A Practical Model for Evaluating the Energy Efficiency of Software Applications

Georgios Kalaitzoglou
Sociallgreen
Thessaloniki, Greece
gkalaitz@sociallgreen.com

Magiel Bruntink
University of Amsterdam
Amsterdam, the Netherlands
m.bruntink@uva.nl

Joost Visser
Software Improvement Group
Amsterdam, The Netherlands
Radboud University Nijmegen
Nijmegen, the Netherlands
j.visser@sig.eu

***Abstract*—Evaluating the energy efficiency of software applications currently is an ad-hoc affair, since no practical and widely applicable model exists for this purpose. The need for such an evaluation model is pressing given the sharp increase in energy demand generated by the ICT industry. In particular, we need to get in control of our software applications since they play a key role in driving the consumption of energy. This paper proposes ME^{3}SA, a Model for Evaluating the Energy Efficiency of Software Applications. ME^{3}SA provides a practical breakdown of energy efficiency into measurements that can be applied to software applications in relation to the quantity of work they deliver. This approach makes it possible to measure and control energy efficiency similar to other software qualities such as performance efficiency or maintainability.**

Furthermore, we report on a case study in which the model was applied to an operational software system of the Software Improvement Group (SIG), a software advisory firm based in the Netherlands. The case study provides evidence that the proposed model is able to identify energy consumption hotspots and efficiency bottlenecks within software applications.

***Index Terms*—Software measurement, Green software**

I. Introduction

Energy consumption by IT systems is increasing to substantial levels. Overall electrical energy consumption by IT was estimated between 1.1% and 1.5% in 2010 [1], and in certain regions data center consumption alone was estimated at 10% in 2011 [2].

In recent years, awareness has grown that energy optimization of IT systems is needed not only at the level of hardware and data center facilities but also at the level of software [3]. In previous work, one of the authors has developed a small set of indicators for the energy-efficiency of IT services as externally observable [4] as well as some more in-depth analysis techniques for identifying energy-optimization opportunities in application software [5].

In this paper, we take a further step by developing an evaluation model for software applications that combines results from in-depth analysis into an overall rating for energy-efficiency. The purpose of this model, named ME^{3}SA, is to allow comparison of energy-efficiency between applications but also to provide strong clues on which aspects of which parts can improved.

A. Research challenges and approach

Evaluating the energy efficiency of software applications is certainly a challenging research topic. We identified the following core challenges by studying the available literature:

Diversity in software applications. Evaluating energy efficiency requires a definition of "work". Before one can speak of energy efficiency of software, one first needs to agree on what it is that software needs to be energy efficient at. The work that software does needs to be defined such that it can be measured and related to its energy consumption. The challenge here lies in the diversity of software applications and their widely different functionalities.

Information hiding. Software energy efficiency essentially refers to the efficient use of computational resources, i.e., hardware. A challenge arises from the prevalent use of design principles like information hiding and abstraction. From the viewpoint of software applications, the hardware is typically hidden under several layers of abstraction, such as middleware, operating systems and drivers, which were designed to be as opaque as possible.

Lack of standards. Unlike other quality aspects of software, energy efficiency is not covered by international standards for software quality (e.g. ISO/IEC 25010 [6]). This means that there is no commonly accepted definition framework to provide the foundation of an evaluation model for energy efficiency of software.

In the context of these challenges we propose a practical evaluation model that could serve as a first step towards a standardized method to evaluate the energy efficiency of software applications. The model is taking a related software quality, *Performance Efficiency* (as defined in ISO/IEC 25010 [6]), as a starting point, leading to an conceptual break-down into three aspects: *Energy Behavior*, *Capacity* and *Utilization*. The scope of the current work is summarized by the measurement goal we set for the model: identifying energy consumption hotspots and energy efficiency bottlenecks on a software component level, with reasonable accuracy and with reasonable effort. The model should not be specific to a particular application domain or infrastructure.

The model design follows a Goal-Question-Metric (GQM [7]) approach in which the high-level measurement

goal is translated into several evaluation questions, followed by an operationalization of the questions using metrics. The input data needed for the metrics consist of utilization information collected on the host hardware, timing and duration information on units of work performed, and finally information on (actual or estimated) power usage by hardware components given a utilization level.

The proposed model is applied in a case study of an operational software application of the Software Improvement Group (SIG), a software advisory firm based on Amsterdam, the Netherlands. The application under study is the so-called Software Monitor, which underlies several core services offered by the SIG. During the case study a measurement period of 1 week was used to collect all data, followed by an analysis and evaluation phase.

B. Contributions and structure

In Section II we discuss research related to our approach. In the subsequent two sections we provide our main contributions: the design of ME^{3}SA, a practical model for evaluating the energy efficiency of software applications (Section III); and the results of a case study that applied ME^{3}SA to an operational software application (Section IV). Discussion and evaluation is provided in Section V. Finally, Section VI discusses future work and concludes the paper.

II. Related work

We focus on presenting related work on evaluation methods and monitoring tools. An overview regarding energy efficiency optimization approaches targeting the software stack can be found in previous work of one of the authors [8].

Kern et. al. propose the GREENSOFT model, which is "a first approach to developing a quality model for green and sustainable software" [9]. The GREENSOFT model shares many of the goals as the model we are proposing here, but still lacks a concrete definition and application of metrics to make the approach practical. We also follow the GREENSOFT approach in the sense of taking existing software quality standards, i.e. ISO 25010 (SQuaRE) [6], as the starting point.

The data center domain has been the cradle of energy efficiency research since it is has been heavily affected by energy efficiency related issues. As a result, several metrics, frameworks and best-practices have been proposed in that context with notable contributors being the Green IT Promotion Council, the Uptime Institute, the Nomura Research institute, the Emerson Corporation, the Green Grid and the GAMES framework which was incorporated in the EU Projects Games [10] [11] and CoolEmAll [12]. Other initiatives a concise presentation of their proposed metrics can be found in [13].

