of the development process for the software. Following the main criteria that are identified, Kern et al. [32] continue with selecting those criteria that should be considered based on the life cycle phase of the software. This selection appears to be in line with the metrics and information needs as discussed above.

We deviate from [32] with respect to the form of representation. The authors build on international examples of eco-labels, although valuable in their own rights, which often do not allow for many details (i.e. low-level metrics) and are solely focused on specific aspects (e.g. CO_2 emissions). Consequently, apart from providing a starting point, the suggested eco-labels would be of limited practical value for those wanting to address the sustainability of their software.

Having said this, we consider the RUS and the eco-labels complementary in the area of green software. Eco-labels could help in selecting the tools, frameworks and services that positively impact the EC of a software product. For example as a criterion for the service-adaptation tactic [33] in a cloud context. The RUS, on the other hand, could serve as a more hands-on tool for software developers and architects.

III. Resource Utilization Score

In the search to determine a RUS for software EC, we investigate a means to combine performance metrics into a single score. However, we also acknowledge the importance of clearly communicating any key findings and the importance of lowering the threshold to interpreting the measurements through its presentation. To this end, we include a means to visualize measurements in our investigation.

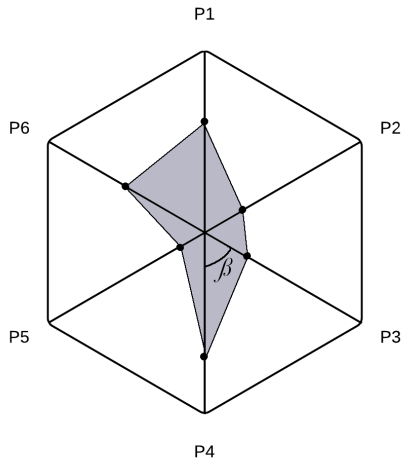

Figure 2. An example of a radar chart with example profile.

A. Visualizing Measurements

In general visualizing a measurement, e.g. per software element [4], simplifies its communication and interpretation compared to raw measurements. However, visualizing measurements individually limits the user in combining metrics and neglects any relation between them. In the case of SEC, the hardware components receive instructions from some instruction set translated from the software. As such there is bound to be a relation between the instructions that the components have to process. Consequently, we aim for a visualization method that is able to encompass all measurements in one figure and can serve as a basis for determining a score.

For our purposes we found a solution in the radar chart. According to Schutz, Speckesser and Schmid [34] the radar chart serves four goals:

1) Visualize interrelated performance measures through standardized scales.
2) Produce an effective description of selected performance dimensions in one synthetic indicator.
3) Analyze the change in overall performance between two points in time by comparing the surface of the same object.
4) Compare different objects through the shape of the surface for these objects.

Translating these goals to our context we can use a radar chart to visualize the performance dimensions related to the SEC, combine the dimensions into one single indicator (i.e. a score), analyze changes on different points in time (e.g. across releases [10]) and compare software products to one another. Under the condition that the same metrics are used for the chart. An example of the radar chart is provided in Fig. 2,

showing the dimensions (P1 through P6) for an unspecified object and the forthcoming surface (grey area) resulting from the scores on these dimensions.

In [35] the idea of the radar chart is applied to benchmark the performance of national labor markets. Although the results look promising, limitations of using the surface of the radar chart are also identified:

- The right dimensions should be selected for benchmarking a specific aspect.
- The right metrics should be selected to characterize these dimensions.
- The dimensions could contribute differently to an indicator (score) and as such could require a weighted inclusion.

The first two limitations concern selecting the right dimensions, i.e. axes, and the right performance metrics to characterize these axes. We address these limitations for our research in Sect. III-B. The third limitation affects the calculation of the RUS and is addressed in Sect. III-D.

B. Determining the Axes

It should be clear that the aspect under investigation in our research is the SEC. The first step is to determine the dimensions, i.e. the axes, that will form the radar chart for this aspect. From the related work we are able to distill the following dimensions that are directly or indirectly affected by the instruction set:

- CPU
- Memory
- Hard disk
- Network
- Power consumption
- Execution time

Each dimension has its own performance metrics (e.g. % usage versus bytes total per second [10]) and each metric is expressed in its own unit and scale (e.g. utilization percentage versus number of (M)Bytes). Selecting the dimensions and corresponding metrics should be done for each individual case as this depends on the product under study. Note that the dimensions are not orthogonal, e.g. higher resource utilization results in increased power consumption.

Following the first goal presented for the radar chart, we should aim for a standardized means to present the measurements. Looking at the diversity of the list, one of the few options to determine a standardized score on each dimension is to use ranges. With regard to resource usage, a minimum resource usage can be determined in the situation where the hardware is idle and a maximum where the hardware is stressed to its maximum capacity [36]. The available resource, i.e. the margin between the minimum and maximum resource usage, forms the range. Note that the range should be determined individually for each performance metric. When an activity is performed using the software, the required resources can be divided by the range. This transforms measurements to a value between 'zero' and 'one' for that specific metric.

