University of Amsterdam

Master Software Engineering

Master Thesis

EVALUATING THE ENERGY EFFICIENCY OF A SOFTWARE SYSTEM

A PRACTICAL MODEL

Georgios Kalaitzoglou

Universiteit van Amsterdam

Thesis Supervisor:
Magiel Bruntink

Internship Supervisor:
Miguel Ferreira

August 13, 2013

[August 13, 2013 at 14:41 – classicthesis version 4.1]

Georgios Kalaitzoglou : *Evaluating The Energy Efficiency of a Software System,* A Practical Model, © August 2013

ABSTRACT

The information era has been marked by a remarkable increase in resource demand that has been facilitated by the Information and Communications Technology (ICT) in an energy-agnostic way. As a result, Energy and Power have started to evolve into major design and operational considerations.

Despite, the rising concern regarding the ramifications that ICT's rising energy consumption creates, there is still no practical and widely applicable method that is able to determine the energy efficiency levels of a software system. In an effort to fill this gap and give an answer to the research question: "How to develop a practical model that will have the ability to evaluate the energy efficiency of software systems?" this thesis proposes a model to evaluate the different aspects of energy efficiency that are relevant to the majority of modern software systems.

We first propose a conceptual division of energy efficiency into sub-characteristics that are taken under account by this research. We then provide rationale and raise relevant questions in order to identify metrics that can be used to quantify those sub-characteristics. Argumentation regarding the metric choice as well as advice on how to act upon their values is provided. Subsequently, an evaluation is performed in order to determine if the model holds any merit, by applying it to the systems of the Software Improvement Group. The results demonstrate that this modeling approach is promising, although more evaluations are needed to come to a tentative answers regarding its value.

The model is intended to be used by stakeholders and SIG consultants in order to evaluate the energy efficiency levels that a software system exhibits. It gives an understanding of how the energy consumption of the system is attributed to its main components, based on the functionality that they deliver, and provides an indication of system spots where energy waste occurs. This enables potential evaluators to discover optimization opportunities and prioritize them more effectively.

I wish there was a way to know
you're in the good old days
before you've actually left them

— Andy Bernard

PREFACE

This thesis is the result of the research carried out during an internship in the Software Improvement Group in order to complete the MSc in Software Engineering at the University of Amsterdam. It would not have been possible to finish this thesis without the help and support of several people.

First, I would like to thank all my professors at the UvA. They have gracefully shared their knowledge, kept me inspired and motivated and guided me through the path to become a software engineer.

I would like to express my gratitude to my supervisor Miguel Ferreira for his valuable guidance, engagement and support during the course of this Master thesis. His insightful comments and constructive criticisms were thought-provoking and they helped me focus my ideas.

I would also like to sincerely thank, Magiel Bruntink for helping me through my first experience as a researcher. He has been a source of motivation and encouragement in times that i got weary.

I would also like to extend my appreciation to Joost Visser for giving me the opportunity to work at the Software Improvement Group and to all my colleagues, who provided valuable help and support in order to conduct the case study presented in this thesis.

Last but not least, my eternal gratitude goes to my parents, Konstantinos and Aikaterini, who have been a constant source of support - emotional, moral and of course financial - and this thesis would certainly not have existed without them. Thank you for your unconditional trust, timely encouragement, and endless patience.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTION

During the past decade the increasing demand for Information and Communication Technology (ICT) services has been remarkable [61]. The massive surge in the use of mobile devices and smartphones (which for the first time surpassed computers in sales) , web-services, and the widespread adaptation of social media and cloud computing are only some aspects of this growth. This rapid transition to the information era, has signaled a vast increase in resource demand and energy consumption that is not expected to stop in the forth-coming future.

In contrast, the resources that are required to sustain such growth in a limited world, only become more scarce. As such, energy availability has started to tackle the pace of evolution, as energy efficiency levels have followed a less steep increase than performance or hardware cost reduction [51], which, up until some years ago, were the sole concerns of organizations. This rapid and unruly expansion, driven by the fast pace with which demand for ICT services was generated, the competitive business environment and time-to-market considerations, is now taking its toll, as it is projected that the energy costs[1] (the financial impact of energy consumption) of operating a typical server will surpass those of acquiring it [5]. Consequently, the ICT is expanding in a way that will not be sustainable in the near future and thus, fundamental changes regarding its operational practices are necessary.

In face of those challenges, there is a growing realization of the energy problem which was expressed by researchers such as Pernici et al. [70]: *"the point that energy efficiency should be given a very relevant role in Information Systems design."*. This increasing awareness regarding the ramifications that the rising energy consumption of the ICT creates, rendered energy efficiency into a mainstream consideration and concepts such as the GreenIT [65] and green computing[2] have sprung with the sole purpose of designing and operating computer systems that will deliver the highest amount of production with the least possible energy consumption [88][56].

At an early stage energy optimizations and power management were focused on the lower levels of the system stack with hardware related optimizations, such as Dynamic Voltage and Frequency Scaling(DVFS)[3] being in the center of attention, leading in turn to considerable energy gains. However, improvements in energy efficiency where still insignificant compared to the rise of energy consumption caused by the the ever-growing demand which ICT should cover and with poor efficiency in many cases, with Ranganathan [73] suggesting that the energy efficiency of today's systems can be improved by at least an order of magnitude.

This lead practitioners in the field to realize that all systems, regardless of context, share a common characteristic. That is, they excute software to deliver some functionality

1 Koomey et al., in [50], estimated that the cost of operating and cooling servers, in 2005, was 7.2 and 2.7 billion dollars, worldwide and in the US respectively. His estimation for 2010 [46] suggested that the energy expenditure of the ICT lay between 1.1% and 1.5% of the global energy consumption. More information regarding the energy consumption of the ICT can be found in Section 3.3

2 According to San Murugesan [65] green computing is: *"the study and practice of designing, manufacturing, using, and disposing of computers, servers, and associated subsystems efficiently and effectively with minimal or no impact on the environment."*

3 DVFS is used to lower the frequency with which the processor executes instructions, signaling in return a corresponding decrease to the Voltage that is required for the resource to function.

and this software had long been developed in an energy agnostic way. Consequently, optimizing solely on the hardware layer in not enough because it neglects the fact that software has an undeniable role in determining the energy consumption of the hardware that is used to host it. Therefore, energy optimizations in software design started to gain attention and many efforts followed, that tried to create energy-aware applications [2], with Brown [13] suggesting that: *"the most strategic aspect of energy efficient computing will be the evolution of application software to facilitate system-wide energy efficiency"* as is also stated in [45].

Regardless of the fact that energy efficiency optimizations are increasing in number, limited work has been done in quantitatively evaluating the energy efficiency of software systems. As such, there is currently no widely accepted model neither conceptual nor practical that offers a standardized and universally applicable way of evaluating the energy efficiency of heterogeneous software system in a holistic fashion by producing results that are fair, comparable and actionable.

A model with the preceding characteristics can prove a viable tool in dealing with energy efficiency problems and increasingly occurring energy costs as it will have the ability to more accurately coordinate potential optimization efforts increasing their impact in turn. This is something that the Software Improvement Group (SIG), which facilitates this research has already identified. The general goal of SIG is to translate technical findings into clear and actionable advice for the upper management and therefore it has already moved towards providing solutions regarding energy efficiency problems through the Green Software Scans [36], with 3 scans already performed, and by creating SEFLab[4] [28]. These scans, are a first approach to evaluate the energy behavior of a system in order to provide recommendations for reducing energy waste. Although these scans provide an effective way for reducing the energy expenditures of a software system *in situ*, they lack an underlying evaluation model as they are heavily dependent on expert opinion and knowledge.

This research delves into the evaluation domain of energy efficiency in an effort to create and incorporate such model in the current practices of SIG. In this regard, the underlying problems and limitations of current evaluations are explored and the modeling effort is presented. The goal of this thesis is to evaluate the energy efficiency of modern software systems and to determine if the proposed model is indeed able to identify parts of a software system that are not energy efficient in order to discover clear optimization opportunities to eliminate energy waste.

The remainder of this document is structured as follows: In Chapter 2 the problem as well as the motivation and the goal behind this research are stated and the research question is defined. In Chapter 3 relevant information on the topic of energy efficiency is given and related work is presented. Chapter 4 presents the research methodology that was followed while Chapter 5 exhibits the conceptual division of efficiency that this research proposes. In Chapter 6 the practical model and its metrics are introduced as well as the aggregation and interpretation methodology. Chapter 7 presents the case study that was conducted in order to evaluate the practical model. The experimental setup is explained and the results are presented. In Chapter 8 the interpretation of the results and the evaluation of the model are given together with the threats to validity. Finally, Chapter 9 presents the answers to the research question and the future work that this research proposes.

4 The setup of the lab includes infrastructure that enables measurements at hardware and software level and alignment of the resulting measurement streams, so that software behavior can be effectively mapped to resource usage and energy consumption.

2 PROBLEM STATEMENT AND MOTIVATION

2.1 THE PROBLEM

Despite the increasing awareness regarding the growing energy problem of software, evaluating software energy efficiency is still an open issue. Not even new `Green IT` initiatives appearing rapidly in the past few years and the intensification of research focus on optimizations regarding energy efficiency has lead to concrete evaluation methods or models.

The problem addressed in this research is, therefore, the construction of a software application energy efficiency evaluation model.

2.2 RESEARCH GOAL

From the above elaboration we formulate the following goal for this research:

- `Research Goal:` *Create a practical model that will be able to help stakeholders and* `SIG` *consultants evaluate the energy efficiency of a software system.*

  Such a model should be able to help evaluators:
  - Identify parts of the system that are not energy-efficient.
  - Discover optimization opportunities.
  - Prioritize optimization activities.

2.3 MOTIVATION AND PROBLEM EXPLANATION

We believe that evaluating the energy efficiency of a software system might be as crucial as optimizing it, because consumers, manufacturers and researchers, need to be able to assess the energy efficiency levels of systems in order to make the correct decisions in purchasing, deploying and operating the best possible solution for them. Furthermore, since the energy usage has been bloated by the energy-agnostic practices of the past, it is crucial to be able to evaluate new and legacy systems, in order to identify inefficient practices and eliminate energy waste. Consequently, standardized and benchmarked models able to evaluate heterogeneous software systems' energy efficiency in a well-defined and universally applicable way are needed. We describe next, the challenges we envisage for the construction of the desired model.

First challenge: Widely applicable software unit of work

In its practice, `SIG` is asked by its clients to analyze all sorts of software systems, ranging from embedded systems for small portable devices to large scale distributed systems that run in multiple data centers throughout the world.

Since efficiency has to do with producing the same (or more) work with less (or the same) resources[1], one needs to define a unit of software work that can capture the (main)

1 A more thorough elaboration is given in Section 3.1.2

functionality offered by any kind of software. This is, however, not trivial due to the vast diversity found in software. The functionality offered by software from different domains (such as finance, health, science, logistics, just to name a few) varies widely. Furthermore, even for similar software systems, functionality can be delivered in very distinct ways depending on technical aspects such as its underlying architecture (e.g. centralized *vs.* distributed). Technology also plays a key role in the diversity found in software, as it can be built in a multitude of programming paradigms, using design patterns from an extensive catalog and deployed in real or virtualized hardware. Some systems are accessed via the web, others are installed locally on the machines that run them. In turn these machines can be anything from a mainframe, to a laptop or a tablet.

Finding the right abstraction for a software unit of work that can cope with such diversity is, therefore, a challenge.

Second challenge: Overcome decades of separating concerns

The principle of separation of concerns enabled the ICT industry to develop more and more complex systems in a manageable fashion. When computers were the size of an entire room, programmers had to operate them via software from booting, to execution of the desired tasks and finally shutdown. With the advent of modern operating systems (OSs), programmers were able to abstract away from the low-level intricacies of the machine since the hardware resources were now offered to them (by the OS) as services. The story goes on as the principle of separation of concerns shaped the multiple layers of abstraction that support software systems as we know them. What is important to learn from this is that these abstraction layers were designed to be opaque. In other words, they were designed to hide the details of what is under them.

Software in itself does not consume any energy. What it does is to use the available computational resources (calculation, storage, communication, etc.) to perform some work. Underlying to the computational resources are, of course, hardware components (CPU, memory, hard drives, network cards, routers, etc.) which, in turn, do consume energy. Consequently, when we talk about software energy efficiency, what we are actually talking about is the efficient use of computational resources by software, in a way that reduces the energy consumption of the hardware components that are hidden way under several layers of abstraction. This means that in order to relate the software activity to the energy consumption of the hardware we must be able to "see" through several layers specifically designed to be opaque.