All of the preceding approaches suffer from various limitations with the most obvious being their confinement to the data center domain. Consequently, they are unsuitable for assessing heterogeneous system types as most of their metrics and the underlying methodology do not produce results that can be comparable and meaningful across a wide spectrum of modern computational systems. For instance, the DCeP metric proposed by the Green Grid, is defined as the ratio between the work output of a data centre in bytes to the total energy consumption of the data centre in kWh [10]. Consequently, it is only applicable and meaningful in that context. Furthermore, some require extensive instrumentation or consist of a large number of metrics, rendering them less practical. The approach proposed by Kipp et al. [11] for instance, requires the calculation of 30 indicators, the collection of performance and energy data across a wide spectrum of system nodes and the aggregation and correlation of values across four levels.

In an industrial context, Intel has published extensive documentation on energy efficiency practices at many levels of the energy problem stack (data centers, application development etc.) [14] [15]. They developed the Energy Checker SDK that can be used to relate energy consumption to useful units of work. This tool offers an API that can be used for importing and exporting counters, which can track specific events that other applications can use to adjust their energy behavior. Nevertheless this approach requires extensive instrumentation of the application code.

Microsoft has also been particularly active on the topic [16] [17] and has produced Joulemeter, a tool that is able to measure the energy consumption of software applications running on Windows platforms. This application uses a power model and performance counters to estimate energy consumption and attribute it to specific processes running on a given system. Although powerful, this approach is limited to specific platforms and to non-distributed software systems.

The Software Improvement Group has developed Green Software Scans [5], which constitute a practical approach in evaluating the energy efficiency of application software in situ. This approach so far lacks an underlying evaluation model with associated rating scheme and depends heavily of expert opinion and knowledge.

Energy monitoring and profiling has also been a particularly active field in non-industrial domains, as evidenced by tools such as Powerscope [18] and pTop [19].

III. Design of the ME^{3}SA model

A. Goal and Subgoals

To structure our design of ME^{3}SA, we applied the Goal-Question-Metric (GQM) approach [7]. This approach defines a measurement model in three levels. First, GQM starts by stating a measurement goal. Secondly, at an operational, level questions are raised that must be answered in order to reach that goal. Finally, this approach reaches the quantitative level, by defining suitable metrics that help in answering these questions. Table I gives a summary of the goals, questions, and metrics that constitute ME^{3}SA.

The overall goal for ME^{3}SA is to provide a practical model for fact-based evaluation and discovery of improvement opportunities for the energy efficiency of software applications. The viewpoint taken for the model is that of application owners or third-party investigators, both with an interest to know recommended areas of focus. In that sense, the aim of

TABLE I
SUMMARY OF THE GOAL-QUESTION-METRIC DESIGN OF ME^{3}SA. THE QUESTIONS AND METRICS ARE STRUCTURED BY THE SUBGOALS OF THE MODEL: ENERGY BEHAVIOR, CAPACITY, AND RESOURCE UTILIZATION. EACH QUESTION IS LINKED TO ONE METRIC AS INDICATED BY THE NUMBERED ACRONYMS STARTING WITH **Q**, E.G., **Q1** IS ANSWERED BY THE ACC METRIC, **Q2** BY RIC, ETC.

Goal	**Purpose**	Provide a practical model for evaluation and discovery of improvement opportunities
	Issue	for the energy efficiency
	Object	of software applications
	Viewpoint	from the viewpoint of an application owner or third-party investigator.
Energy Behavior		The degree to which the energy consumed by an application meets requirements.
Questions	**Q1**	**What is the energy consumption of each application component?** Answering this question will reveal the biggest energy consumers from a software perspective. This enables evaluators to focus on those software parts where optimizations will potentially have the greatest impact.
	Q2	**How much energy is wasted by an application in the idle state?** Since idle consumption is energy which is by definition wasted [20], an answer to this question will provide additional insight as to where to target optimizations.
	Q3	**How much energy is consumed per unit of work?** By answering this question, evaluators are able to form a perception about the energy consumption of application components in accordance to the work each one begets. This permits them to judge if this cost is reasonable and focus on components that consume disproportionally more energy than expected.
Metrics	**Q1:** ACC	**Annual Component Consumption:** The annual energy consumption per application component measured in kWh.
	Q2: RIC	**Relative Idle Consumption:** The percentage of annual idle energy consumption to total per application component.
	Q3: $CCUW$	**Component Consumption per Unit of Work:** The average energy consumption (in kWh) of each software component per unit of work delivered.
Capacity		The degree to which the maximum energy consumption limits of an application meet requirements.
Questions	**Q4**	**How much power does the application require during peak workload?** The answer to this question will show the difference between power requirements and the actual provisioning capability of hardware during peak workload. This enables evaluators to identify spots in the system where power is over- or under-provisioned.
	Q5	**How much of the theoretical maximum energy budget does the application use?** The answer to this question will enable evaluators to identify the extent to which the average workload of the system utilizes the available energy budget.
Metrics	**Q4:** PGR	**Peak Growth:** A percentage of the actual power demand of application components to the maximum power that hardware can provide during peak workload.
	Q5: PRO	**Provisioning:** A percentage of the energy usage of application components to the maximum energy budget of the hardware.
Resource Utilization		The degree to which the utilization of resources used by an application meets requirements.
Questions	**Q6**	**How does energy consumption scale with an increasing workload?** The answer to this question can show one of the most notable causes for energy waste: applications do not reduce energy consumption when workload and utilization are low.
	Q7	**How power efficient are the host resources with respect to the average workload of the application?** By examining the difference between power usage by host resources and the ideal power usage during average load, evaluators can determine the impact on the energy budget that average workload and hardware choices have.
	Q8	**How much of the total energy consumption of the infrastructure is attributed to the application?** The answer to this question will reveal how much energy is 'lost' in excess by the host resources in order to sustain the operations of the application.
Metrics	**Q6:** CNS	**Consumption Near Sweet-spot:** A percentage that shows the efficiency of the application with respect to its optimal efficiency when delivering work units [4].
	Q7: PG	**Proportionality Gap:** Difference between ideal power provisioning and actual provisioning during average application utilization [21].
	Q8: OPO	**Operational Overhead:** A percentage that denotes the energy overhead required by the infrastructure to process application component workloads.

ME^{3}SA is to be a supplement to software product quality characteristics such as the ones defined by ISO/IEC 25010 [6].