There is however a downside to working with ranges, as not every aspect can be expressed using a range. The execution time, for example, could have an infinite maximum, i.e. run as long as required without limitations. As such, we suggest to exclude these aspects from the chart itself and instead report these separately. For example, the units of work [15] to create a workload model are described separately to correctly interpret the measurements and the context in which they were found. We continue our work using the range method.

A final aspect is the order in which the axes are included in the radar chart. Using the same data in a different order can result in a difference of up to 300% [35], posing a threat to the third and fourth goal identified with the radar chart. We take this issue into account in the next section where we calculate a score for SEC.

C. Calculating the RUS

Continuing on the path of the radar chart, following goal three, we are able to obtain an objective performance measure by calculating the surface of the chart. This calculation is described by Mosley and Mayer [35] as the Surface Measure of Overall Performance (SMOP) and is calculated using Eq. 1.

$$\text{SMOP} = ((P_1 \cdot P_2) + \ldots + (P_n \cdot P_1)) \cdot \sin(\frac{\pi}{n}) \qquad (1)$$

In the equation, the P-values represent the axes of the radar chart. The resulting number represents the surface of the figure created by all of the connected dots on the chart, i.e. the grey surface in Fig. 2.

The problem with Eq. 1 is that the axes are ordered implicitly. As there is no clear order between the different measures the axes represented, ordering them differently results in different values for the surface. As we do not have an explicit, clear order for the measures, a solution is sought by calculating the average surface based on all possible surfaces. Rephrased, we consider all possible relations between the axes. Instead of calculating for each possible order the corresponding surface, we observe that each possible triangle of axes is taken into account an equal number of times. Hence, calculating the surface for all different triangle suffices. Translating this to an equation results in Eq. 2:

$$\text{SMOP} = \sin(\frac{\pi}{n})(\sum_{i=1}^{n}\sum_{j=1}^{n}(P_i \cdot P_j)) \qquad (2)$$

Observe that each score is multiplied by a constant factor. As we want to use the score for comparing different solutions, this constant can be left out. However, in its current form we see that axes could be paired with themselves, and that all pairs are counted twice (i.e. the symmetrical pairs $P_i \cdot P_j$ and $P_j \cdot P_i$). Taking these elements into account, the equation can be simplified as follows:

$$\text{RUS} = \sum_{i=1}^{n}\sum_{i<j}^{n}(P_i \cdot P_j) \qquad (3)$$

A side effect of excluding the constant factor is that we do not longer calculate the surface of the chart, but rather a *score*

based on the relation between the axes. As a result we are able to include non-standardized dimensions (e.g. execution time) in the equation that are not included in the radar chart. We labeled the score as the Resource Utilization Score.

D. Weight Factor

With the axes for benchmarking SEC identified and the ability to calculate a score, one important limitation remains that should be addressed: weighting the contribution of the different indicators. In our case this limitation counts for the performance metrics, but extends to the level of dimensions (i.e. axes). Concerning the first, we can only argue that the right performance metrics should be used to determine the scores on the respective axes. A hard disk, for example, could be characterized using the 'Disk I/O per second' and the '# Mb per second' metrics [9].

With regard to the axes we acknowledge that some combinations could be considered more important than others and have a bigger influence on EC [37] depending on the context. In large scale infrastructures, for example, memory plays an important role. As such, specific combinations that include memory could be argued have a greater contribution to the RUS. As these combinations are context dependent, we introduce a weight to the different ax-combinations, which results in the following score:

$$RUS = \sum_{i=1}^{n} \sum_{i<j}^{n} W_{i,j}(P_i \cdot P_j) \tag{4}$$

This score (Eq. 4) provides us with the RUS to characterize the resource utilization in relation to SEC. The weight factor can be used to reduce or amplify the effect of certain combinations of measures. In-depth analysis of the performance data, for example through a regression model [10], can help in determining the weight factors.

IV. Experiment Design

To evaluate the RUS, an experiment was performed to select the most resource efficient algorithm to calculate the first N decimals of π from an application that provides many algorithms to calculate π. Additionally, as a more practical evaluation, we apply the RUS to an already available dataset [9]. In this section we describe the setup of the experiment for which we followed the guidelines provided in [38]–[40].

A. Experiment environment

For our experiment a test environment was prepared consisting of an application system, a logging system, and a measurement device (Fig. 3). The application system is the test hardware on which the software product is to be installed and as such the system to monitor. The logging system collects data from multiple sources and provides task instructions to the application system. Finally, the measurement device, a WattsUp? Pro (WUP)[1], is used to measure the power drawn by the application system and calculate the SEC. Since the WUP is a separate device, the energy usage of the application system was not influenced by its measurements.

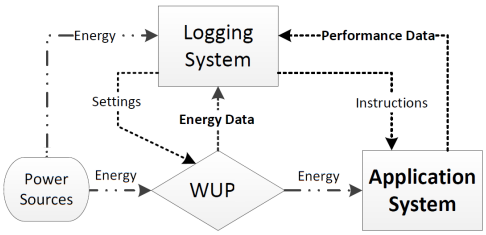

Figure 3. Experiment setup

In total three different application systems were included in the experiment, a laptop, desktop and a server, each representing a different computer class. For a system to be included in the experiment the device had to run the Microsoft Windows 7 *Professional* (or higher) operating system and be equipped with a multi-core Intel processor. These requirements provided us with systems that are capable of remote performance monitoring and are representative for modern systems in terms of computational capabilities. Details of the selected systems are shown in Table I.