One additional difficulty arises from the vast amount of possible combinations of hardware and software layers. On the one hand, the same hardware running different software layers (OS, device drivers, middleware, etc.) will most likely exhibit different energy consumption characteristics. While, on the other hand, the same software running on different hardware will also most likely induce differences in its energy consumption [26]. Therefore, despite the fact that this research is focused on evaluation of software systems, it cannot neglect the underlying hardware.

Overcoming decades of separating concerns in abstraction layers is, therefore, a challenge.

Third challenge: Lack of conceptual starting point

Unlike many other quality aspects of software, energy efficiency is not covered by international standards for software quality (e.g. ISO/IEC 25010 [44]). This means that there

is no commonly accepted definition framework of software energy efficiency to start building an evaluation model from.

The literature around the topic of software energy efficiency does not provide the necessary guidance for building such a model either. The literature reviewed for this thesis revealed two main groups where researchers and practitioners are focusing their efforts. One group deals with very high-level considerations such as creating awareness for the environmental problem that comes with the increase in energy consumption of the ICT sector, or the development of green business strategies where organizations can employ alternative ways for offsetting their environmental impact without tackling the software energy efficiency directly. The other group deals with improvements at a very low level. These focus on areas such as power profiling tools, scheduling and power management strategies.[2]

In addition to have a crisp definition of software energy efficiency and the software characteristics that influence it, it is important to understand how this quality aspect of software influences, and is influenced by, other quality aspects of software, such as performance, security or maintainability. It is likely that increasing energy efficiency will have a positive impact in some quality aspects as well as a negative impact in others. For example, when optimizing software for performance by re-writing parts of its source code in a way that it will execute faster, one many times sacrifices the understandability (thus also the maintainability) of the code. Trade offs such as this are a common place in software engineering.[3].

Not having found a suitable conceptual model to depart from, this research will have to develop its own conceptual model.

2.4 RESEARCH QUESTIONS

From the elaboration and goal that where set above, the following research question is formulated:

- `Research Question:` *How to develop a practical model that will have the ability to evaluate the energy efficiency of software systems?*

The preceding question can be divided into the following sub-questions:

- `Research Sub-question 1:` *What are the aspects that determine the Energy-Efficiency levels of a Software System?* The answer to this question is presented in Chapter 5.
- `Research Sub-question 2:` *How can these aspects be quantified in a meaningful way?* The answer to this question is presented in Chapter 6.
- `Research Sub-question 3:` *How is it possible to combine and aggregate evaluation results so that they can produce actionable advice?* The answer to this question is presented in Section 6.3.

2 The reader is also welcome to navigate through Section 3.5 to Section 3.9 of Chapter 3 to find additional information on the topic.

3 An exploration of the relationship between energy efficiency and other qualities aspects can be found in Appendix B

3 BACKGROUND INFORMATION AND RELATED WORK

3.1 DEFINITIONS

3.1.1 *Energy and Power*

Although in many cases power and energy are used interchangeably in the literature, this is rarely correct and therefore, it is important to make a distinction between these two values for the sake of clarity. The conventional and simplified definition of energy is the *"capacity to do work"*. For computing systems energy is delivered as electricity and can be more simply described as Power over time.

$$\mathrm{Energy} = \mathrm{Power} \times \mathrm{Time} \tag{1}$$

Consequently, an energy efficient system requires less energy to deliver *"useful work"* than its energy-inefficient counterpart [75].

Power on the other hand, expresses the rate at which energy is transferred by an electrical circuit or more simply put, it is the amount of energy consumed per unit of time.

$$\mathrm{Power} = \frac{\mathrm{Energy}}{\mathrm{Time}} \tag{2}$$

3.1.2 *Energy Efficiency*

Energy efficiency can be roughly defined as the ratio between useful work to the energy consumption that is required to sustain this production. Hence, an energy efficient system is a system that consumes the minimum amount of energy required to perform any task.

$$\mathrm{EnergyEfficiency} = \frac{\mathrm{UsefulWork}}{\mathrm{Energy}} \tag{3}$$

As can be seen from the above equation, an increase in energy efficiency can be achieved by either reducing the energy consumption for a fixed level of work or by delivering more work for a fixed amount of energy. Of course increasing the ratio can also be achieved by increasing both the nominator and the denominator and therefore energy efficiency can also be enhanced by increasing energy consumption if and only if the work output justifies the additional energy expenditure. As such, sometimes it might be preferable to spent more energy in order to become more energy-efficient and thus increasing energy efficiency does not always imply a decrease in energy consumption.

In alignment with [41] [91] and Equation 3 the above relationship can be transformed to:

$$(3) = \frac{\mathrm{UsefulWork}}{\mathrm{Power} \times \mathrm{Time}} = \frac{\mathrm{Performance}}{\mathrm{Power}} \tag{4}$$

As a result, an increase in energy efficiency can be accomplished by increasing system performance within the same power budget or by exhibiting the same performance levels for a reduced power budget.

In any case and for the sake of clarity it is necessary to mention here that modern computer system reach their maximum energy efficiency level when they operate at peak utilization. This is justified by numerous research papers [6][74][76]. A quick explanation is that at this point (100% utilization) the energy efficiency and power-utilization curves meet ,as can be seen in Figure 1, and therefore the work output of the system becomes proportional to the energy that is dissipated (The energy efficiency ratio reaches its maximum limit).

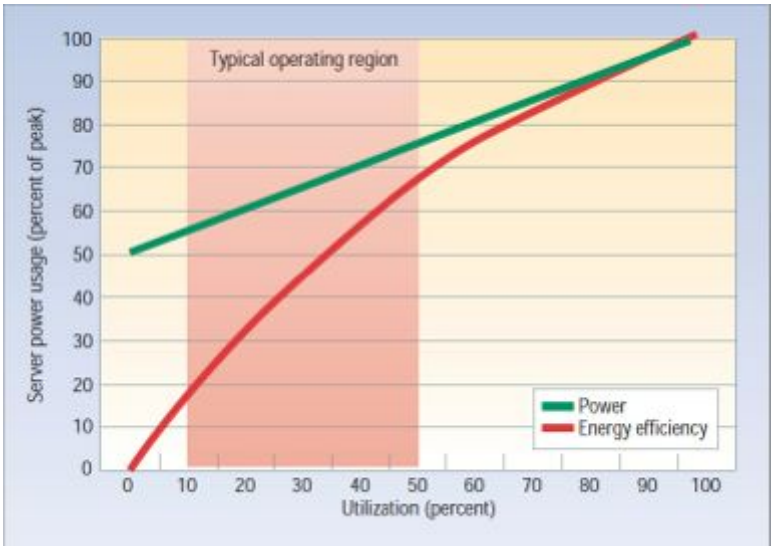


Figure 1: Power Usage and Energy Efficiency at various utilization level figure from [6] .

Of course this does not imply that all systems should operate at 100% utilization constantly, as in order to increase energy-efficiency, a concomitant increase in work output is implied. Since higher utilizations are translated to higher energy consumption but also higher production, the 100% utilization mark signals the point at which the per unit of work energy consumption reaches its lower limit. Therefore each unit of work that is produced at that point costs the least possible amount of energy for a specific system.

Of course this is only accurate if the additional energy consumption is accompanied by a proportional increase in work output or by a decrease in its production time. Therefore it is clear, that if the useful product of the system remains constant, an increase in utilization only leads in increased energy consumption and therefore lower energy-efficiency.

3.2 EVOLUTION TRENDS IN ICT

During the past decades there has been a remarkable increase in the performance of computational resources. This is consistent with `Moore's law`, which states that the number of transistors per square inch on a processor doubles every 18 months [63]. An adjunct to that law, is `Moore's law` for power consumption which declares that power consumption per computational node also follows the same trend [27].

Alhtough Moore's law predicts a steady increase in the levels of processor performance, the same does not apply when performance is related with the energy expenditure required to sustain this upward trend. In the supercomputer domain for instance, although performance has doubled more than 3000 times in the past two decades, performance per watt has increased only 300 times [101].

Of course, this is not only true in the case of supercomputers as the same tendency also holds for typical low-end servers, such as the ones used by Google, where performance and performance-per-server-price curves rose steadily during the course of the last decade whereas the performance-per-watt ratio remained almost immutable, although significant efforts where made to increase the power efficiency of the respective hardware resources [5].

Consequently, every increase in performance has been accompanied by a concomitant raise in the overall power consumption of the hardware nodes, with Barosso [5] estimating that if the performance-per-watt trend remains the same during the forthcoming years, energy costs will surpass hardware ones by a considerable margin. Of course, this does not imply that power management techniques have not evolved, as modern hardware resource consume significantly less energy than their older counterparts. This trend just points out that energy related optimizations have evolved more slowly than performance related ones, a fact that is clearly imposed by the energy-agnostic way that software is developed and used. According to Wirth's Law [99]: *"Software is getting slower more rapidly than hardware becomes faster"* and as such it also becomes more energy demanding as advances in hardware resource performance and power management have been impended by the energy demand and resource greediness of modern applications [36].

This delay in Moore's has created implications regarding the management of modern computational resources, as power management mechanisms should balance the benefits of reduced energy consumption to their respective impact on performance [5], and has also created the need for more energy-efficient software. This can be seen in Figure 2 [101][95].

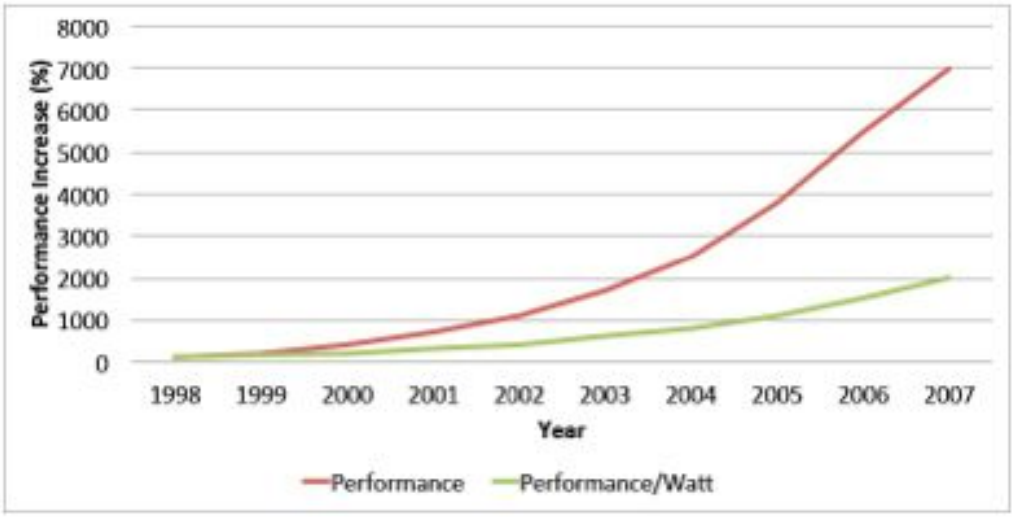


Figure 2: Evolution of Performance and Performance per Watt figure from [101] .

3.3 THE ENERGY PROBLEM IN NUMBERS

The energy impact of the ICT in the global energy footprint is evident. Estimates suggest that ICT is responsible for 3% of the total energy consumption worldwide and that within the U.S, data centers consume more than 2% of the electrical supply nation-wide [61]. Koomey et al., in [50], estimated that the cost of operating and cooling servers, in 2005, was 7.2 and 2.7 billion dollars, worldwide and in the US respectively. His estimation for 2010 [46] suggested that the energy expenditure of the ICT lay between 1.1% and 1.5% of the global energy consumption. Of course, regional trends also exist, as according to [4] data centers accounted for the 10% of the total electrical consumption in the Amsterdam region.

These costs are not steady, as the energy expenditure presents a remarkable upwards inclination. The U.S Environmental Protection Agency (EPA) reported that in the U.S. the energy consumption of data-centers rose from 28 billion kWh in 2000 to 61 billion kWh in 2006, which is translated to $4.5 billion, accounting for 1.5% of the total energy usage in the states [24]. The trend was the same on a global scale as data centers consumption increased from 58 TWh in 2000 to 123 TWh in 2005 [50], following a constant upwards trend.