In fact, one of the eight main software quality characteristic defined in this standard, namely performance efficiency, is used as a starting point for the definition of our model. Energy efficiency and performance efficiency are similar on the conceptual level, where each concerns the efficient use of a resource: energy and time, respectively. Adapting from the performance efficiency definition we characterize energy efficiency globally as: *Energy consumed relative to the amount of resources used under stated conditions*. By substituting energy for time in the break-down of performance efficiency provided by ISO/IEC 25010, we obtain the following three high-level issues for evaluation:

- *Energy Behavior*. The degree to which the energy consumed by an application meets requirements.
- *Capacity*. The degree to which the maximum energy consumption limits of an application meet requirements.
- *Resource Utilization*. The degree to which the utilization of resources used by an application meets requirements.

These three evaluation issues constitute the subgoals of our measurement model, for which separate questions and metrics will be defined in the next section.

B. Questions and Metrics

In Table I we raise evaluation questions for each of the three subgoals Energy Behavior, Capacity and Resource Utilization. For space reasons we provide a combined presentation of questions and metrics. In the definitions of metrics, we will consider an application to be composed of a set C of compo-

nents c. In the case where co-allocated components exist, there are some necessary corrective actions that should be taken per metric which are presented in [22].

Energy Behavior

Q1. What is the energy consumption of each application component?: Answering this question will reveal the biggest energy consumers from a software perspective. This enables evaluators to focus on those software parts where optimizations will potentially have the greatest impact.

The energy consumption of a component sampled over a period of time can be calculated as the weighted sum over samples of hardware utilization attributed to a component, weighted by the power drawn at each sample. Formally, we define the ACC (Annual Component Consumption) as follows:

$$ACC(c) := \sum_{s \in S} U_c(s) \times P(U_c(s)), \tag{1}$$

where S is the set of samples in the measurement period, $U_c(s)$ is the CPU utilization attributed to c at sample s, and $P(U)$ is the power consumption in watts at $U\%$ of utilization. Obviously, a sufficiently small unit of time (e.g., seconds) is needed to obtain accurate results, which may not feasible on an annual timescale. To estimate annual consumption we multiply the results of a weekly measurement period by 52.

The hardware utilization attribution function $U(s)$ can be implemented using OS-reported CPU utilization numbers. This approach is also being taken by Kansal and Zhao [17], and Noureddine et al. [23], among others.

Q2. How much energy is wasted by an application in the idle state?: In the context of this model, we consider that application components have two major states: *running* or *idle*. When a component is actively using hardware resources (i.e., utilization is above 0%) and it is producing *units of work* (for some definition) then the component is said to be running. Otherwise, it is idle, and energy used by hardware is essentially wasted from the perspective of the component-under-study [20]. Of course the cause of energy consumption in one application's idle state could be the running of other application (components), OS maintenance processes, or otherwise.

The Relative Idle Consumption (RIC) metric indicates the level of energy inefficiency due to idle consumption. It is defined as follows:

$$RIC(c) := \frac{AIC(c)}{ACC(c)}, \tag{2}$$

$ACC(c)$ refers to the annual consumption for component c (as defined above in Equation 1) and $AIC(c)$ refers to the Annual Idle Consumption for component c. Values for RIC closer to 0% indicate less 'waste' due to idle consumption. $AIC(c)$ can be obtained by sampling in a similar way as for ACC, but summing the power usage levels only for those samples where the utilization is 0%. More formally:

$$AIC(c) := \sum \{P(U_c(s)) | s \in S, U_c(s) = 0\%\} \tag{3}$$

Similar to ACC, we estimate annual idle consumption by multiplying weekly measurement results by 52.

Q3. How much energy is consumed per unit of work?: By answering this question, evaluators are able to judge if energy cost is reasonable and focus on components that consume disproportionally more energy than justified by the amount of work delivered. The difficulty of this question is quantifying the notion of units of work. A universal definition is not feasible. Our model leaves the task of defining an appropriate unit of work per application component to the model users (consisting of evaluators and stakeholders of the application).

We define the metric Component Consumption per Unit of Work (CCUW) as follows:

$$CCUW(c) := \frac{ACC(c)}{AUW(c)}, \tag{4}$$

where the $AUW(c)$ function maps a component to the annual number of units of work delivered.

$CCUW$ is a consumption-based metric [24] that is expressed in kWh / unit of work. Consequently, $CCUW$ can be used to gain insight in the relation between energy consumption and work production. Furthermore, expressing energy costs as business-relevant workloads enables an organization to plan and make more accurate administrative decisions.

Capacity

Q4. How much power does the application require during peak workload?: The answer to this question shows the difference between power requirements and the actual provisioning capability of hardware during peak workload. This enables evaluators to identify spots in the system where power is over- or under-provisioned.

To quantify this aspect, we define the Peak Growth (PGR) metric as follows:

$$PGR(c) := \frac{|P(U_c^{\Delta}) - P_{100\%}|}{P_{100\%}} \times 100\%, \tag{5}$$

where U_c^{Δ} refers to the maximum (peak) utilization that component c reaches during the measurement period, $P_{100\%}$ denotes the theoretical maximum power that the hosting hardware can provide.

The PGR metric is comparable between applications of different types and functional complexity and scales from 0% to 100%. For the purpose of the model we consider very low or high values undesirable from an energy efficiency perspective. PGR can also be used for estimating the sustainable growth of workload of an application given its currently provisioned resources.

Q5. How much of the theoretical maximum energy budget does the application use?: The answer to this question will enable evaluators to identify the extent to which the average workload of the system utilizes the available energy budget.

The use of this metric is justified by the normal practice of provisioning hardware to match (or exceed) the peak workload generated by software applications. At the same time, the power consumption in the average case is responsible for

determining the larger portion of overall energy consumption [25].

We define the Provisioning (PRO) metric as follows:

$$PRO(c) := \frac{ACC(c) - AIC(c)}{|S| \times P_{100\%}}, \quad (6)$$

where S refers to the number of samples (seconds) done during the measurement period. The nominator term reflects that energy consumption when the application is considering performing useful work. The denominator is the maximum theoretical power provided during the measurement period.