Performance measurements were collected using the Windows Performance Monitor[2] (Perfmon). Perfmon enables remote performance monitoring of systems with a one second interval in between measurements and is freely available with the Windows operating system.

B. Test Application

To simulate activity, we used *Systester* (version 1.5.1)[3]; an application that calculates Pi decimals using the *The Quadratic Convergence of Borwein* and *Gauss-Legendre* algorithms. For the first algorithm both a single- and multi-core variant was available This enabled us to not only compare the difference between the two algorithms, but also between a single- and multi-core configuration. In order to have controllable runs the choice was made to calculate $8 \cdot 10^6$ Pi decimals per run.

For the actual experiment the remotely executable command line version of Systester was used, i.e. from a batch script using the logging system. In addition, a modified version of Systester was compiled where the application waits five seconds after initiating and before ending the process. The presence of this five second interval allowed Perfmon to collect all data related to a task and made it easier to identify the specific runs during processing, thereby directly contributing to the quality of the data and forthcoming analysis.

C. Metrics and Utilization Ranges

Visualizing measurements on a radar chart requires the metrics to be expressed on a standardized scale. To do so, we require the minimum (zero) and maximum (one) resource utilization figures which allows us to express measurements as a value on this continuum. The idle measurements (minimum)

[1]https://www.wattsupmeters.com/secure/products.php?pn=0&wai=0

[2]https://technet.microsoft.com/en-us/library/cc749154.aspx

[3]http://systester.sourceforge.net/

Table I
SPECIFICATIONS OF THE APPLICATION SYSTEMS.

Property	System		
	Laptop	Desktop	Server
Brand/model	ASUS F3JA	*Custom PC*	HP DL380 G5
Processor	Core 2 Duo T7200	Core 2 Duo E6750	Intel Xeon E5335
FSB / TDP	667 / 34	1333 / 65	1333 / 80
Chipset	Intel i945PM	Intel P35	Intel 5000P
Memory	2GB DDR2	2GB DDR2	4GB EDO
Operating System	Windows 7 Professional (32 bit) Servicepack 2	Windows 7 Professional (32 bit) Servicepack 2	Windows server 2008 (64 bit) Servicepack 1

were performed by leaving the application systems idle (without going to a sleep state) for at least 30 hours and monitoring the resource utilization and power consumption during this period. To determine the maximum resource utilization figures the application systems were stressed to their maximum capacities using HeavyLoad[4].

Based on the characteristics of Systester, the following metrics were selected to create a radar chart:

- **CPU**: '% CPU time'. The maximum utilization is 100% per core adding up to a percentage above 100% for multi-core systems. The range was determined with the '% CPU time' while idle and using HeavyLoad.
- **Memory**: 'Available bytes'. The number of bytes that is available of which the value decreases as processes require memory. The maximum is the available bytes while idle which also indicates the range for this metric.
- **Disk**: '% disk idle time'. The time that the disk was idle. The maximum utilization for the hard disk is 100% and its range is found by subtracting the '% idle time' of an idle system from this 100%. While the '% disk time' metric can also be used, this metric exaggerates[5] disk utilization.
- **Power consumption**: The 'power consumption' (in Watt) by the system while performing a run. The range was determined per system by measuring the 'power consumption' while idle and while using HeavyLoad.

Given the nature of the application we decided to exclude network metrics. Note that the actual SEC is calculated using the WUP measurements and stems from a different source than the performance measurements.

To calculate the RUS, the standardized metrics of the radar chart will be combined with the non-standardized 'execution time' metric. The 'execution time' is defined as the time required to perform a specific task and could be a determining aspect for SEC [9]. In our experiment the execution time is the time for Systester to calculate $8 \cdot 10^6$ Pi decimals.

D. Experiment protocol

To actually perform the experiment a protocol was followed containing every activity required to perform a series of runs. A run is one time for the application to calculate $8 \cdot 10^6$ Pi decimals plus the five seconds before and after performing this task. A series can be configured to include multiple runs. For each series a script was used, *PiBatch*, which automates monitoring with PerfMon, optionally includes rebooting the system, and performs a specified number of runs. The script minimizes human interference, provided that the following preparations are made:

- Install PsTools[6] for executing commands remotely.
- Install software to remotely manage the WUP.
- Remove the battery from the laptop to eliminate battery charging/discharging effects.
- Configure Windows power settings and disable unnecessary services (e.g. Windows Search and Update).
- Configure Perfmon data collector set.

Additionally, the effect of rebooting the application systems was investigated. In a small experiment we found that a system was 'unpredictable', i.e. random active processes, in the first 15 minutes after rebooting. The PiBatch script takes this into account by waiting at least 15 minutes before starting the first run of a series. This resulted in the following protocol:

- Clear WUP meter data and test connections.
- Configure and initiate PiBatch script .
- Collect data from PerfMon and WUP.