Of course, computational resources are not the sole energy contributor of the ICT. Guerra et al. [38] suggested that storage equipment accounts for almost 37-40% of the energy consumption of all data center components, with the authors expecting this trend to continue. Furthermore, in [77] it was estimated that the annual energy consumption of networking equipment lays between 6-20 TWh. Ancillary, to the main operational equipment resources also have a significant impact in energy expenditures, as according to Patel et al. [69] for every 1W used by typical servers, 0.5 to 1W is required for operating the necessary cooling equipment.

The wasteful practices of the past as well as the ever-increasing demand for ICT resources has transformed energy management into a major consideration in most ICT domains, as people have finally understood that the world is limited and resources are not in abundance. The implications that were bred by this energy-agnostic expansion are manifold and thus, energy conservation is gaining importance not only for reducing the steadily increasing energy costs, but also for dealing with the ramification that these costs have on the environment as the ICT has been linked with phenomena such as global warming.

The Climate Group[1], reported that ICT was responsible for 2% (830MtCO2e) in carbon emissions worldwide in 2007, a number which is expected to rise to 6%(1,430 MtCO2e) by 2020 [86]. As it is suggested in [23], the energy consumption of the ICT has been estimated to produce 4 million tons of carbon dioxide annually.

ICT energy costs and carbon emissions are not limited to the data center context, as this domain is expected to account for less than 20% of the global carbon emissions caused by the ICT in 2020, with personal computers and ancillary equipment being responsible for the majority of the consumption [86]. According to Nordman et al. [67], personal computers required 100 TWh annually, a considerable 3% of the total electricity usage in the US for 2009.

1 http://www.theclimategroup.org/

3.4 ENERGY CONSIDERATIONS AND SYSTEM TYPES

The energy problem becomes more apparent if one considers the vast heterogeneity of computing equipment and the variety of contexts in which it is used – e. g.laptop and desktops machines for personal and enterprise use, embedded systems such as smartphones; data centers of all sizes; large computing clusters for scientific purposes [1] –. Therefore, energy efficiency has an undeniable impact in almost all computational domains.

For embedded systems, battery life has long been an issue which have constraint evolution on that domain, with costumers constantly requiring more, as recharging might not be possible any given time and no significant improvements in battery capacity [3] were recently made. Data centers, which have grown remarkably in number and scale due to increasing demand of services and cloud computing, are constraint by power and cooling issues and crucial design limitations such as peak load capacity has long been a driving factor in deployment and operations, as a simple drop in power provisioning might be translated to large customer dissatisfaction. The raw energy costs in that domain also impose a clear cut constraint regarding data center scalability, as they are a major part of the total operational costs and a pressing concern [25]. Furthermore, power demand has started to reach a point of imposing physical constraints, as in some cases such as the one mentioned in [13], `Morgan Stanley` could not physically get the power required to sustain the operations of a new data center in Manhattan.

For large computing clusters, the processing density required to deliver the necessary performance to leverage intensive computations, has been constraint by reliability and compaction issues as thermal dissipation due to the tremendous increase in power density has become a clear limiting factor that can cause serious degradation to hardware resources. In regular desktop machines energy dissipation will continue to rise as long as demand increases. Thus, reducing energy consumption and increasing the efficiency with which energy is distributed and used across all computing domains has become an important area of focus.

3.5 ENVIROMENTAL ORGANIZATIONS AND GOVERNMENTAL AGENCIES

The increasing energy consumption of the `ICT` has inevitably lead in the formulation of recommendations, best-practices, legislations and incentives from many environmental organizations and federal or governmental agencies such as the `U.S. Environmental Protection Agency (EPA)`[2] , the `Intelligent Energy Europe`[3] and `TopRunner`[4] which aim to lower `ICT` energy consumption and carbon emissions and promote the use of renewable energy and therefore protocols such as `GHG`[5] where developed for that purpose.

A representative regulation is given in [61]. According to that, `the Carbon Reduction Commitment Energy Efficiency Scheme (CRC)`[6] in the United Kingdom (UK), enforces organization that consume more than 6000 MWh annually to participate in a carbon trading scheme, with lower consuming organizations.

2 http://www.epa.gov/
3 ec.europa.eu/energy/intelligent/
4 http://www.eccj.or.jp/top_runner/index.html
5 http://www.ghgprotocol.org/
6 https://www.gov.uk/crc-energy-efficiency-scheme

3.6 INDUSTRIAL RESEARCH

Industrial research on the topic, has proven fruitful mostly on monitoring the consumption of their respective products and in providing documentation regarding best-practices on the field. The most notable contribution – based on the research presented here – has been done by Intel[7] which has published extensive documentation on energy-efficient practices at many levels of the problem stack (data centers, application development etc.) [82][83][53] and has taken part in projects like PowerTOP[8] a tool that is able to measure battery dissipation for laptops running Linux distributions. Intel has also developed the Energy Checker SDK[9] which can be used to relate energy consumption to useful units of work. It offers an API that can be used for importing and exporting counters, which can track specific events that other applications can use in order to take necessary actions in order to adjust their energy behavior. As such it can be used for relating energy consumption to delivered functionality but suffers from the limitation of needing additional software instrumentation.

IBM[10] has been focused on publishing documentation but is also selling tools, such as the IBM Systems Director Active Energy Manager[11] , that aims at helping organizations to monitor and manage the energy consumption of their software systems. Dell[12] also provides documentation regarding energy efficiency and power management and has been responsible for the Energy Intensity metric [21], an indicator that can be used to relate energy consumption of hardware resources to work unit production.

Microsoft[13] has also been particularly active on the topic [48][47], and has produced Joulemeter[14], a free tool that is able to measure the energy consumption of software applications running on Windows 7 platform. This tool uses a power model that tracks CPU usage and disk CPU I/O of each application running on the platform in order to estimate their power consumption. Furthermore, Microsoft has provided documentation and recommendations on how to reduce energy consumption and increase efficiency for systems using its operating systems in both mobile and desktop domain.

Finally, both Apple[15] and Google[16] have presented research on the topic, with Apple mainly focusing on its mobile devices and Google publishing extensive research regarding its data-centers [25][6][5][43].

3.7 MONITORING ENERGY CONSUMPTION

On a non-industrial context, research has also been done on monitoring and estimating energy consumption in a holistic fashion, by relating software behavior to energy dissipation. POWERAPI [68] is an OS-level library that can be used to estimate energy consumption at a process level by using specific system counters. POWERSCOPE [29], is another energy profiler that uses statistical sampling and is targeted at the process level and as

7 http://www.intel.com/content/www/us/en/corporate-responsibility/energy-efficient-computing.html
8 https://01.org/powertop/
9 http://software.intel.com/en-us/articles/intel-energy-checker-sdk
10 http://www.ibm.com/ibm/green/
11 http://www-03.ibm.com/systems/software/director/aem/
12 http://www.dell.com/content/topics/topic.aspx/global/products/optix/topics/en/optix_energy?c=us&l=en
13 http://www.microsoft.com/environment/products-and-solutions/GreenerIT.aspx
14 http://research.microsoft.com/en-us/projects/joulemeter/default.aspx
15 http://www.apple.com/environment/energy-efficiency/
16 http://www.google.com/green/efficiency/

such it can relate software activity to energy dissipation. Nevertheless, POWERSCOPE does not estimate the energy consumption of a process but actually measures it, by relating lower-level system counters with actual energy measurements derived by a multimeter and thus it requires additional hardware instrumentation.

PowerBooter [102] is a power modeling technique, targeted for Android mobile devices. It uses the voltage sensors of the device's battery and information on its discharging behavior in order to monitor consumption of components potentially acting as a power manager for the device. PowerTutor is an ancillary tool that can use the models that are generated by PowerBooter to make online power estimations. This tool uses Advanced Configuration and Power Interface (ACPI) power readings and combines them with machine learning techniques in order to enhance its accuracy. pTop [22] is another process level power profiler which offers a Linux TOP-like display and can provide information on power consumption of all the processes that run on the respective platform by using system counters derived from the /proc directory.

Various research efforts have been focused in measuring/estimating the energy consumption of hardware components based on knowledge derived by system performance counters and resource utilization information, however they fail to relate energy expenditures with application behavior and as such they can not be used for holistic evaluations. An indicative effort is Mantis [23], a method for modeling full-system energy consumption that can be used to provide real-time power prediction.

3.8 EVALUATING ENERGY EFFICIENCY

The data-center domain has been the cradle of extensive research regarding energy efficiency, which is quite logical since it is the domain that is affected more heavily by the energy problem. Therefore, several metrics, frameworks and best-practices have been proposed in that context with most notable contributors being the Green IT Promotion Council[17], an industry-government-university partnership under the Green IT Initiate of the Japanese government, the Uptime Institute[18], a consortium of companies and consultants specializing in the data-center domain, the Nomura Research institute[19], Japan's largest consulting firm, the Emerson Corporation[20], a leading solution provider targeted at many domains, the Green Grid[21], a non-profit industry consortium aiming at increasing energy efficiency of data-centers and the GAMES[22] framework which was incorporated in the EU Project Games regarding Green IT. A more thorough presentation on the last framework can be found in [84][49]. For all the other initiatives a concise presentation of the metrics proposed by them, can be found in [56][64].

The steadily increasing energy consumption of the ICT has also created the need for energy benchmarks and organizations like the Transaction Processing Performance Council (TPC)[23] [72] [71] (a non-profit organization responsible for defining transaction processing and database benchmarks), the Standard Performance Evaluation Corporation (SPEC)[24] (a non-profit corporation responsible for defining and maintaining benchmarks relevant to high-performance computing), and the Storage Performance Council (SPC)[25]

17 http://home.jeita.or.jp/greenit-pc/e/
18 http://uptimeinstitute.com/
19 http://www.nri.co.jp/english/
20 http://www.emerson.com/en-US/Pages/Default.aspx
21 http://www.thegreengrid.org/
22 http://www.green-datacenters.eu/
23 http://www.tpc.org/tpc_energy/default.asp
24 http://www.spec.org/benchmarks.html#power
25 http://www.storageperformance.org/home/

(a non-profit corporation founded to define and standardize benchmarks relevant to storage subsystems), answered to that call.

All of the preceding models, frameworks and benchmarks suffer from various limitations and thus they are not more extensively presented here. Most of them are confined in the data center context and are unsuitable for assessing heterogeneous system types since they do not produce results that can be comparable and meaningful across a wide spectrum of modern computational systems. Others are extensive frameworks or require extensive hardware or software instrumentation which can significantly increase the overhead of applying a model. A fact that renders them impractical. Furthermore, as `Mahmoud et al.` [56] states, many of the metrics and models that currently exist, do not present a clear way regarding their application (it is not clear how to apply them) whereas we believe that in many cases interpretation is even harder, as it is rather trivial on how to act on reducing energy waste, based solely on the values that are produced by some of them. Therefore, they are not sufficiently actionable.

As is stated by `Williams et al.` [98], there is no widely accepted framework that will be easily applicable and suitable for making clear-cut assessments even at certain domains such as the data center context, with `Mahmoud et al.` [56], indicating that the current metrics or frameworks, if used improperly may lead to contradicting conclusion and as such no easily-applicable and widely accepted framework that will be able to assess efficiency, at least at a certain context, is out there.

3.9 INCREASING ENERGY EFFICIENCY

Hardware domain

Over the past decades manifold efforts have been made on the topic from a hardware perspective with the primary focus being the processor of a software system. Techniques such as `clock gating` and `DVFS` (Dynamic Voltage and Frequency Scaling) [54] for runtime optimizations have been the center of extensive research and have lead to significant energy reduction on the processing domain. These techniques focus on providing lower-power modes, that are operational, with a corresponding decrease in performance. For instance, `DVFS` is used to lower the frequency with which the processor executes instructions, signaling in return a corresponding decrease to the Voltage that is required for the resource to function. According to `Holzle et al.` [43] since low utilization regions are more frequent and last longer than complete inactivity, active-low power modes can lead to considerable energy savings.

This resulted in `CPUs` not being the primary energy contributor for many hardware resource types, such as storage servers for instance, [90] and therefore extensive research such as phase changing memory, solid state disk drives, lower rotational disk speeds, and ethernet technologies which have the potential to change speed and therefore consumption based on network traffic [66] have lead in significant reduction in the energy consumption of other hardware resources. Consequently, many modern devices now provide lower-power active states, which aim at reducing energy consumption and no single hardware resource dominates the energy usage according to [91].