PRO values range from 0% to 100% and are comparable between applications, since workload and system types are factored out. On one hand, low percentages point to hardware that is rarely used by application components for production and is probably under-utilized or idle for a significant amount of time. Therefore, consolidating those hardware resources and virtualizing the application components might be a beneficial optimization. On the other hand, very high percentages point to software that is close to violating its energy consumption limits (as defined by current hardware provisioning).

Resource Utilization

Q6. How does energy consumption scale with an increasing workload?: The answer to this question can show one of the most notable causes for energy waste: applications do not reduce energy consumption when workload and utilization are low. The term energy proportionality was first coined by Barosso and Holtze [26] and has been used to describe the ability of software systems to scale energy consumption based on the variability of the workload.

Ideally, the energy efficiency of applications is proportional to the level of workload. This implies that applications consume zero energy when idle, and linearly scale up energy consumption with the workload. Unfortunately, in realistic systems this is hardly ever the case [26].

In our model, the Consumption Near Sweet-Spot (CNS) metric aims at quantifying inefficiencies caused by disproportional energy behavior. It is defined as follows:

$$CNS(c) := \frac{CCUW^*(c)}{CCUW(c)}, \quad (7)$$

where $CCUW^*(c)$ is obtained by identifying the minimum observed energy consumption per work unit throughout the measurement period (typically at a peak workload moment).

CNS ranges from 0% to 100%, where values close to 0% indicate that on average, a component consumes significantly more energy compared to its optimum. Conversely, 100% indicates a fully energy proportional component. Furthermore, since $CCUW$ factors the definition of units of work, this metric is system type independent and can be used for making direct comparisons between applications.

Q7. How power efficient are the host resources with respect to the average workload of the application?: By examining the difference between power usage by host resources and the ideal power usage during average load, evaluators can determine the impact on the energy budget that current hardware choices have.

To quantify this, Wong et al. [21] propose the Proportionality Gap (PG) metric. This metrics expresses the difference between the ideal power consumption and the actual one, at distinct hardware utilization levels. We slightly differentiate from the actual metric and define PG as follows:

$$PG(c) := \frac{P(\overline{U}_c) - P^*(\overline{U}_c)}{P_{100\%}}, \quad (8)$$

where $\overline{U}_c$ denotes the average utilization of application component c during the measurement period. P is the power that this component actually draws on average utilization while P^* is the power that hardware should ideally provide at this point if it was fully proportional (i.e., no consumption in the idle state). Finally $P_{100\%}$ represents the power that host hardware can provision at 100% utilization.

Since PG denotes the extent at which the component deviates from the optimal power consumption case, ideally we would want it to be 0% or as close to zero as possible. This can happen only in cases where the system is fully proportional and thus draws the ideal amount of power regardless of the workload.

As is clear from Equation 8, PG factors out workload definitions and is therefore useful for comparison between application (components) with different functionality.

Q8: How much of the total energy consumption of the infrastructure is attributed to the application?: The answer to this question will reveal how much energy is wasted in excess by the host resources in order to sustain the operations of the application.

The Operational Overhead (OPO) quantifies this aspect as follows:

$$OPO(c) := \frac{ACC(c)}{ASC}, \quad (9)$$

where ASC is the total annual energy consumption by the host resources on which an application component c is deployed. OPO ranges from 0% to 100%, and the higher this value is the less energy is consumed by hardware for operations that are external to the application (component). An OPO value of 90% for instance, would denote that for every watt that is consumed by hardware, 0.9W is used for sustaining an application component, while 0.1W is lost in excess. Nevertheless, OPO can never be 100% as a portion of energy will inevitably be used by infrastructure such as operation systems, middleware, etc.

Similar to Power Usage Effectiveness (PUE), OPO factors out workload as well as application type and therefore it can be used for direct comparisons between applications.

IV. Case Study

In the previous section we defined ME^{3}SA, a practical model for the evaluation of energy efficiency of software applications. In this section we report on a case study, which

was performed in July 2013, that applied ME^{3}SA to an application of the Software Improvement Group (SIG), a software consultancy based in Amsterdam, the Netherlands. The case study was supported by the Software Energy Footprint Lab (SEFlab)[1], which is a research facility to study the impact of software on energy consumption, founded jointly by SIG and Amsterdam University of Applied Science.

A. Goal and Case Study Expectations

The goal of ME^{3}SA is to be able to identify energy consumption hotspots and energy efficiency bottlenecks on a software component level, with reasonable accuracy and within a reasonable timeframe. In this case study, the goal was to apply ME^{3}SA in a realistic context for a measurement period limited to one week (7 days, from July 10th to July 16th). This would allow us to evaluate the feasibility to applying ME^{3}SA in a reasonable timeframe. Furthermore, ME^{3}SA should not be specific to a particular application domain or infrastructure. As the case study considered only one application, it does not allow us to evaluate the general applicability of the model.

B. The Application-Under-Study

One of the main applications of SIG is the Software Monitor. This application takes as input frequent source code snapshots of other applications, analyzes them by applying source code-level metrics, and presents the results on a website. SIG uses this application in consultancy services such as software risk assessments and software risk monitoring.

Software Architecture: Figure 1 provides an overview of the Software Monitor application and its components. On a high-level there are two sub-systems with the Software Monitor. First, the SAT, or Software Analysis Toolkit, provides the core metric functionality by means of two components: Monitor Admin and SAT (MA+SAT) and Acceptance Database (AD). MA+SAT consists of an administration interface, source code metrics calculation, and code logistics functionality. AD is a database that stores the output of the MA component. The second sub-system, the Monitor, consist of two front-end components, respectively for use by clients and internal use: the Production Monitor (PM) and Acceptance Monitor (AM). Finally, the Production Database (PD) component serves (validated) data to the PM front-end.

Hardware Deployment: Each of the five components is deployed on a dedicated, virtualized, Linux server.

Units of Work: The Software Monitor performs two types of units of work, corresponding to the subsystems described earlier.

1) Application analysis. For the components MA+SAT and AD, one unit of work consists of the analysis of one snapshot of source code and the subsequent storage of the results.
2) User session. For the components PM, PD and AM, one unit of work consists of a user session on the PM front-end.