At the time of the experiment, rebooting was made optional in the script as rebooting the server appeared not possible with a virtual machine running. The virtual machine was running on one single, dedicated server and was isolated from other infrastructural facilities ensuring that the only hardware that is affected is the hardware being measured. To get a representative data set, we decided to continue until at least thirty clean measurements per combination were obtained.

E. Post-processing the Measurements

After performing a series of runs, post-processing was required before analyzing the data.

Determine run execution time; The runs appeared of variable length and hence we needed to determine the exact execution time for each run using the '%CPU Time' of the application process (provided by PerfMon). The execution interval started when the '%CPU Time' was more than zero and ended when it went back to zero again.

Synchronize WUP and Perfmon timestamps; Since the WUP and PerfMon data stemmed from separate sources, the

[4]http://www.jam-software.com/heavyload/

[5]https://technet.microsoft.com/en-us/library/cc938959.aspx

[6]https://technet.microsoft.com/en-us/sysinternals/bb896649.aspx/

Table II
THE RUS FOR EACH COMBINATION (LOWER IS BETTER).

	Laptop	Desktop	Server
Borwein, single-core	393.44	421.15	354.57
Borwein, multi-core	358.87	405.73	377.52
Gauss-Legendre	158.53	194.56	147.74

Table III
EC CONSUMPTION FIGURES FOR EACH COMBINATION IN JOULE.

	Laptop	Desktop	Server
Borwein, single-core	17,989	26,092	103,165
Borwein, multi-core	14,724	20,555	82,204
Gauss-Legendre	7,442	10,891	44,870

timestamps of the measurements required synchronization. The start of an interval in the WUP measurements, i.e. an increase in power drawn, was matched to the moment of initiation of the associated processes in the PerfMon data. Cross-checking on corresponding end points ensured that the synchronization was correct.

Assess quality of runs; Despite our efforts to control any effects that could influence the experiment, we observed activity unrelated to Systester during the experiment. First, we used the '% CPU time' on process level to check whether a system was solely processing tasks related to Systester during a run. In addition we monitored the EC for discrepancies as the performance measurements could not always explain increases in power consumption. Runs including showing odd patterns were excluded from further processing.

V. EVALUATING THE RUS

The results of the experiment are summarized in Fig. 4 which shows the radar charts for each combination of the three systems and Systester options. The charts show the average score on each metric for the specific configuration, e.g. the laptop on average used 69% of the available CPU resources with the multi-core quadratic convergence of Borwein. In total 32 runs were performed for each algorithm on the laptop, which were are all clean. On the desktop 34 runs were performed with the Gauss-Legendre algorithm and 32 runs with both Borwein algorithms, providing 31 clean runs per algorithm. The server appeared the most problematic system where we performed 80, 72 and 55 runs for the Gauss-Legendre, single-core and multi-core Borwein algorithms to obtain 42, 42 and 55 clean runs respectively. Given the instability, we decided to obtain at least 40 clean runs for the server.

Since the execution time was not suitable to express on an axis, we provided this information alongside the corresponding radar chart. At a glance we can conclude that the Gauss-Legendre algorithm is the fastest option of the three, and that the multi-core variant of the Borwein algorithm is faster than its single-core variant. Surprisingly we find that the server, despite its computational capacity, on average is at least 30 seconds slower compared to the other systems with the Gauss-Legendre algorithm. A trend that also shows with the other algorithms. We were not able to find the cause of this discrepancy, but we argue these figures could be typical for the server and associated hardware possibly in combination with the multi-core capabilities of Systester itself.

The impact of the execution time can be made clear using the actual SEC figures (Tbl. III) on the account of Systester. Although the metrics indicate a fairly similar resource utilization pattern across machines, in terms of absolute SEC we find that the server consumes more energy. The same holds for the desktop compared to the laptop; similar scores, higher SEC by the desktop in absolute terms. If we consider the SEC findings in light of the radar charts we solidify the argument on adding the execution time to the visualization.

Looking at the dimensions themselves we find that in this particular case there seems to be a relation between the scores on the CPU and power dimensions, i.e. an increase on the CPU dimension pairs with an increase on the power dimension. Also, as expected, the CPU scores are highest with the multi-core Borwein algorithm, but do not double in comparison to the single-core version. The difference between the Borwein measurements could be an indication of the potential for multi-core (i.e. multi-threaded) applications.

With regard to the memory and disk metrics, the laptop and server charts indicate minimal impact on the memory and disk dimensions. However, the radar charts clearly indicate a different situation for the disk utilization by the desktop. While we find the high disk utilization for the desktop peculiar, we cannot attribute this utilization to Systester as the other systems do no exhibit this behavior. Based on the information a further analysis can be performed on the desktop.

A. RUS Scores

The corresponding RUS for each combination is provided in Tbl. II. The RUS was calculated using the standardized scores of the performance metrics and the execution time in seconds (non-standardized). Hence, the scores are larger than 'one'. As there were no indications to prefer a specific dimension over the others we set the weight factor for all pairs of dimensions to 'one'. Important to notice is that a lower score means that a specific combination scores better; i.e. requires less resources.