State Transitioning

Although, lower power active modes have been researched extensively, a significant amount of effort has also been focused on accurately transitioning hardware resources to inactive states which require less energy than active ones, until those resources need

to be accessed again [40]. This transition to lower power states usually occurs after some time of inactivity and the transition back to an active state happens according to some fixed schedule, algorithm or usually on demand.

A general approach that is described in [37], consists of two steps and requires collaboration between power modeling as well as algorithmic optimization techniques. According to this methodology, model based estimation techniques are used to attribute distinct power values among idle and active states. This values are then used by optimization algorithms, combined with information regarding activation costs, to choose the most suitable execution path based on application workload.

However, the transition from an inactive to an active state imposes a certain amount of overhead, in terms of energy consumption thus, it might not be worthy to power down hardware components if the general system context exhibits a certain level of instability. Web services for instance, that might exhibit a seemingly random workload pattern, as web workloads are evidently unpredictable and bursty in nature [10], might not be suitable candidates for powering down parts of the infrastructure or specific hardware resources. Moreover, there is an additional latency penalty due to state transitions. Barosso et al. in [25] stressed that the aforementioned penalties make an inactive energy savings mode less useful for servers, by demonstrating that *a disk drive in a spun-down, deep-sleep state might use almost no energy, but a transition to active mode incurs a latency penalty 1,000 times higher than a regular access latency.* Nevertheless, as it is pointed by Heath et. al [42], this strategy usually works sufficiently when there is enough idle time to justify the transition costs. This approach also seems to yield satisfactory results in the (Mobile, embedded context) where usually screens and hard drives become inactive after a period of idleness [56].

Idle energy consumption

Transitioning to lower power modes or idle states is not a panacea as the vast majority of modern system's will have a fixed static energy consumption even at those states. Dawson et al. studied the idle power consumption of various server machines and observed that it was on average 50% of their peak. This consequently means that a server with low utilization will be very power-and in extend energy-inefficient [20]. Furthermore, in [93] it was observed that idle servers usually consume 20-50% of their peak power at 0% utilization, while Kansal et al. [47] stressed the need for energy optimizations targeted at low component utilization and idle power consumption by demonstrating that the typical idle power consumption of a personal computer is usually 50-60% of its peak power.

Barosso et al. [25] examined the idle consumption of servers for five typical configurations deployed in Google Data Centers. He observed that idle power is significantly lower than peak, but generally never below 50% of peak. In order to further, investigate the impact of lower power states, he conducted an experiment in which the idle power consumption of every server in a cluster was altered to 10% of the peak value. He observed a 30% drop in maximum power consumption while energy consumption was reduced by half. In a later publication, Barosso et al. [6] suggested that in the data center context most servers operate at 10-50% of their theoretical peak performance. Therefore, idle consumption holds a central role in energy-efficiency literature and a huge percentage of optimization efforts aim at reducing this portion of energy consumption.

Software domain

While hardware optimizations have been manifold and have proven useful in reducing the energy footprint of modern systems, as components have drastically increased their ability to consume energy in proportion to their usage [41], it has become apparent that these techniques by themselves are not sufficient and higher-level strategies are necessary in order to increase energy-efficiency and enhance the proportionality levels of software systems [39][12], especially since there has been a growing realization that software choices and characteristics are the ones that ultimately determine to a large degree the energy efficiency of a software system. Therefore, a portion of the research interest has shifted into identifying optimizations at an application level.

Matthew Garrett [31] was one of the first researchers that identified energy-inefficiencies caused by software. One of the major energy contributors according to Garret are fixed-tick schedulers of operating systems that fire interrupts periodically even if no task is scheduled to run. Those interrupts prevent the processor from idling or entering other inactivity modes.

This behavior also persists in the application domain as according to Garret and Eric Saxe [79], many applications set timers that upon expiration result in processor wake-ups. Consequently, any unnecessary software poling behavior can be considered harmful with respect to energy efficiency.

According to Saxe, other sources of waste that originate from the application domain are for instance, keeping memory mapped even when no longer needed, or generating an excessive number of I/O requests. Of course, all of the above can be leveraged by restructuring the application or the operating system, in order to be event-driven and not time-driven, or poll infrequently if the previous shift is not possible, free or unmap memory when not needed and batch I/O requests whenever and wherever possible [79].

Matthew Garrett [31] is also responsible for the – race to idle – principle which is in conjecture with the mantra that *"maximum performance is green"* and has played a central role in energy efficiency related optimizations. Garrett observed that running a task at full processor speed and then idling is more power efficient than executing the same task for a bigger amount of time in lower processor frequencies, since the power consumption of an idle processor is usually much lower than an active one running at lower speed. Therefore, it is usually preferable to run the task fast and quickly idle, than execute it at a lower frequency for a bigger amount of time. Other research on the topic has also confirmed the benefits of idling fast. Chen et al. [16] for instance, observed energy saving of 16%, when applications finish quick allowing hardware resources to idle faster and eventually be turned off.

Hardware components require some minimum time-frame in order to switch to idle state and eventually power down as it was described above. This time-frame is usually determined by the variance of the system workload. This consequently means, that power management techniques implemented at a software level are needed in order to leverage the advantages of low-power or idle hardware modes. Since the idleness in workloads is ultimately determined by applications, software systems that can adjust their workloads to create more opportunities for power management are needed [87]. Such applications are called energy-aware applications and the need for software systems that are energy-aware is eminent in order to attain high levels of energy efficiency. An energy-aware data processing application for instance may switch from frequent I/O communication to batching its disk reads in order to create such periods and allow the hardware resource to be powered down in order to save energy. This technique has been explored for instance by Weissel et al. [96].

According to Barroso et. al. [43], in the distributed system domain, it is a common strategy for software to spread data and computations across different resources, reducing in that way the existence of long idle periods for the respective hardware resources. Thus, energy management software must create such idleness in a way that does not interfere with other quality characteristics of the system such as availability. In the data center context however this can be quite challenging as the workload variance ultimately determines the frequency with which idle time frames occur and therefore the opportunities that the hardware resources have to enter some lower level sleep states are scarce [6][43][60]. Consequently, energy saving techniques that aim at increasing the level of energy efficiency apart from targeting the hardware or software stack of the problem can be also determined by the system type and the general system deployment and context. Specific techniques for instance that are targeted in the data-center domain focus on *ensemble level* optimizations [90][93], while other system level approaches focus on minimizing the energy consumption of servers during idle periods [59].

Ranganathan in [73] also offers a list of common inefficiencies as well as ways to deal with them. Many of those, such as that system layers usually ignore the power saving capabilities of the others and therefore do not utilize them, originates from the software domain.

Noureddine et al. [68], tried to relate energy consumption to algorithmic constructs and programming languages. They observed that recursive implementations written in C++ usually consume less energy than their iterative counterparts, although this difference disappeared when compiler optimizations where deployed. They then tried to investigate the impact of programming languages on energy consumption and discovered significant differences in consumption for the same algorithmic implementations, with Perl recursive being the most energy demanding and C++ recursive presenting the least energy expenditure.

Sahin et al.[78], investigated the impact of design patterns on the energy consumption of an application. They divided design patterns in three categories (Creational, Structural, Behavioral) and chose five patterns to evaluate from each. They found that differences in consumption between design-pattern and non-design pattern implementations exist for all the 15 design patterns that where chosen for assessment, however this difference was not consistent in nature(some increased while others decreased energy dissipation).

4 RESEARCH METHOD

In this chapter the steps followed in order to fulfill the goal of the project are explained to let the reader understand the process that led to the final solution. Based on the problem statement (Chapter 2) and the information gathered during the literature survey (Chapter 3), a suitable research method was chosen.

4.1 METHODOLOGY

The methodology used in the this project is a combination of an exploratory and a constructive approach. The exploratory nature of the research lies in the fact that in the beginning of the research it was relatively unknown whether or to what extent it was possible to establish an effective practical model, which would be able to evaluate the energy efficiency of software systems, due to their broad heterogeneity. Furthermore, it has proven difficult to identify and quantify all the aspects and factors that have an impact on energy efficiency, as no conceptual division of the quality was identified during the literature survey. Hence, an exploratory approach was desirable.

On the other hand, since work was carried out towards the establishment of a practical solution – *a practical model that will evaluate the energy efficiency of a software system* –, this research is also constructive. The initial goal, was to provide quantitative data to enable evaluation and comparison between different software systems. An ideal case would be to create thresholds and values for every aspect identified through system benchmarking. However, since time and resources were limited and empirical data on this context are scarce, creating a practical model with calibrated thresholds for quantitative assessment was infeasible from the begging, hence evaluation was primarily more qualitative in nature.

4.2 APPROACH

The project followed a top-down approach. An initial conceptual division of energy efficiency into sub-characteristics was first made. This break-down was based on findings from the literature and feedback from `SIG` researchers and consultants.

The *"Goal Question Metric approach"* [7] was used as the basis for the design of the practical model. This approach defines a measurement model in three levels:

- `Conceptual`: define a goal.
- `Operational`: identify questions to fulfill the goal.
- `Quantitative`: find metrics that produce the right data to answer the questions.

There where two reasons for choosing the `GQM` approach. First, it is suitable to guide the construction of a minimal model since it promotes a top-down model development approach. Second, this methodology enables early interpretation of the results since the opposite approach (bottom-up) can be followed after its application, to map metric results to their respective questions and sub-characteristics.

The GQM parameters include the purpose (what objects and why are measured), perspective (what aspect and who) and the context characteristic (where). Consequently, the Research Goal with respect to the GQM approach was structured in the following way:

- Purpose: *Analyze a software system for the purpose of evaluation,*
- Perspective: *with respect to energy-efficiency from the point of view of system stakeholder or SIG consultant,*
- Environment: *In the context of short term projects carried out at SIG.*

The research goal was then divided into sub-goals, which evaluate each of the characteristics that constitute energy efficiency. Then, each sub-goal was characterized by a set of question, which in turn can be answered by data collected via specific metrics.

As expected many iterations followed the initial application of the GQM methodology. Sub-characteristics of energy efficiency changed as well as the questions for assessing them, as more knowledge was gained on the topic. The fact that there is no general consensus of the sub-characteristics that constitute energy efficiency was another reason for iterating. Having defined and operationalized a model, the next logical step is its application. To this end, we defined an experiment for collecting the necessary input data. Furthermore, an aggregation technique and a rating methodology were developed.

4.3 EXPERIMENT SETUP

The system that acted as a source to evaluate the model was that of the Software Improvement Group. That system consists of five major software components that are responsible for delivering two main functionalities: *analyzing client source-code; enabling clients to monitor the evolution of their applications.* The necessary input data that is required by the model, represents the usage of the host hardware resources by those software components, information about their work output as well as their respective energy consumption.

Direct energy measurements proved infeasible in the context of this research and therefore a linear power model was created in order to derive the necessary energy consumption information.

Since a production system was chosen as source to evaluate the model, any potential interference from the data collection methodology could create ramifications. We have considered the use of commercial tools such as Linux TOP to derive a portion of the required data. This thought was abandoned for two reasons. First, tools like Linux TOP, track resource consumption of every process. Although we can filter the displayed information, we didn't know for sure if that would cause less overhead. Second, such tools only track individual processes, whereas we needed to track a set of processes that together form an application. Given the above, we scripted our own lightweight monitoring tool.

The implementation of the following scripts was necessary in order to derive model results.

1. A script to analyze the output of the data collection script in order to derive CPU as well as software component utilization statistics.
2. Two scripts to analyze software component log files in order to derive information regarding their work units.

3. A script to combine and consolidate the information that is collected from the preceding scripts.
4. A final script to calculate the model metrics.

Finally an aggregation and a rating methodology were developed so that a system rating regarding energy efficiency can be derived from metric results.

At an initial stage, some preliminary measurements were applied in order to test the quality of the results, the relevance of the model indicators and the data gathering methodology. Once this step was completed and both model indicators and data gathering and analysis scripts were tested at a satisfactory level, a formal measurement took place in order to fully apply the model. A more thorough description of the system, the process that was followed and the results can be found in Chapter 7.

4.4 EVALUATION

A quantitative validation for this model is infeasible within the scope of this project. This is because the results that were produced cannot be compared against some absolute and correct value. Furthermore, it was know before hand that we would only have a very limited data set at our disposal. Mainly, the data set that was derived by the case study that was conducted. Consequently, concrete threshold values and an aggregation methodology that would map metric values to the overall energy efficiency of the system could not be developed in a statistical sound fashion.