[1]http://www.seflab.com

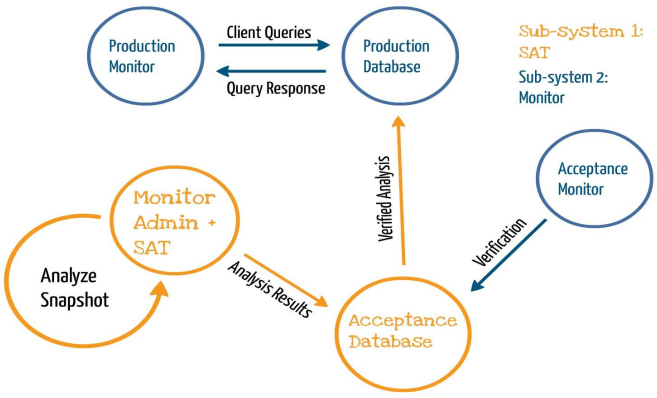

Fig. 1. Software architecture of the Software Monitor.

Power Modelling: In this case study, it was not feasible to measure the power consumption (the $P(U)$ function of our model) on the hardware resource themselves. The hardware was located off-site in a data centre, which was not accessible for power measurement within the timespan of the study. The study therefore used the following (linear) power model for each hardware resource:

$$P(U) := (P_{100\%} - P_{0\%}) \times U + P_{0\%}, \tag{10}$$

where U is a utilization percentage. In accordance to SEFLab measurements, $P_{100\%}$ was set to 220W and $P_{0\%}$ was set to 160W, reflecting the maximum and idle power consumption of a hardware resource. These settings are conservative choices based on measurements done on servers at the SEFLab [27]. Compared to power numbers reported by Meisner et al. [28], our power model is on the conservative side, using less power in both maximum or idle states.

C. Data Collection and Analysis

Figure 2 provides an overview of the data collection process that we followed during the case study. During the measurement period of 7 days, scripts installed on each hardware resource collected data given two sources:

Log files: The components of the Software Monitor log the start and completion of units of work. The script monitors the log files and extracts the number of units of work and their start and completion times.

Process data: Every second, the script polls the CPU utilization data of each process known to belong to a component of the Software Monitor, using the OS-provided tools (on Linux, the `/proc/<PID>/stat` file).

The consolidation script combines the data on utilizations and units of work to obtain utilization information per Unit of Work. This step is needed for the $CCUW^*$ component (the ideal consumption per Unit of Work) of the CNS metric. Finally, the metric calculation script calculates all metric of the model using the utilization data, the units of work data, and the power model provided by Equation 10.

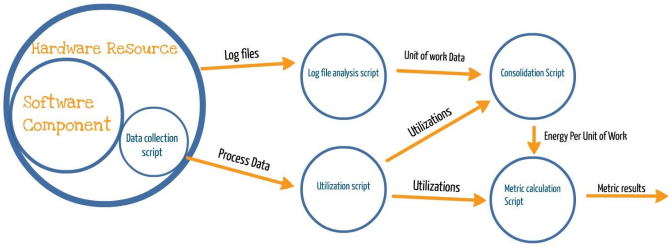

Fig. 2. Overview of the Data Collection and Analysis Process.

D. Results and Observations

Table II presents results obtained for the measurement period July 10th – July 16th ($|S|$ = 604,800 seconds). Here we will briefly discuss the results per metric, while full details can be found in [22]. As the measurement period of the case study spans only 1 week, the results where scaled to an annual timespan by multiplying by 52.

TABLE II
RESULTS FOR THE PERIOD JULY 10TH – JULY 16TH, 2013, FOR THE COMPONENTS MONITOR ADMIN AND SAT (MA+SAT), ACCEPTANCE DATABASE (AD), ACCEPTANCE MONITOR (AM), PRODUCTION DATABASE (PD), AND PRODUCTION MONITOR (PM).

Metric	MA+SAT	AD	AM	PD	PM
$ACC(kWh)$	1,422	1,399	1,398	1,399	1,400
RIC	0%	65%	48%	86%	65%
$CCUW(kWh/unit)$	0.105	0.103	0.099	0.099	0.099
PGR	1%	21%	3%	23%	7%
PRO	76%	25%	38%	10%	25%
CNS	30%	21%	13%	13%	13%
PG	64%	72%	73%	73%	72%
OPO	94%	99%	99%	99%	99%

ACC: At first glance, the results for ACC point out that the MA+SAT component the biggest energy consumer of the application, but by only a small margin of 22 kWh different to the runner up (the PM component). The relatively uniform consumption among the components can be partially attributed to our use of the linear power model, in particular its relatively small dynamic range (220W peak power, 160W idle power, dynamic range 60W). Nevertheless, since MA+SAT arguably exhibits the highest functional complexity, being the analysis of source code snapshots, its consumption being the highest was expected.

RIC: Except for the MA+SAT component (which has a RIC of 0% and thus hardly ever idles), all components exhibit significant RIC values, pointing to energy waste due to components idling. In particular, the energy consumption of the PD component consists for 86% of energy wasted while idling. In Table III, which shows the units of work delivered on a daily basis, it can be observed that the PD, PM, and AM components deliver relatively small amounts of work on the weekend-days. Since the unit of work defined for those components represents a user session performed by clients of SIG, this result is not surprising.

It could be worthwhile to focus optimizations effort on reducing the idle consumption of the PD, PM and AM components, in particular during the weekends. Feasible optimization paths in that case would be to host the production database to hardware that has physical components with lower-power idle states or to schedule its workload in a way that creates certain timeframes during which the resource can be powered down.

$CCUW$: The $CCUW$ results in Table II show very similar energy consumptions per unit of work: around 0.1 kWh per unit of work, representing a unit cost of a little less than a Euro-cent[2]. The uniformity is surprising given different definitions of the units of work, and different software components (however, the similar values obtained for ACC and AUW obviously indicate the uniformity observed for $CCUW$).