Comparing the RUS with the EC figures (Tbl. III) we find that in general lower RUS scores are accompanied by lower EC figures. The only exception is with the single- and multi-core Borwein algorithms on the server. Looking at the radar charts we find that the multi-core variant shows higher utilization scores on the power and CPU dimension, an increase that is also visible on the other systems. In relation to the other systems we can only conclude that the difference in execution time is enough to offset the EC figures but not the RUS. From a practical perspective, the single-core variant could be preferred above the multi-core variant when trade-offs should be made (e.g. when resources are shared).

If we solely use the RUS scores to choose an algorithm and platform, the decision would be to run the Gauss-Legendre algorithm on the server. However, based on the EC we should actually prefer this algorithm on the laptop or, if we favor speed, on the desktop. This observation learns that the RUS

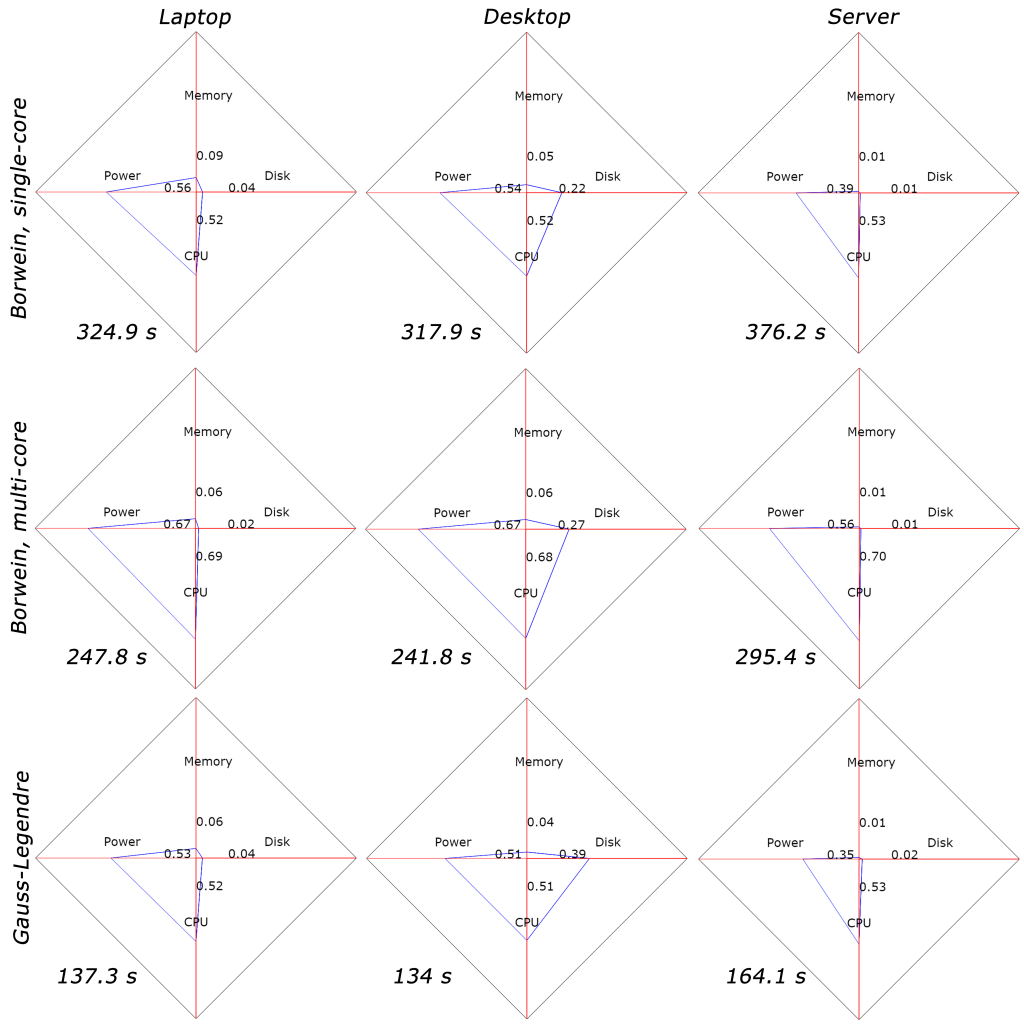

Figure 4. The collection of radar charts showing the resource utilization on each dimension and execution time for the nine investigated combinations.

should be considered complementary to the EC and that we cannot use the RUS to compare across classes of systems.

B. RUS for a Commercial Software Product

As an additional evaluation we applied the theory to a commercial software product (Document Generator) with the instructions to generate 5000 documents [9]. The metrics for the radar chart were '% CPU time' (CPU), 'available MBytes' (memory), '% disk idle time' (hard disk), 'power consumption' (in Watt) and the 'total Bytes per second' (network). Compared to Systester the network metric was included and and its range was determined using Lan Speed Test[7].

In this case an architectural change was applied to make Document Generator multi-threaded. The resulting decrease in CPU Utilization, i.e. from 49% to 19.2%, lowered the EC per document with 67.1% This finding is visible in the radar charts (Fig. 5) where a decrease in the the CPU and power utilization can be observed. A minimal increase in utilization of the disk and memory was found, whereas the network utilization remains unchanged. Especially the CPU utilization, or more specifically the division of the workload between CPU cores [9], seems decisive for the power dimension.