Since a proper validation was out of scope from the beginning of this research, it was decided that an evaluation path should be followed instead. Consequently, we chose to evaluate this effort based on how well it satisfies the goal that was set. If the model is able to produce results that will reveal unnecessary energy waste and point to optimization opportunities then it can be considered to some extent successful. This is because it will have the potential to inform evaluators, at least at some degree, as to how energy efficient a software system is.

Another evaluation input will be the extent to which the model abides to the requirements that where set in (Section 6.1). If the model satisfies the majority of those requirements then it should also be considered successful, as it can be potentially act as a basis on which future research can build upon.

5 CONCEPTUAL MODEL

This chapter conceptually divides energy efficiency and presents the sub-characteristics.

5.1 ENERGY EFFICIENCY AS A SOFTWARE QUALITY

A first step in assessing the energy efficiency of a software system would be the conceptual division of the quality. This would divide it in smaller and more manageable parts. Concepts are valuable, as they offer an effective way of structuring knowledge and are consisted in turn by a number of characteristics which can be more accurately and effectively assessed. Nevertheless, as it was stated in Section 2.1, energy efficiency has not yet been elevated to a point where is considered equal among other fundamental software qualities, and awareness that it should has only recently increased. Consequently no research that aimed or managed to divide this quality into its prime concepts was identified.

In the context of this model, the conceptual model of energy efficiency is adapted from the corresponding performance efficiency conceptual model as presented in the `ISO/IEC 25010` [44]. There are several reasons for following this approach. First, both those qualities share a dynamic nature and are heavily related to resource demand and usage. Second, both qualities are heavily related due to the race to idle principle and the maximum performance is green mantra, which holds a central role in the energy efficiency context [31][16]. What those two principles essentially state, is that running a task at full processor speed and then idling is more energy efficient than executing the same task for a bigger amount of time in lower processor frequencies, since the power consumption of an idle processor is usually much lower than an active one running at lower speed. Therefore it is preferable to run the task fast and quickly idle, than execute it at a lower frequency for a bigger amount of time. Third, we observed that all the performance efficiency sub-characteristic that are defined in `ISO/IEC 25010` are meaningful in the energy efficiency context, as long as the time variable of those characteristics is substituted by energy.

Finally, many efficiency related qualities share many commonalities, as in general, efficiency deals with identifying the optimal (from the respective quality point of view) configuration that can support operations.

Dividing energy efficiency conceptually, sanctions a more practical application of the `GQM` method (Section 4.2). The conceptual division that is proposed bellow gives an answer to `Research sub-question 1:` *What are the aspects that determine the Energy-Efficiency levels of a Software System?*(Section 2.4). These are the main aspects that we believe are necessary in an evaluation of energy efficiency. Other aspects, such as environmental impact are not included in this conceptual division on purpose and are considered out of scope.

5.1.1 *Definition*

According to `ISO/IEC 25010` [44], performance efficiency is defined as *"performance relative to the amount of resources used under stated conditions"*. Adapting from this definition

we define energy efficiency as:

"Energy consumed relative to the amount of resources used under stated conditions"

Other definitions of efficiency that where identified during the literature survey convey the same semantic meaning and some of them are presented below to justify the preceding definition. Those definition can be found in [58].

According to Boehm, efficiency is defined as *"extent that software fulfills its purpose without waste of resources"*. Gilb introduces energy and useful work into the efficiency context by defining efficiency as*"the ratio of useful work performed to the total energy expended"*.

The stated conditions that are mentioned in the definition that is used by this model can be regarded as requirements or constraints and are external to it. That means that they do not interfere with the evaluation results produced by the model.

5.1.2 *Conceptual Division*

According to ISO/IEC 25010 [44], performance efficiency is divided into the following sub-characteristics:

- Time Behavior: Degree to which the response and processing times and throughput rates of a product or system, when performing its functions, meet requirements.
- Capacity: Degree to which the maximum limits of a product or system parameter meet requirements.
- Resource Utilization: Degree to which the amounts and types of resources used by a product or system, when performing its functions, meet requirements.

Based on, and in alignment with the conceptual division of performance efficiency, the characterization of energy efficiency in the context of this model is the following:

- Energy Behavior: Degree to which the energy consumed by a product or system, when performing its functions, meet requirements.
- Capacity: Degree to which the maximum limits of a product or system parameter meet requirements.
- Resource Utilization: Degree to which the amounts and types of resources used by a product or system, when performing its functions, meet requirements.

The above concepts act as constituents of energy efficiency. Therefore the main goal of the model, which more simply expressed is to: *evaluate the energy efficiency of a software system* can be divided into the three distinct sub-goals of evaluating all of the preceding concepts.

5.2 RATIONALE

This section more thoroughly describes the sub-characteristics of energy efficiency that where presented above.

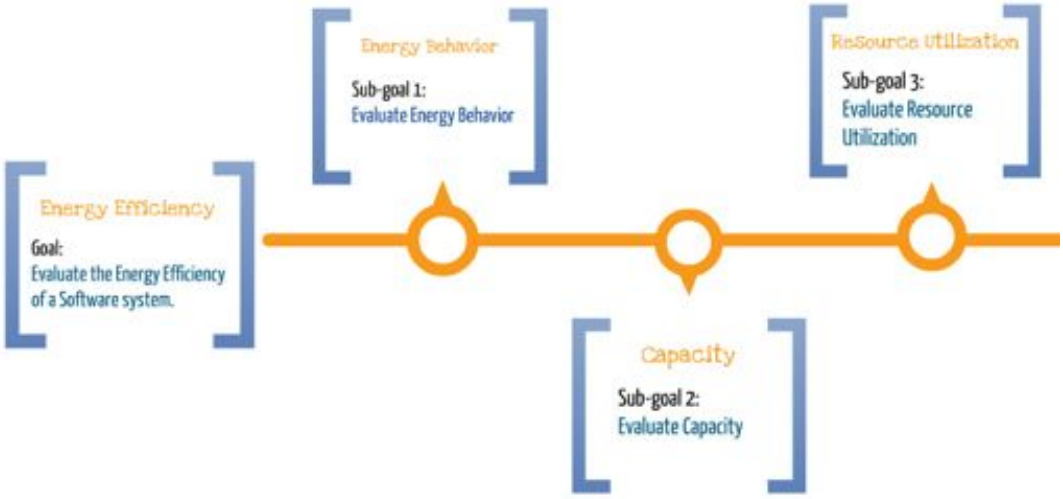


Figure 3: Conceptual division of Energy Efficiency

5.2.1 *Energy Behavior*

The Energy Behavior concept is concerned with the amount, type and place of energy expenditure. Thus, by evaluating energy behavior answers will be given as to where, how and to what extent energy is dissipated when the system operates. The goal of assessing Energy Behavior is to understand:

1. The energy consumption of software components in order to identify the major energy contributors.
2. The energy consumption of each work unit that those components deliver.
3. The type of this energy consumption. Do software components consume energy to produce work or is it spent most in the idle state?

Description

The energy consumption of a software system that hosts an application is dependent on a plethora of factors both internal and external to it. On the one hand, internal factors that affect this consumption are specific hardware resources used by an executing application, their internal configuration and characteristics (e. g.`CPU` frequencies), their topology and functional suitability. On the other hand, external factors consist of performance or other `QoS` requirements, the variability, volume and type of the workload and any other financial or environmental regulation that might impose some kind of constrain to the way the system operates.

This energy consumption can be attributed to software components that use hardware resources in order to deliver functionality and in extent are subjected to the preceding constraints. Thus, the energy consumption of a software component can be calculated by summing the consumption of hardware resources that are used by it in order to produce *"work"*. The overall energy consumption of the system hosting an application can be aggregated by summing the energy consumption of every software component that is part of it. Consequently it is feasible to identify the major energy consumers

among a set of software components which is the first objective of the Energy Behavior sub-characteristic.

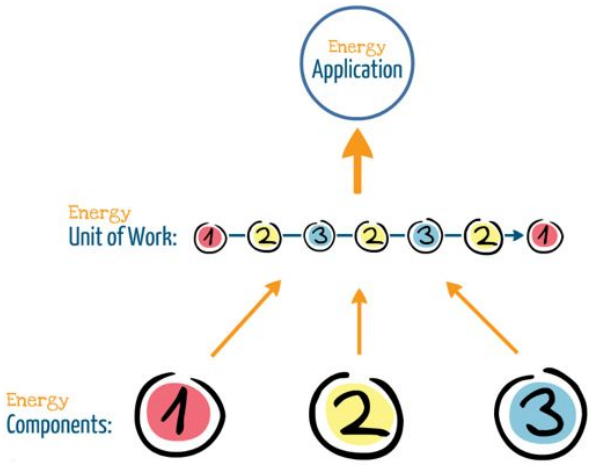


Figure 4: Relationship between Software Components, Unit of Work Steps and Application Energy Consumption

The energy expenditures of a system can also be related to produced functionality and workload in a more understandable way than simply expressing software component consumptions. A workload is a logical grouping of work performed on a computer system. For the purpose of this model and in order to associate energy consumption with delivered functionality it is useful to describe the system workload through the perspective of a useful unit of work [4]. According to [19], a useful unit of work is a measurable quantity of work done, as opposed to the amount of system resources used to accomplish that work. Workloads can be classified by a wide variety of criteria such as: *who is doing the work; what type of work is being done; how the work is being done.* Consequently, for each system a useful unit of work can be derived from its main functionality.

For the purpose of this model a useful unit of work is considered to be an abstract layer, between application and software component consumptions (Figure 4), in which energy costs can be expressed. Since the total energy consumption of an application can be denoted as the sum of the total application component consumptions, the same value can be obtained by summing up the energy that was consumed per unit of work delivered, during application execution. Therefore, unit of work and software component energy consumption can be related as they express the same overall consumption at an application level. Therefore, the energy consumption of a software component can be related to the work that this component produces. Figure 4 demonstrates that relationship.

Finally, it is important to distinguish between the different software component "states". For the sake of clarity and time, we focus only on two major states namely the `Active` and the `Idle`. For active states, an increase in system workload inevitably signals an immediate rise in energy consumption. Thus, the overall energy consumption of a system can be divided into two portions. The active portion which varies with utilization, and a static one, the idle consumption which is fixed, whenever the system enters this state [74][17]. Typically hardware resource have three major states: Active, Idle and Sleep and

each of those major states usually contains other minor ones. The StrongARM SA-1100 processor for instance, has 20 minor active and 10 minor idle states [94][1].

We chose to abstract from the hardware resource state categorization, because this would introduce much complexity mainly due to the vast heterogeneity that those resources present (different resources have different number of states). Moreover, it would only add slivers of value with respect to added overhead (the goal is to make a practical model that would be platform independent and Active and Idle categories provide sufficient information).

An active application or software component is one that executes in order to produce some useful work from the perspective of the end-user. Consequently, an application can be considered active when for instance, it plays some multimedia, or loads and displays a web-page, or calculates the payment slips of company employees etc. Thus, whenever an application or a software component uses the CPU or the GPU of its host hardware to process some kind of workload, then the energy consumed by this host is counted as active energy consumption from the perspective of the application.

On the contrary, an idle application is an application that is *"running"* without producing any useful work although its host resource might be active, executing instruction from other external software systems or OS routines. Thus, an application is considered idle if it does not allocate any CPU or GPU cycles. For instance, a music player that is stopped, or a video conferencing client that runs in the background but is not actively used, can be considered idle applications. The energy spent on that state, constitutes the idle consumption of an application and is energy which is by definition wasted [81].

From the above we can loosely define energy behavior, as the high level energy footprint of the application. Thus, energy behavior deals with the types of consumption (Active or Idle) and their impact on the total system consumption, with individual component consumptions (identifying main contributors), and with the energy expenditure when delivering useful work (relating energy consumption to functionality).

5.2.2 *Capacity*

The Capacity sub-characteristic is concerned with the difference between the theoretical energy provisioning capability of host resources and application or software component energy usage. To do so, it is important to identify how far does a system deviate from the *"cheapest optimal hardware and software configuration"* that can sustain production while attaining the highest possible levels of energy efficiency. This is done by identifying the amount and type of hardware resources that are required to host an application in order to deliver desired levels of output for given workload characteristics, volume and distribution, with the least possible energy consumption.