Table III shows the units of work delivered during the measurement period, and the scaled annual number used for the AUW metric. Again, we observe that the components of the Monitor subsystem (AM, PD, PM) perform significantly less work (consisting of user sessions) on weekend days. Obviously, this has a consequence for the $CCUW$ if calculated for a daily timespan. Figure 3 shows the daily $CCUW$ for the PM component. On Sunday, July 14th, the $CCUW$ reaches its highest value; 3,500 kJ or around 1 kWh per user session. The unit cost for the PM component on a Sunday is around 8 Euro-cents, or around 8 times higher than average.

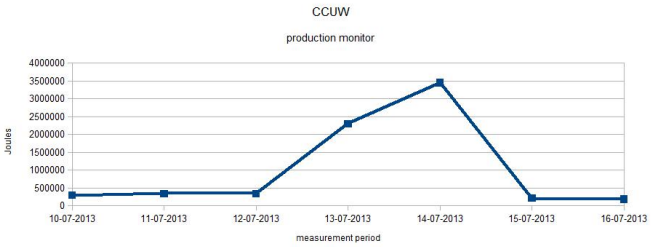

Fig. 3. $CCUW$ for the Production Monitor during the measurement period.

It is interesting to consider the results for $CCUW$ together with RIC. For the MA+SAT component, close to all of its energy cost per unit is coming from actual utilization, i.e., computation, while for the Monitor components, the unit costs are dominated by idle consumption.

PGR: The first observation that can be made regarding PGR metric is that the MA+SAT component reaches the maximum power provisioning capacity of its hardware, by sometimes operating at near 100% utilization. Optimization efforts therefore could examine the performance impact of these peak utilization moments, in order to determine if additional latency and thus energy consumption are imposed.

A second observation that can be made is that both databases (AD and PD) operate far from the maximum power provisioning capability of their hardware. This denotes that they are under-utilized and thus less energy efficient even during peak load. Consequently, a feasible optimization in order to improve their energy footprint would be to virtualize

[2]According to Eurostat energy price statistics for industrial consumers in the Netherlands in 2013 (Eur 0.0789 per kWh).

TABLE III
UNITS OF WORK DELIVERED BY COMPONENTS.

Component	Wednesday, July 10th	Thursday, July 11th	Friday, July 12th	Saturday, July 13th	Sunday, July 14th	Monday, July 15th	Tuesday, July 16th	Total	Scaled annually (AUW)
MA+SAT	19	15	38	52	49	58	29	260	13,520
AD	19	15	38	52	49	58	29	260	13,520
AM	46	40	40	6	4	63	72	271	14,092
PD	46	40	40	6	4	63	72	271	14,092
PM	46	40	40	6	4	63	72	271	14,092

TABLE IV
DAILY AND TOTAL AVERAGE UTILIZATION % PER COMPONENT ($\overline{U_c}$).

Component	Wednesday, July 10th	Thursday, July 11th	Friday, July 12th	Saturday, July 13th	Sunday, July 14th	Monday, July 15th	Tuesday, July 16th	Overall
MA+SAT	9.37%	11.54%	14.60%	11.84%	11.66%	12.57%	11.33%	11.84%
AD	0.23%	0.32%	0.47%	0.33%	0.34%	0.54%	0.55%	0.40%
AM	0.13%	0.15%	0.15%	0.12%	0.11%	0.17%	0.16%	0.14%
PD	0.10%	0.14%	0.12%	0.09%	0.23%	0.40%	0.24%	0.18%
PM	0.39%	0.38%	0.41%	0.39%	0.40%	0.42%	0.46%	0.41%

both components and host them on the same hardware, if their peak load occurs in different time-frames.

A final observation is that even though the AM component presents a low PGR, by inspecting its daily results (not shown here for space reasons, see Figure 18 in [22]) we find that its peak load only occupied a very small time frame on a daily basis leaving the component under-utilized for the rest of the day. This type of information can be used to guide virtualization efforts.

PRO: The results in Table II clearly show the MA+SAT component uses its hardware extensively, whereas the other components do not. Since PRO is calculated based on active consumption, the low values of the other components are explained by their higher idle consumption (also pointed out by RIC) Consequently, optimizations with respect to PRO should first focus on PD. Feasible optimizations would be to co-host this component with AD since, at least for the measurement period, they do not present a significant energy demand.

CNS: A CNS value of 30% for MA+SAT indicates that the most efficient code analysis was performed at 30% of the energy consumption of the average code analysis. The monitor and database components have even lower values for CNS. There can be two causes for these results:

1) Low utilizations,
2) Variability in the running-time of the units of work.

Table IV shows that components have low average utilizations. Furthermore, it is probable that the units of work, code analyses and user session, are not uniform. Code analyses could be analysing different software systems from different clients, and durations of user sessions are highly dependent on user behavior.

A possible optimization to increase CNS of those components would be an approach proposed by [29] and [30], consisting of virtualization, software driven workload allocation, virtual machine migration, and suspensions of hardware during low workload periods.

PG: PG represents the gap between actual and optimal proportionality of energy consumption given the average utilization. Referring to the actual power model in Equation 10, the ideal power model is obtained by setting $P_{0\%}$ to 0W. The values for PG in Table II show that for all components a significant gap exist, that could in theory be reduced by using hardware with better scaling capabilities.

Table IV shows the average utilizations per day for the components. It is obvious that all operated at highly inefficient utilization regions. This can be attributed to both the significant idle time frames that exist and to the low intensity of the workload.

OPO: The OPO results shown in Table II are unsurprising for dedicated servers. The energy consumption of infrastructure software, such as the OS, is marginal.

V. DISCUSSION

The goal of ME^{3}SA was stated as follows: "identifying energy consumption hotspots and energy efficiency bottlenecks on a software component level, with reasonable accuracy and within a reasonable timeframe. The model should not be specific to a particular application domain or infrastructure." In this section we discuss some high-level outcomes of the case study, we evaluate to what extent ME^{3}SA has met our goals, and we discuss directions for future work.

A. Outcomes

The high-level outcomes of the case study can be summarized as follows:

- The overall utilization of the hardware resources is rather low for all of the studied software components (see Table IV).
- For all except one component (MA+SAT), the low utilization appears to be caused mainly by a large amount of idle time (see results for RIC in Table II).
- The energy cost per unit of work (code analysis or user session) is in the approximate range of 0.1 to 1.0 kWh per unit.