The RUS scores (Tbl. IV) were calculated using the execution time and appear to be in line with the EC measurements; i.e. a lower score means less SEC. This finding possibly suggests that the utilization patterns after adjusting the software are more 'natural' to the system.

VI. Discussion

In this research we investigated the possibility for a RUS to express resource utilization in relation to the SEC. The constructed score is based on those dimensions that are deemed relevant for the software and is flexible to be adjusted depending on the product and the environment in which it will be executed. Initial evaluation showed promising results to engage in green software practices. There are, however, several limitation to our work which we discuss below.

Hardware dependency; Like EC, the RUS is dependent on the hardware that the software is executed on. Although the measurements are standardized, the ranges themselves showed

[7]http://totusoft.com/lanspeed/

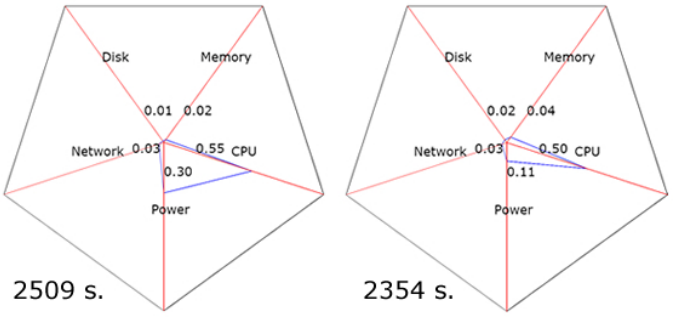

Figure 5. The radar charts for the Document Generator software product before (left) and after (right) making the software multi-threaded [9].

considerable differences across systems. Comparing the RUS and EC figures led to the insight that the RUS cannot be used to compare a software product across classes of systems. Additionally, it can be impossible to determine the ranges in environments with 'unlimited' resources (e.g. cloud data centers). Different benchmarks should be found when this is the case, for example benchmark release of software to one another to visualize the effect of software changes.

Visualization; The RUS is based on the theory related to the radar chart. Even though we investigated different theories, other options could exist that better fit the purpose. For example, theories that better consider the relation between dimensions or express a score based on the dimensions.

Robustness; The RUS was successfully applied to both a synthetic (Systester) and commercial (Document Generator) application, which shows the generic ability of the theory to be applied. Although we are confident in the validity of our results, more applications of RUS are required to prove or disprove the robustness of the RUS.

Weight factor; The weight factor for calculating the RUS is a topic that still requires further investigation. In our experiment we could not motivate a higher score on one combination of dimensions over the other and decided to set the weight factor to '1' for each combination. However, other situations might require a thorough investigation to determine the correct weight factors.

A. Experiment Limitations

Despite our best efforts, there are limitations to the experiment as described:

Windows processes; Thirty minutes after rebooting we observed an increase in activity for an unspecified period of time. The cause is unknown, but we assume that Windows-related processes are triggered by a timed mechanism which we cannot control. However, we did not find significant differences between runs executed after twenty or 200 minutes and between Windows 7 and Windows Server 2008.

Measurement interval; WUP and Perfmon perform measurements with a one second interval, while computers process millions of instructions per second. Although we argue our measurements are sufficient for our purposes, we acknowledge the fact that data is lost with the instruments at hand.

Table IV
THE EC (IN JOULE) AND RUS FOR THE DOCUMENT GENERATOR SOFTWARE PRODUCT BEFORE AND AFTER CHANGING THE SOFTWARE.

	EC	RUS
Single-threaded	17,560	2,313.09
Multi-threaded	5,782	1,644.49

Room temperature; Of the three systems the server was the only one situated in a climate-controlled data center and as a consequence we can only guarantee identical conditions for this system. Although we tried to maintain consistency, we acknowledge the fact that, among others, room temperature could have influenced our measurements. We consider the insignificant differences found between measurements as a confirmation that the influence in our experiment was limited.

VII. CONCLUSION

In this paper we propose a metric to effectively communicate resource utilization measurements for a software product in relation to EC. The metric should be easy to understand, reported uniformly and clearly communicate key findings. We consider the viewpoints of multiple stakeholders wanting to address the sustainability of their software product through green software practices, and posed the following **research question**: 'How can we effectively express the resource utilization for executing a software product in relation to the SEC?'. We provide an answer by constructing the RUS.

Following the goals of the of radar chart, the RUS delivers a single score based on selected dimensions and performance metrics. To calculate the RUS, the equation to calculate the surface of a radar chart was transformed into one that considers the relation between dimensions. A weight factor is added that enables stakeholders to determine the importance of each pair of dimensions. As the measurements can be expressed on a standardized scale they can be interpreted more easily, do not require knowledge on the individual metrics and can be compared between software applications. Additionally, the radar chart provides a means to visualize the measurements which helps to identify key findings.