More simply put, an evaluation of the Capacity sub-characteristic aims at shedding light onto the following:

- "How much energy can the system provision compared to what is actually used by an application or software components during:
  - Peak workload
  - Average workload

1 Specifications of systems using lower power states can be found for instance in the Advanced Configuration and Power Management Interface(ACPI) which aims at providing industry-standard interfaces in order to enable power and thermal management of modern desktop, mobile and server systems.[1]

Description

An application running on a specific platform will have a fixed amount of power capacity (that is a theoretical limit of *"available"* power) which it can exploit in order to produce useful work. Moreover, this power capacity is tightly related with resource utilization, as higher usage corresponds to increased power consumption, with 100% percent utilization reflecting peak power consumption. Consequently, and since resource utilization is related to the type and volume of the workload, that is the type of useful work characteristics and the number of work units delivered, application power demand might not be in alignment with the power provisioning capability and characteristics of the infrastructure. This is a common case in modern systems as the energy-agnostic expansion of the previous years is now taking its toll, as power over-provisioning has become a tangible and pressing problem, especially in the data-center context.

Peak workload characteristics usually guide the selection of hardware resources as most modern software systems are designed to provide adequate quality of service during such time-frames. This suggests, that the energy provisioning capacity of the platform is in most cases determined by peak application demand [74]. In the case of web services for instance, servers are chosen in order to be able to handle user requests during peak load while exhibiting in parallel an acceptable response time.

Assessing how often an application or a software component operates on peak load is important for three reasons. First, if this level of load intensity occurs scarcely, then the respective component will be underutilized most of the time and thus operate in less efficient regions. As it is most evident in the web-service domain according to `Bohrer et al.` [10], servers will rarely operate at their maximum capacity, as a considerable amount of their operating time is spent on lower utilization levels. This is caused due to the *bursty nature and randomness* of the workload. Second, if peak workload requires some amount of power which is by far lower than what the hardware resource can actually provision, then a software component will be underutilized even at its highest load intensity and even more energy than the previous scenario will be wasted. Third, if those two limits are completely aligned during peak workload (which denotes that the respective component operates at 100% utilization), additional latency is imposed, which is harmful from an energy efficiency perspective, since it imposes additional energy consumption.

While, peak provisioning capability is an essential quality for guiding hardware resource choice and deployment, average power and in extent energy consumption is the one responsible for determining the larger portion of energy consumption [25]. Furthermore, since peak workload is typically much higher than the average case, software systems will spent the majority of their operating time in the low utilization spectrum which is less energy efficient.

This characteristic reveals implications with respect to the energy consumption of a hardware node, as it raises the challenge of configuring and deploying software systems in a way so that they exhibit sufficient provisioning capacity to serve peak application workload while being energy efficient in all other cases. Consequently, since software systems are not static, capacity can only be assessed by ascertaining the dynamic nature of the application workload and the impact that different levels of computing demands might have. Thus, in order to increase energy efficiency and eliminate energy waste it is crucial to evaluate how close the provisioning capability of the host platform is to the actual energy demands of an application both at peak and average workload.

By assessing the capacity sub-characteristic, evaluators are able to identify areas of the system where energy is over-provisioned and eliminate the waste, increasing energy efficiency levels in return. Furthermore, the properties that are used in order to assess

capacity can guide practitioners through consolidation process, and help them explore and set realistic values for the level that other QoS requirements should exhibit.

5.2.3 *Resource Utilization*

The Resource Utilization sub-characteristic is concerned with how *"well"* the host infrastructure is used by an application or a software component in order to deliver useful work. Therefore, answers will be given as to whether the amount of work output of the application justifies the amount of resources used. In this regard, we will investigate how close to the optimal system resources operate to deliver useful work and to do so, it is useful to assess to what extent the system under evaluation tackles known sources of inefficiencies.

The goal of assessing Resource Utilization is to understand:

- The impact that the average usage of a software component has on the energy consumption of its host hardware.
- The deviation between average and optimal energy consumption when a software component produces work.
- The impact that external systems have of on the energy consumption of an application or a software component.

Description

Assessing the resource utilization characteristic, boils down at evaluating each hardware and software resource individually but also holistically from a static and a dynamic perspective, by also nullifying the impact that external systems have on the application under evaluation.

In the context of a practical and universally applicable model, however, evaluating all those aspect is impossible since it would create a tremendous overhead. Therefore a higher level holistic evaluation is necessary. This is done by assessing resource utilization at an application level and under the prism of a useful unit of work. Consequently, each software component is evaluated with the aim of identifying whether it operates close to its respective optimal efficiency.

Of course such an assessment immediately creates two implications. First, there is neither a unified and meaningful unit of work that holds value for every system type or software component. Second, there is no universally applicable optimum to set as an efficiency goal for every system or component [35]. In order to minimize the impact that such issues have on the generality of the model, the choice of a work unit as well as the optimum efficiency level become relative to the specific system. This relativity, maintains the generalizability and comparability of the results therefore enabling comparisons between different system types and software components.

In order to further understand the challenges here, it is important to briefly describe what *energy proportionality* is. One of the prime reason that leads to in-efficient systems is that usually, more resources are used than the ones that are actually needed to produce a certain work output. Modern systems in their majority, do not scale their energy consumption in accordance to the intensity of the workload. Therefore, as is also pointed out in [90] the concept of energy-proportional computing [6] or energy scale-down [57] has become a pressing concern and is drawing increasing attention in the Green IT field.

The term energy proportionality was first coined by Barosso and Holtze [6]. Ideally, systems should consume zero energy when they idle, and proportionally more energy as workload increases. The inverse also applies, as a decrease in workload intensity, should signal in turn, a proportional decrease in energy consumption and thus, consumption and workload intensity should scale proportionally.

From the above, a first aspect of the resource utilization characteristic, is the ability of software component to scale energy consumption with respect to the variability of the workload. Arguably, energy proportional systems would be equally power efficient regardless of workload level [25]. However, as can be seen in Figure 1 this is far apart from the actual case, and modern systems consume a significant portion of their energy when they idle or at very low utilization. This is because the power that is drawn from the host resources at those utilization levels or when they idle is a significant portion compared to their peak power[2].

Another important aspect has to do with how close to its optimal consumption each software component operates during production. Since some amount of energy is required by every software component to produce a work unit, there will be some minimal best case energy consumption for each component. Therefore, an efficient component – one that uses his host resource optimally – should consistently require the same amount of "optimal" energy to produce a work unit.

A final important aspect is the impact that external software systems have on the energy consumption of the components or the application that is under evaluation. If resources are used optimally then this interference should be minimal. However, applications rarely exist in isolation and the most common case is that a system will host a plethora of other applications – apart from the one that is the model input – all of which compete for shared resources.

Although resource contention is tightly related to performance, since it causes degradation, it also heavily affects energy efficiency as any additional latency in this domain is translated to palpable energy consumption. Consider, an application thread competing with threads from external systems. If one of those – external to the application – threads manages to lock the resource for which they are competing the application thread will have to wait until the respective resource is released. This additional latency implies that resources remain powered up for more time than they normally would, consuming additionally energy in extent. Therefore, although if measured in isolation (that is without other external systems using the infrastructure) an application or a component might get sufficient results regarding its energy efficiency levels, when put into a shared environment, the same application might be considered highly inefficient due to external system impact. Consequently, the impact that other co-allocated applications have on the energy consumption of an application under evaluation must also be explored in order to come to a tentative evaluation regarding its energy efficiency levels .

By assessing the resource utilization sub-characteristic, evaluators are able to identify areas of the system that are not used optimally, a fact that will enable them to take the necessary actions in order to eliminate energy waste. Furthermore, the properties that are used in order to assess resource utilization can guide administrative choices in order to reduce the impact that other co-allocated application might have on the energy footprint of an application under evaluation.

2 Barosso et al. for instance observed that the idle power that is drawn by a typical server deployed at Google Data center is never below 50% of its peak. More information can be found in Section 3.9.

6 PRACTICAL MODEL

This section presents the practical model that is proposed in order to evaluate the energy efficiency of software systems. This is done by assessing how efficiently hardware resources use energy when doing work on behalf of the software components of the system under evaluation.

The first consideration to be made in such a practical model is the scope of the evaluation. All components of the application under evaluation are, of course, in scope. When the application under evaluation relies on external applications/services, a decision has to be made on whether these external elements are in scope or not. Such decision precedes the evaluation and the choice its a joint responsibility of the system stakeholder and the evaluator.

Having set the scope for the evaluation, it is necessary to identify the main functionality offered by the system. Without it, it is not possible to define a work unit and consequently it is not possible to evaluate energy efficiency.

6.1 THE REQUIREMENTS

Based on the limitations that where discussed in the problem section (Section 2.1) and in alignment with any findings during the course of the literature survey, the model presented in this thesis should exhibit certain desired characteristics. Most of them are derived by the following sources [74][76][4][37][36], and others are contribution of this work. We believe that those characteristics are crucial for a model that will be able to make high-level efficiency evaluation regardless of system context.

PRACTICAL: The model must be practical and as such, it should be small and as simple as possible to use. Thus, the number of inputs required and the complexity of using the model and deriving results should be minimal.

HOLISTIC: The model should focus on the application software as the target for future optimizations. However, the model should not ignore relevant aspects regarding hardware and should evaluate the complete system stack (Hardware and Software).

ACTIONABLE: The model should be sufficiently accurate, in order to identify optimization opportunities. Thus, it should produce results that are actionable, in as sense that they should point to clear optimization opportunities. The goal is not only to assess systems, but also to identify the optimization paths that will have the biggest impact in reducing the energy waste.

COMPARABLE RESULTS: The model and its metrics should be applicable and meaningful to different types of system in a way that results are still comparable. This in extent will mean that the results will also be benchmarkable.

UNDERSTANDABLE: The model should be easily understandable and should produce results which are easy to interpret and communicate.

INTERNAL VIEW: The model should be impartial regarding external requirements. This essentially means that all systems are treated as equal by the model, regardless application type and any external requirements (`Quality of Service (QoS)`, `Service Level Management (SLM)`, `Workload`, `Performance Requirements`). The latter are used to more accurately contextualize evaluation results, in order to facilitate communication and filter out optimization opportunities when the model results are interpreted and communicated to the system stakeholders.

6.2 MODEL

This section deploys the `GQM` approach based on the conceptual division of Chapter 5. The main goal of the research is broken down accordingly and as such the goal of evaluating the energy-efficiency of a software system is divided into the sub-goals of assessing each sub-characteristic of the quality as can be see in Figure 3.

For each sub-characteristic of energy efficiency (`Energy Behavior (Figure 5)`, `Capacity (Figure 7)`, `Resource Utilization (Figure 8)`), this section applies the `GQM` approach described in Section 4.2, stating the goal and raising questions that must be answered and metrics that shape a practical model to answer these questions.

All the questions are raised from the perspective of an evaluator that wants to assess the energy efficiency of a software system, with the purpose of identifying energy inefficiencies and feasible optimization opportunities. The practical model that is proposed in this section as well as the metrics that are used by it provide the answer to `Research sub question 2:` *How can these aspects be quantified in a meaningful way?* (Section 2.4)

6.2.1 *Energy Behavior*

Goal Statement

Based on the main goal of the model as expressed in Section 2.1 the following sub-goal is set:

- SUB-GOAL 1: Evaluate the Energy Behavior of a software system (Figure 5).

Questions

From the elaboration presented in Section 5.2.1 and the sub-goal that was set above the following questions arise:

- Q1: What is the energy consumption of each software component?

  Answering this question will reveal the biggest energy consumers from a software perspective. This will allow practitioners to abstract from hardware resource consumption and relate energy to software components enabling a better understanding as to how energy is attributed to application elements, allowing them in return to focus on those software parts where optimization will potentially have the greatest impact.

- Q2: How much energy is expended by the system when no useful work is delivered?

Answering this question will reveal the impact the the idle consumption has on the overall energy consumption of the system. Ideally, an idle system should consume zero energy, but this is never the case and modern systems consume a significant amount of energy when they idle. Thus, by answering this question evaluators can gain additional insight as to where to target optimization efforts in order for them to have bigger impact (Active or Idle consumption).

- Q3: How much energy is consumed by each software component per useful unit of work?