An interesting question is: How much energy could potentially be saved by implementing optimizations? Referring back to Table II, the ACC metric shows how much energy a component uses in total. Of that total, RIC measures the percentage of energy consumption while idle (i.e., not producing units of work). Clearly, the components AD, AM, PD and PM are consuming a major part of their energy while idle (with RIC values of 65%, 48%, 86%, and 65%, respectively). Theoretically, given that energy consumption could be reduced to zero if a component is idle, a sum of approximately 3,700 kWh could be saved from those components on a yearly basis.

The PG metric also allows us to estimate potential savings. Given proportionally scaling hardware (i.e., hardware having a power model starting from zero and scaling linearly with utilization), the value of PG represents the percentage of ACC that could be saved. For all components together, this amounts to a savings potential of 5,000 kWh on a yearly basis. This number obviously subsumes the previous result of 3,700 kWh since proportionally scaling hardware will by definition consume no energy while idling.

Zooming in further, as we did for the $CCUW$ metric (see Figure 3), we can get more insight as to implementing optimizations. For the components AD, PD and PM, it is clear the weekends constitute mostly idle time. Implementing a policy of scaling down hardware during weekends could therefore already result in substantial savings.

B. Validity Threats

In this section we evaluate our study with respect to potential validity threats (as defined in [31]).

1) Construct validity: Construct validity refers to the question whether the employed measures capture the phenomenon of interest. In our study, the measurement model was directly derived from research questions as outlined in Table I. Furthermore, the used measures were (partly) based on related literature.

2) Internal validity: Internal validity regards the (causal) relations between factors of the study. Our study does not consider causality, however there still may be factors that influence the results. The accuracy of the results is determined by the following three inputs to the model:

- The power model (P function),
- The utilizations per component (U_c function),
- The logging of start and completion of units of work.

In the case study, the utilizations and units of work were automatically measured on a scale of seconds, which is accurate enough for the purpose of the model. However, the power model used (see Equation 10) is an over-simplification of the behavior of real hardware resources. Evidently, the results have been influenced by the power model, and should therefore be considered indicate at best. In the ideal case, the power function is validated by measurement of the actual power consumption by the relevant hardware.

3) External validity: External validity regards the question to what extent the results could be generalized and applied to other cases. There are two main threats to external validity in our study:

Duration. The case study lasted for only 7 days. On the one hand, this short timespan argues in favor of the model, since we still identified some bottlenecks in the energy consumption profile of the Software Monitor application. On the other hand, a short duration raises the issues of representativeness. What is the chance that in another week, the energy consumption profile would be completely different? There may be factors such as project scheduling, infrastructure changes, seasonal effects, or otherwise, influencing the results. In the context of this study, this issue cannot be evaluated further, as the necessary data is not available. As a result, every application of the model will need to re-evaluate the representativeness of the measurement period.

One subject study. Our study (by design) considered only one application. It is therefore not possible to empirically validate the generizability of the model to other applications or application domains. Referring back to the model design (Section III) however, we can observe that as long as the three inputs (P and U_c functions, unit of work logging) can be provided, the metrics of the model could be applied. On a more technical level, there are obvious dependencies on the infrastructure and application domain. First, obtaining the P and U_c functions will require infrastructure-specific measurement tooling or instrumentation. Second, the definition of the 'unit of work' remains application domain-specific, hence the model itself cannot provide this definition.

4) Reliability: Finally, reliability considers the extent to which the results depend on the researchers that executed the study. There is mostly a dependency on the operational behavior of the underlying application during a measurement period, as measurement itself is a (semi-)automated process. The data collection and processing steps that were followed by the researchers are described in detail in [22].

C. Future work

We foresee several directions for future work. First, as this paper provides only the first application of the ME^{3}SA model, an essential next step is to apply the model to different applications, in different domains, for diverse infrastructures, in order to further validate its usefulness.

Supporting future applications, we envision a suite of automated tools that assist in generating the model's inputs. For utilization data, platform-specific scripting could be developed to collect utilization information on several standard platforms such as Unix/Linux and Windows Server. For the identification of units of work, generic tooling is probably out of reach given the diversity of work done by software. However, some formalized guidelines for identification of work units could be provided. Finally, for power models, a very useful tool would be a database of power models for common hardware configurations. Given such a database, hardware power consumption would not need to be measured for every application of the model.

Given a growing number of documented applications of ME^{3}SA, a further step would be to develop a benchmark for the energy efficiency of software applications. A similar benchmark has been developed for the maintenance costs of software applications [32]. Such a benchmark allows applications to be rated and compared to each other quantitatively, improving decision-making. In [22], an initial version of such a benchmark and rating scheme has been developed.

Last, the current model assesses the energy efficiency of applications that are operational. An interesting extension would be to include the energy consumption of the complete life cycle of a software application. In particular, how much energy is used while designing, developing, testing, deploying, and supporting the application? Such an extension would be necessary to truly compare the energy efficiency of software compared to other solutions.

VI. Conclusion

In this paper we have presented ME^{3}SA, a practical model that defines a set of measurements for evaluating the energy efficiency of software applications (Section III). The model consists of a GQM-based design including an overall goal, evaluation questions, and metrics. In total 8 metrics are defined, quantifying the Energy Behavior, Capacity, and Resource Utilization aspects of a software application.

To provide an initial validation of ME^{3}SA, we presented a case study in which the model was applied to an operational software application (Section IV). For 7 consecutive days, the application's utilizations and produced units of work were measured. The results indicate that this application could in theory perform the same work using up to 5,000 kWh (71%) less energy annually on fully proportional hardware. It should be noted that due to the usage of a simplified (linear) power model, the results of the case study have to be considered only an indication. Finally, the case study showed that the metrics used in the model allow for zooming in on finer-grained time periods to identify opportunities to implement savings.

Acknowledgments

The authors would like to thank Miguel A. Ferreira (at the time at Software Improvement Group, now at Schuberg Philis) for his guidance and critical commenting on the research, and the SEFLab for their collaboration on measuring the energy consumption of software.