Evaluating the RUS with two different datasets, showed that the RUS should be considered complementary to the EC and the execution time related to a software product. In general a lower RUS corresponds to a lower EC consumption figure, but with the server a case was also found where a lower RUS was accompanied by a higher EC. In these situations a trade-off should be made, like with quality attributes, favoring the aspect that is considered more important in a specific context. A limitation of the RUS found in its inability to be compared across systems of different classes.

Based on the work presented in this paper, we identify several direction for future research. First is to investigate the RUS more thoroughly. For example, the RUS could be used to compare between systems within the same class. Also a (standardized) means to determine the weight factor could aid in the RUS' acceptance. A second direction is to investigate the positioning of the RUS in relation to more

the generic eco-labels for the ICT domain. A final direction is to use RUS to create awareness on green software during the development process. By showing the impact of software development activities, software developers are enabled to address sustainability issues that might arise.

ACKNOWLEDGMENT

We would like to thank Fabiano Dalpiaz and Garm Lucassen and the reviewers of ICT4S for their valuable feedback to improve the paper.

REFERENCES

[1] S. Murugesan, "Harnessing green it: Principles and practices," *IT Professional*, vol. 10, no. 1, pp. 24–33, Jan 2008.

[2] Y. Sun, Y. Zhao, Y. Song, Y. Yang, H. Fang, H. Zang, Y. Li, and Y. Gao, "Green challenges to system software in data centers," *Frontiers of Computer Science in China*, vol. 5, no. 3, pp. 353–368, 2011.

[3] K. Grosskop and J. Visser, "Identification of application-level energy optimizations," in *Proceedings of ICT for Sustainability (ICT4S)*. Atlantis Press, 2013, pp. 101–107.

[4] A. Noureddine, R. Rouvoy, and L. Seinturier, "Monitoring energy hotspots in software," *Automated Software Engineering*, pp. 1–42, 2015.

[5] A. Hindle, "Green mining: a methodology of relating software change and configuration to power consumption," *Empirical Software Engineering*, pp. 1–36, 2013.

[6] S. Jansen, S. Brinkkemper, J. Souer, and L. Luinenburg, "Shades of gray: Opening up a software producing organization with the open software enterprise model," *Journal of Systems and Software*, vol. 85, no. 7, pp. 1495–1510, 2012.

[7] C. Pang, A. Hindle, B. Adams, and A. E. Hassan, "What do programmers know about the energy consumption of software?" *PeerJ PrePrints*, vol. 3, p. e1094, 2015.

[8] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*, ser. SEI Series in Software Engineering. Pearson Education, 2012.

[9] E. A. Jagroep, J. M. E. M. van der Werf, R. Spauwen, L. Blom, R. van Vliet, and S. Brinkkemper, "An energy consumption perspective on software architecture," in *Software Architecture*, ser. LNCS, no. 9278. Springer, 2015, pp. 239–247.

[10] E. A. Jagroep, J. M. van der Werf, S. Brinkkemper, G. Procaccianti, P. Lago, L. Blom, and R. van Vliet, "Software energy profiling: Comparing releases of a software product," in *Proceedings of the 38th International Conference on Software Engineering Companion*, ser. ICSE '16. New York, NY, USA: ACM, 2016, pp. 523–532.

[11] F. Chasin, "Sustainability: Are we all talking about the same thing state-of-the-art and proposals for an integrative definition of sustainability in information systems," in *Proceeding of ICT for Sustainability (ICT4S)*. Atlantis Press, 2014, pp. 342–351.

[12] P. Lago, S. A. Koçak, I. Crnkovic, and B. Penzenstadler, "Framing sustainability as a property of software quality," *Commun. ACM*, vol. 58, no. 10, pp. 70–78, sep 2015.

[13] P. Lago, R. Kazman, N. Meyer, M. Morisio, H. A. Müller, F. Paulisch, G. Scanniello, B. Penzenstadler, and O. Zimmermann, "Exploring initial challenges for green software engineering: summary of the first GREENS workshop, at ICSE 2012," *ACM SIGSOFT Software Engineering Notes*, vol. 38, no. 1, pp. 31–33, 2013.

[14] G. Procaccianti, P. Lago, and G. A. Lewis, "Green architectural tactics for the cloud," in *Software Architecture (WICSA), 2014 IEEE/IFIP Conference on*, April 2014, pp. 41–44.

[15] G. Kalaitzoglou, M. Bruntink, and J. Visser, "A practical model for evaluating the energy efficiency of software applications," in *Proceedings of ICT for Sustainability (ICT4S-14)*. Atlantis Press, 2014.

[16] C. Sahin, M. Wan, P. Tornquist, R. McKenna, Z. Pearson, W. G. J. Halfond, and J. Clause, "How does code obfuscation impact energy usage?" *Journal of Software: Evolution and Process*, 2016.

[17] ISO, "Systems and software engineering – systems and software quality requirements and evaluation (SQuaRE) – system and software quality models," International Organization for Standardization, Geneva, Switzerland, ISO 2510:2011, 2011.

[18] N. Rozanski and E. Woods, *Software Systems Architecture: Working with Stakeholders Using Viewpoints and Perspectives*. Addison-Wesley, 2011.