  By answering this question, evaluators will be able to form a perception about the energy consumption of software components in accordance to the work each one begets and form an opinion as to whether this cost is reasonable. This in turn, will permit them to focus on components which consume disproportionally more energy than the work that they produce.

Metrics

In order to give an answer to the previous questions, the following metrics are used. The metrics are presented in order with the respective question that they are called to answer.

- [ACC] Annual Component Consumption: The energy consumption per software component on an annual basis measured in KWh.
- [RIC] Relative Idle Consumption: The percentage of the annual idle application component consumption to the annual total application component consumption.
- [CCUW] Component Consumption per Unit of Work: The average energy consumption of every software component per unit of work delivered.

6.2.1.1 *Metric Definition*

[ACC] Annual Component Consumption

Software components use hardware resources to deliver useful work and therefore it is feasible and viable to attribute portions of the hardware energy consumption, to residing software components. Since co-allocated components are the most common case in modern systems, a methodology must be used in order to accurately attribute energy consumption to each component in a shared environment.

In order to achieve this, `CPU` usage ratio for each software component is used. That is the `CPU` utilization caused by the software component's process and sub-processes. For a certain software component, `CPU` usage can be calculated by dividing the number of `CPU` cycles allocated on processing component requests, to the number of total `CPU` cycles for a certain time frame. The same methodology is used for co-allocated software components. OS-reported `CPU` utilization has been often used as a first-order proxy for energy attribution and estimation [25][48][47][23][68].

The total consumption for a software component can be calculated by multiplying the `CPU` utilization caused by the respective component process and all of its sub-processes, to the power drawn by the host hardware resource at those utilization levels. The more frequent the utilization samples, the more accurate the results. Consequently, the total

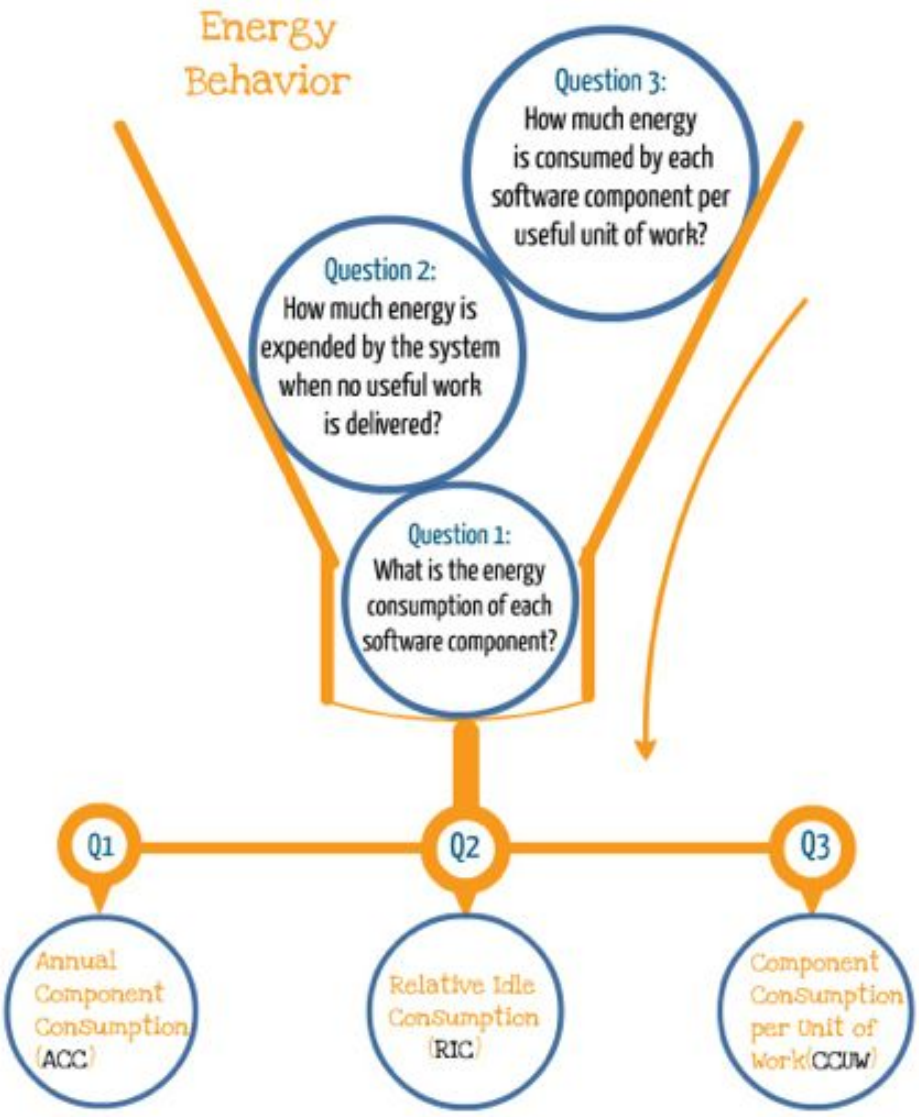


Figure 5: Goal-Question-Metric approach applied to the Energy Behavior sub-characteristic

consumption caused by a software component can be calculated as the sum of component utilizations to the power drawn at those utilization levels by the hardware resource throughout the period.

$$\mathrm{Util}_{\text{soft.component}} = \mathrm{Util}_{\text{parent process}} + \mathrm{Util}_{\text{(sub-processes)}} \tag{5}$$

$$\mathrm{TC}_{\text{component}} = \sum_{i=1}^{n} \mathrm{Util}_{\text{soft.component.i}} \times \mathrm{Power.drawn}_i \tag{6}$$

Equation 5 presents a way of calculating the total CPU utilization caused by a software component. Equation 6 presents a way of calculating the total energy consumption (TC) caused by a software component where n is the number of samples in seconds(for a 7 day period n=604800), Util is the CPU utilization caused by the component during that second, and Power.drawn is the power consumption at that point in time.

If power measurements are not available, then one can use a model that relates utilization to power consumption to estimate it. When doing so, the total utilization should be

matched to estimates of power consumption. Subsequently, this power consumption can be attributed to the components in proportion to their respective utilization.

When calculating the energy consumption of a software component it is crucial to attribute idle consumption accurately. Therefore, it is also important to distinguish between `System Idle Consumption` and `Application Idle Consumption` [37]

- System Baseline Consumption: This is the consumption of the system when hardware resources idle.
- Application Idle Consumption: The consumption of the system when the application does not perform work.

From the above its is obvious that the idle consumption of a hardware resource is different than the idle consumption of a software component hosted by this resource, since the the component might be idle (Application Idle) and the resource might be active, executing instructions caused by external software components. The above methodology attributes the correct portion of idle consumption (Application Idle) to the total energy consumption of software components as it uses process utilization statistics and not total (machine) utilization statistics. Therefore if the process idles but the host resource is not, since the process will have 0% utilization at that point, the power attributed to that process will be the application idle consumption. This methodology is used for all hardware resources that host some application component resulting in a `TC` value for all the software components that belong to the application under evaluation.

Measuring the annual component consumption might not be a feasible goal since it requires continuous measurement for the entire period. Thus, in alignment with the methodology that `SIG` already applies [4], a small time-frame (a month or one week) is used for measurement. Once, the measurements for this period are obtained they are then aggregated on a yearly scale (multiplying `TC` values by 52 in the case of weekly, or 12 for monthly measurements). Consequently, the value derived by this metric is not the actual yearly consumption of the software components, but a relative to the measurement period value. Thus, the bigger the time frame used to obtain the annual consumption the more accurate the result, since the actual energy consumption might fluctuate for certain periods of a given year.

[RIC] Relative Idle Consumption

Energy is power over time and therefore a component's idle consumption, depends entirely on two factors. The idle power values of the machine hosting the component and the time that this component spends in the idle state. Ideally, one would want one of those two variables to be zero, in order to achieve zero idle consumption. However, this is not the case in the vast majority of modern systems. The possibility of a machine that would draw zero power at zero workload is an unrealistic prospect according to [66].

Since idle consumption is energy which is by definition wasted [81], it is necessary to identify its impact on the overall energy footprint of the system. In order to calculate the Relative Idle Consumption the following methodology is used.

- First the `Idle Power Values(IPV)` of every hardware node that belongs to the system in scope are obtained.
- Afterwards, the time that all application components spent idling is calculated. That is, the amount of time that an application component idled on the respective hardware resource.

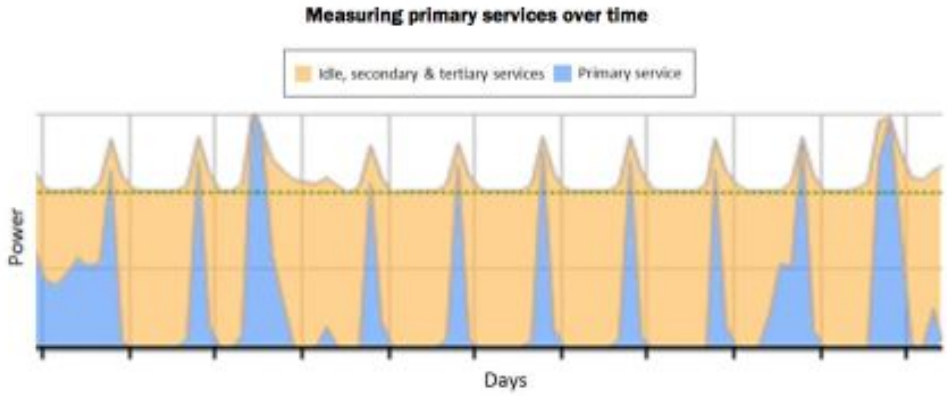


Figure 6: Typical Idle Energy Consumption Pattern of a Web Service. Figure from [34]

In order to calculate this time, apart from counting the actual idle time of the host hardware node, periods in which a software component is idle but the specific system resource is not are also taken under account. This is done in order to distinguish between system and application idle time (as discussed in Section 6.2.1.1) and derive a value for this indicator that reflects the impact that the component idle consumption has on its total energy consumption.

Software component and hardware resource idle times are expected to be different, since in many cases in which an application component idles, its hardware resource might be active, executing external workloads.

Consequently, when the `CPU` or `GPU` of a specific hardware node, is not executing instructions that originate from an application component, but is used by external software components, this time is also counted as `Component Idle Time(CIT)`, for the purpose of calculating the `Annual Idle Consumption(AIC)` for the specific application component. Therefore the time a component spent in the idle state (`CIT`) for a specific hardware node for the purpose of the model is counted as:

$$\mathrm{CIT}_{(i)} = \mathrm{Resource}_{\mathrm{idletime(i)}} + \mathrm{Resource}_{\mathrm{time\ on\ external\ workloads(i)}} \tag{7}$$

This is done for every application component. Next, in order to calculate the Annual Idle Consumption, the component idle time (`CIT`) is multiplied with the idle power values (`IPV`), which are the baseline idle values, of the respective hardware resource. This produces the `Annual Idle Consumption (AIC(i))` of every software component that belongs to the application. The following equation is used in order to calculate this value for a component *i*:

$$\mathrm{AIC}_i = \mathrm{IPV.host.resource}_i \times \mathrm{CIT}_i \tag{8}$$

Since measurements are done on a process level (each software component corresponds to a process), the idle time of the component can be simply calculated by the *"idle usage"* of the corresponding process based on the measurement methodology that is followed. By multiplying this time with the idle power values of the host resource `AIC` can be obtained.

As it is evident, the above indicator is not comparable between different component types, as it suffers from the same limitations as the `ACC` metric. Consequently, in order to derive a value that will be comparable among different software components and system classes it is crucial that this value is made workload and system type independent. In order to factor out the above the Annual Idle Consumption (`AIC`) is divided by the Annual

Component Consumption (ACC). This produces the Relative Idle Consumption(RIC) for a specific software component.

$$RIC = \frac{AIC}{ACC} \tag{9}$$

[CCUW] Component Consumption per Unit of Work

This metric, aims at relating work production to energy consumption, by assessing the energy consumption of application components with respect to the units of work they deliver when executing application workload.

A universally applicable unit of work is utopical since different software systems provide a wide variety of distinct functionalities. For a web service for instance, the number of transactions can be considered as a useful unit of work. These transactions in turn can be the number of processed orders, executed payments, update dossiers, or permits issued [4]. For a system that handles batch workloads, the unit of work may be a batch whereas a search will be the most suitable unit of work for a search engine. Consequently, it is up to the stakeholders of the system, to define a main responsibility and a useful unit of work for the purposes of this model. All other system functions, apart from the one set as the useful unit of work, are considered auxiliary to the main unit and therefore their energy consumption is attributed to it.