References

[1] J. Koomey, "Growth in data center electricity use 2005 to 2010," 2011.
[2] P. Teunissen and E. Lambregts, "Energiebesparing bij datacenters," 2012.
[3] A. Noureddine, R. Rouvoy, and L. Seinturier, "A review of energy measurement approaches," *ACM SIGOPS Operating Systems Review*, vol. 47, no. 3, pp. 42–49, 2013.
[4] J. Arnoldus, J. Gresnigt, K. Grosskop, and J. Visser, "Energy-efficiency indicators for e-services," in *Green and Sustainable Software (GREENS), 2013 2nd International Workshop on*, pp. 24–29, IEEE, 2013.
[5] K. Grosskop and J. Visser, "Identification of Application-level Energy Optimizations," *Proceedings of the First International Conference on Information and Communication for Sustainability (ICT4S)*, pp. 101–107, 2013.
[6] ISO/IEC, "ISO/IEC 25010 - Systems and software Quality Requirements and Evaluation (SQuaRE) - System and software quality models," tech. rep., 2010.
[7] V. R. Basili, G. Caldiera, and H. D. Rombach, "The goal question metric approach," in *Encyclopedia of Software Engineering*, Wiley, 1994.
[8] K. Grosskop and J. Visser, "Chapter 5- energy efficiency optimization of application software," *Advances in Computers*, vol. 88, pp. 199–241, 2013.
[9] E. Kern, M. Dick, S. Naumann, A. Guldner, and T. Johann, "Green Software and Green Software Engineering–Definitions, Measurements, and Quality Aspects," in *Proceedings of the First International Conference on Information and Communication for Sustainability*, p. 87, 2013.
[10] M. Bertoncini, B. Pernici, I. Salomie, and S. Wesner, "Games: Green active management of energy in IT service centres," in *Information systems evolution*, pp. 238–252, Springer, 2011.
[11] 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.
[12] L. Siso, J. Salom, M. Jarus, A. Oleksiak, and T. Zilio, "Energy and Heat-Aware Metrics for Data Centers: Metrics Analysis in the Framework of CoolEmAll Project," in *2013 International Conference on Cloud and Green Computing (CGC)*, pp. 428–434, IEEE, 2013.
[13] N. B. Morteza Jamalzadeh, "An exhaustive framework for better data centers' energy efficiency and greenness by using metrics," *Indian Journal of Computer Science and Engineering*, vol. 2, no. 6, pp. 813–822, 2012.
[14] B. Steigerwald and A. Agrawal, "Developing green software," tech. rep., Intel Software Solutions Group, 2011.
[15] P. Larsson, "Energy-efficient software guidelines," *White Paper*, 2012.
[16] 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*, pp. 39–50, ACM, 2010.
[17] A. Kansal and F. Zhao, "Fine-grained energy profiling for power-aware application design," *SIGMETRICS Perform. Eval. Rev.*, vol. 36, no. 2, pp. 26–31, 2008.
[18] J. Flinn and M. Satyanarayanan, "Powerscope: A tool for profiling the energy usage of mobile applications," in *Proceedings of the Second IEEE Workshop on Mobile Computer Systems and Applications*, WMCSA '99, pp. 2–, IEEE Computer Society, 1999.
[19] T. Do, S. Rawshdeh, and W. Shi, "pTop: A Process-level Power Profiling Tool," in *HotPower '09: Proceedings of the Workshop on Power Aware Computing and Systems*, ACM, Oct. 2009.
[20] M. R. Stan and K. Skadron, "Guest editors' introduction: Power-aware computing," *Computer*, vol. 36, no. 12, pp. 35–38, 2003.
[21] D. Wong and M. Annavaram, "Knightshift: Scaling the energy proportionality wall through server-level heterogeneity," in *Microarchitecture (MICRO), 2012 45th Annual IEEE/ACM International Symposium on*, pp. 119–130, 2012.
[22] G. Kalaitzoglou, "Evaluating the energy efficiency of a software system – a practical model," Master's thesis, University of Amsterdam, 2013.
[23] A. Noureddine, A. Bourdon, R. Rouvoy, and L. Seinturier, "A preliminary study of the impact of software engineering on GreenIT," in *GREENS*, pp. 21–27, IEEE, 2012.
[24] R. P. Larrick and K. W. Cameron, "Consumption-based metrics: From Autos to IT.," *Computer*, vol. 44, no. 7, pp. 97–99, 2011.
[25] X. Fan, W.-D. Weber, and L. A. Barroso, "Power provisioning for a warehouse-sized computer," *SIGARCH Comput. Archit. News*, vol. 35, pp. 13–23, June 2007.
[26] L. A. Barroso and U. Hölzle, "The case for energy-proportional computing," *Computer*, vol. 40, pp. 33–37, Dec. 2007.
[27] M. A. Ferreira, E. Hoekstra, B. Merkus, B. Visser, and J. Visser, "SEFLab: A lab for measuring software energy footprints," in *Green and Sustainable Software (GREENS), 2013 2nd International Workshop on*, pp. 30–37, IEEE, 2013.
[28] D. Meisner, B. T. Gold, and T. F. Wenisch, "Powernap: eliminating server idle power," in *Proceedings of the 14th international conference on Architectural support for programming languages and operating systems*, ASPLOS XIV, pp. 205–216, ACM, 2009.
[29] R. Urgaonkar, U. C. Kozat, K. Igarashi, and M. J. Neely, "Dynamic resource allocation and power management in virtualized data centers.," in *NOMS*, pp. 479–486, IEEE, 2010.
[30] I. Goiri, F. Julia, R. Nou, J. L. Berral, J. Guitart, and J. Torres, "Energy-aware scheduling in virtualized datacenters," in *Proceedings of the 2010 IEEE International Conference on Cluster Computing*, CLUSTER '10, pp. 58–67, IEEE Computer Society, 2010.
[31] P. Runeson and M. Höst, "Guidelines for conducting and reporting case study research in software engineering," *Empirical Software Engineering*, vol. 14, pp. 131–164, Dec. 2008.
[32] R. Baggen, J. P. Correia, K. Schill, and J. Visser, "Standardized code quality benchmarking for improving software maintainability," *Software Quality Journal*, vol. 20, no. 2, pp. 287–307, 2012.