[19] S. Barlowe and A. Scott, "O-charts: Towards an effective toolkit for teaching time complexity," in *Frontiers in Education Conference (FIE), 2015. 32614 2015. IEEE*, Oct 2015, pp. 1–4.

[20] A. Kansal, F. Zhao, J. Liu, N. Kothari, and A. A. Bhattacharya, "Virtual machine power metering and provisioning," in *Proceedings of the 1st ACM Symposium on Cloud Computing*, ser. SoCC '10. New York, NY, USA: ACM, 2010, pp. 39–50.

[21] C. Ebert and S. Brinkkemper, "Software product management - an industry evaluation," *Journal of Systems and Software*, vol. 95, no. 0, pp. 10 – 18, 2014.

[22] A. Kipp, T. Jiang, M. Fugini, and I. Salomie, "Layered green performance indicators," *Future Generation Computer Systems*, vol. 28, no. 2, pp. 478 – 489, 2012.

[23] K. Lundfall, P. Grosso, P. Lago, and G. Procaccianti, "The green practitioner: A decision-making tool for green ict," in *Proceedings of ICT for Sustainability (ICT4S)*. Atlantis Press, 2015, pp. 74–81.

[24] D. Magalhães, R. N. Calheiros, R. Buyya, and D. G. Gomes, "Workload modeling for resource usage analysis and simulation in cloud computing," *Computers & Electrical Engineering*, vol. 47, pp. 69–81, 2015.

[25] M. Poess and R. O. Nambiar, "Energy cost, the key challenge of today's data centers: a power consumption analysis of tpc-c results," *Proceedings of the VLDB Endowment*, vol. 1, no. 2, pp. 1229–1240, 2008.

[26] K. Singh, M. Bhadauria, and S. A. McKee, "Real time power estimation and thread scheduling via performance counters," *ACM SIGARCH Computer Architecture News*, vol. 37, no. 2, pp. 46–55, 2009.

[27] A. Beloglazov, J. Abawajy, and R. Buyya, "Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing," *Future Generation Computer Systems*, vol. 28, no. 5, pp. 755 – 768, 2012, special Section: Energy efficiency in large-scale distributed systems.

[28] R. Koller, A. Verma, and A. Neogi, "WattApp: an application aware power meter for shared data centers," in *Proceedings of the 7th international conference on Autonomic computing*. ACM, 2010, pp. 31–40.

[29] P. Somavat, V. Namboodiri *et al.*, "Energy consumption of personal computing including portable communication devices," *Journal of Green Engineering*, vol. 1, no. 4, pp. 447–475, 2011.

[30] R. Carpa, O. Gluck, L. Lefevre, and J.-C. Mignot, "Improving the energy efficiency of software-defined backbone networks," *Photonic Network Communications*, vol. 30, no. 3, pp. 337–347, 2015.

[31] J. Espadas, A. Molina, G. Jimnez, M. Molina, R. Ramrez, and D. Concha, "A tenant-based resource allocation model for scaling software-as-a-service applications over cloud computing infrastructures," *Future Generation Computer Systems*, vol. 29, no. 1, pp. 273 – 286, 2013, including Special section: AIRCC-NetCoM 2009 and Special section: Clouds and Service-Oriented Architectures.

[32] E. Kern, M. Dick, S. Naumann, and A. Filler, "Labelling sustainable software products and websites: Ideas, approaches, and challenges," in *Proceedings of ICT for Sustainability (ICT4S)*. Atlantis Press, 2015, pp. 82–91.

[33] G. Procaccianti, P. Lago, and G. A. Lewis, "A catalogue of green architectural tactics for the cloud," in *Maint. and Evol. of Service-Oriented and Cloud-Based Systems (MESOCA), 2014 IEEE 8th Int'l Symp. on the*, Sept 2014, pp. 29–36.

[34] H. Schütz, S. Speckesser, and G. Schmid, "Benchmarking labour market performance and labour market policies: Theoretical foundations and applications," WZB Discussion Paper FS I 98-205, 1998. [Online]. Available: http://hdl.handle.net/10419/43918

[35] H. Mosley and A. Mayer, "Benchmarking national labour market performance: A radar chart approach," WZB Discussion Paper FS I 99-202, 1999. [Online]. Available: http://hdl.handle.net/10419/43952

[36] G. Bekaroo, C. Bokhoree, and C. Pattinson, "Power measurement of computers: analysis of the effectiveness of the software based approach," *Int. J. Emerg. Technol. Adv. Eng*, vol. 4, no. 5, pp. 755–762, 2014.

[37] T. Vogelsang, "Understanding the energy consumption of dynamic random access memories," in *2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture*, Dec 2010, pp. 363–374.

[38] N. Juristo and A. M. Moreno, *Basics of Software Engineering Experimentation*, 1st ed. Springer Publishing Company, Incorporated, 2010.

[39] P. Runeson and M. Höst, "Guidelines for conducting and reporting case study research in software engineering," *Empirical software engineering*, vol. 14, no. 2, pp. 131–164, 2009.

[40] C. Wohlin, P. Runeson, M. Hst, M. C. Ohlsson, B. Regnell, and A. Wessln, *Experimentation in Software Engineering*. Springer Publishing Company, Incorporated, 2012.