In order to calculate CCUW the ACC value of each software component is divided by the number of units of work that the respective component delivered during the measurement period scaled annually. Therefore for a software component i, CCUW can be calculated using the following equation:

$$CCUW_i = \frac{ACC_i}{No.UW_i} \tag{10}$$

Since workload is factored out, this metric is not destined to discover the largest consumers among a set of software components, but can be used to identify those that exhibit a higher amount of consumption than would be expected with respect to their functional complexity [4]. Therefore, software components and in extent software systems (after the proper aggregation) that have a relatively similar functional complexity, can be compared regardless of their workload.

CCUW is expressed in Joules per unit of work. This is done to differentiate from other *"performance oriented"* metrics that where identified during the literature survey (MHz/watt, Bandwidth/watt etc.) [80]. Using a performance based metric to express energy consumption requires the selection of a fixed performance level before completing a series of steps in order to derive and normalize the reciprocal. Furthermore, such metrics obscure the fact that small improvements can produce substantial gains regarding energy consumption.

On the other hand, consumption based metrics such as CCUW, which are expressed in joules/unit of work, use a standard performance level (unit of work) and therefore present the amount of energy that is required in order to produce this output. Consequently, they can be used to more accurately form a perception about the relation of energy consumption to generated work production [52], and thus seem to be more suitable for decision making and planning as they provide more actionable results.

6.2.2 *Capacity*

Goal Statement

Based on the main goal of the model as expressed in Section 2.1 the following sub-goal is set:

- SUB-GOAL 2: Evaluate how suitable the energy provisioning capability of the host platform is for the average and the maximum application energy demand (Figure 7).

Questions

From the elaboration presented in Section 5.2.2 and the sub-goal that was set above the following questions arise:

- Q4: What is the minimum energy provisioning capacity that the platform should exhibit in order not to affect application behavior?

  The answer to this question will reveal *"safe"* growth or consolidation opportunities. These are optimization opportunities that will not affect other desired qualities and service requirements that the system should exhibit. This is done by assessing the energy requirements of the application when executing peak level workloads. A provisioning gap at that level will point to safe consolidation opportunities, whereas in the case where provisioning capabilities and application requirements meet, the most probable optimization strategy would be to either increase the provisioning capability of the hardware node or re-allocate and reschedule application workload if possible in order to eliminate the probability that bottlenecks occur.

- Q5: To what extent are the energy provisioning capabilities of the infrastructure utilized and stressed when executing application workloads and what is the impact of workload variability on the energy budget?

  The answer to this question will enable practitioners to identify the extent to which workload fluctuations affect resource usage in order to discover resources that are consistently under or over utilized. In the former case, un-proportional behaviors might be revealed, as under-utilized resources in most cases consume disproportionaly more energy than the work that they deliver. In the latter case, although high levels of utilization might correspond to high level of efficiency, this is only accurate when resources are considered in isolation. On a shared platform environment however, an overly utilized resource might create bottle necks for the whole system, impeding the overall energy consumption of the application. This knowledge will enable practitioners to act accordingly by following the suitable optimization strategy to eliminate waste.

Metrics

In order to give an answer to the previous questions, the following metrics are used. The metrics are presented in order with the respective question that they are called to answer.

- [PGR] Peak Growth: A percentage that denotes the difference between the theoretical provisioning capability of hardware resources and actual application energy demand when serving peak workload for every application component.
- [PRO] Provisioning: A percentage that denotes the relation between theoretical provisioning platform capabilities and daily energy capacity used in order to handle application workloads for every application component.

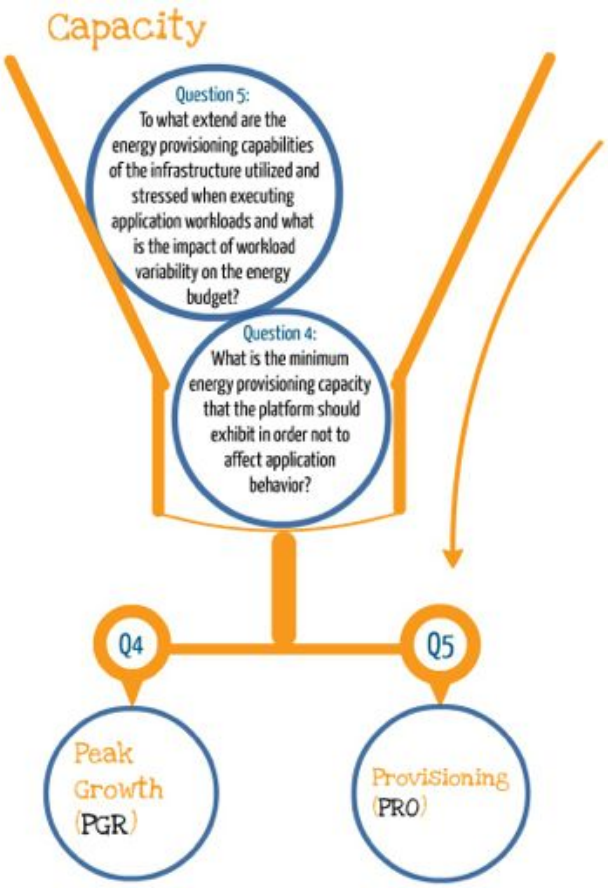


Figure 7: Goal-Question-Metric approach applied to the Capacity sub-characteristic

6.2.2.1 *Metric Definition*

[PGR] Peak Growth

As it was stated in Section 5.2.2 it is important to assess the capacity sub-characteristic both in peak and average workload cases. This metric aims at evaluating the former by identifying the deviation level between the actual theoretical provisioning limits of the host infrastructure, to the maximum power drawn by the application components during peak workload. Since power drawn is related to utilization and utilization is ultimately determined by the workload, the maximum power drawn is expected to occur during a moment of peak workload. Obviously, since peak workload denotes an upper

limit on the component energy demand, assessing the gap between the former and the provisioning capability of the platform will reveal *"safe"* optimization opportunities.

Of course as it is already mentioned, applications will not always reside on the same hardware node and therefore will utilize different parts of the infrastructure in a divergent way. Consequently, based on the structure of the work unit and the intensity of the workload, different hardware resources will present different levels of deviation between peak power demand and provisioning capability according to the respective application components that they host. As it is also stressed by `Weske et al.` [97], the energy consumption of an application service is dependent to the nature of the business process. Therefore, the resources used and their respective energy consumption will vary based upon the type of the application(computationally or data intensive or hybrid in nature) [49].

In order to calculate `Peak Growth(PGR)` the following equation is used for every application component:

$$\mathrm{PeakGrowth}_{\mathrm{app.comp}} = \frac{|\mathrm{Power}_{\mathrm{peak\ load}} - \mathrm{Power}_{\mathrm{theoretical}}|}{\mathrm{Power}_{\mathrm{theoretical}}} \times 100\% \tag{11}$$

The $\mathrm{Power}_{\mathrm{peak\ load}}$ is the maximum power drawn by an application component on daily basis. This can be measured at a moment of peak load where sofware component reaches its maximum utilization whereas, $\mathrm{Power}_{\mathrm{theoretical}}$ is the maximum power that the machine can potentially provision at its maximum utilization. This value can be obtained by actual measurements during load testing or by using the nameplate value of the respective hardware resource, although many researchers suggest that those values cannot act as a solid reference point for determining peak power consumption as deviations from 30% to 60% have been observed between actual peak consumption and nameplate values [62][25][72].

For co-allocated software components the $\mathrm{Power}_{\mathrm{peak\ load}}$ is calculated for all components combined. This is done because, if `PGR` was calculated based on each component utilization, in the case of co-allocated components, it would produce results that would hold no value, since it would present very wide power gaps individually for each component, whereas the gap at an application level would be much smaller. Therefore the $\mathrm{Power}_{\mathrm{peak\ load}}$ is calculated at an application level (co-allocated components combined) and then `PGR` is scaled on a component level based on their respective contribution $\mathrm{Power}_{\mathrm{peak\ load}}$ value.

[PRO] Provisioning

While the previous metric (Section 6.2.2.1) was focused on assessing peak demand and supply, this metric aims at identifying how the average workload of an application exercises the provisioning capabilities of the host platform.

While two hardware resources or systems might dissipate the same amount of energy, when serving identical load for a fixed period, their peak power consumption might be entirely different, as one of them might exhibit a bursty behavior and therefore have a small `PGR` value, whereas the other might operate on relatively steady and lower utilization spectrum consistently when serving workload and therefore present a relatively large `PGR` value.

If judgments are made based solely on peak load and consumption, then the first system will not be able to facilitate any additional future peak load, while any consolidation effort might potentially hinder other `QoS` aspects for the moments that the load

is more intense. On the other hand the second system, will be more susceptible to either growth or consolidation efforts as it might easily reduce its theoretical provisioning limit, by consolidating parts of the system for instance, or by serving a higher amount of load within the current platform configuration. Consequently, although the preceding systems might expend the same amount of energy in total, they do so in a highly divergent way as their respective trends in consumption are shaped by the variability and characteristics of the workload and thus both systems can save energy and improve their efficiency by following different optimization strategies in order to provide just as much power as needed at the right time frames in order to deliver useful work.

To calculate PRO the energy consumption of each application component is divided by the total theoretical energy that the host resource could have provided for the measurement period. This results in a percentage for each software component that informs evaluators about the difference between actual and theoretical energy usage for that period. When calculating the energy consumption for a component its idle consumption is not taken under account. This is because this portion of energy falls more under the domain of unused energy capacity as it could have been potentially used to serve workload. Therefore when calculating the daily energy consumption only the Active energy of every software component is taken under account.

$$\mathrm{PRO}_{\text{soft.comp}} = \frac{Active.Energy_{\text{soft.component}}}{Max.Energy_{\text{host.resource}}} \tag{12}$$

The $Active.Energy_{\text{component}}$ for every software component can be calculated by subtracting its AIC (Equation 6.2.1.1) value to its ACC (Section 6.2.1.1) value.

The $Max.Energy_{\text{host.resource}}$ can be calculated by multiplying the power that the hardware draws at peak utilization by the measurement period in seconds. For co-allocated components the same methodology as the one described in the PGR (Section 6.2.2.1) is used. Therefore the Active Energy of co-allocated components is summed and after PRO is calculated, the percentage is scaled based on each component contribution to the $Active.Energy_{\text{component}}$ value.

6.2.3 *Resource Utilization*

Goal Statement

Based main goal of the model as expressed in Section 2.1 the following sub-goal is set:

- SUB-GOAL 3: Evaluate the relationship between the amounts and types of hardware resources used by an application, when serving workload, to the corresponding energy expenditure and workload intensity (Figure 8).

Questions

From the elaboration presented in Section 5.2.3 and the sub-goal that was set above the following questions arise:

- Q6: How proportionally does the energy consumption of a system scales with respect to the variability and intensity of the workload?

  The answer to this question will reveal one of the most notable reasons that cause energy waste: Systems that do not reduce energy consumption when workload is relatively low. Usually at lower utilization levels, the energy that is expended by

the system to produce a certain amount of work units is disproportionaly more compared to the energy cost of each unit when the same system operates at higher utilization levels since the throughput is higher.

- Q7: How elastic is the energy consumption compared to the theoretical best case proportional behavior with respect to distinct system utilization levels?

  The answer to this question will enable practitioners to more accurately quantify the impact that a disproportional system behavior might have on energy consumption. By examining the elasticity of the system at different utilization levels, through assessing how far the power provided by the infrastructure deviates from the ideal case at distinct utilization levels, evaluators can determine the impact that the average usage of a software component and in extent the hardware that hosts it has on the energy consumption. Thus, by increasing or decreasing the average resource utilization, through shifting or rescheduling workload to levels that present a smaller gap between ideal and actual case, evaluators will have the chance to minimize the impact that disporportionality has on the total energy cost.

- Q8: How much of the total energy consumption of the system is used by application components to deliver useful work?

  The answer to this question will reveal the energy overhead that is required to deliver useful work. Since the application under evaluation is considered the only source that generates value for the purpose of this model, it should account for all, or the most, energy that the infrastructure provisions. Therefore any difference between the application energy consumption and the total energy consumption of the host platform can be considered as an overhead, since it is not used by the application and therefore it does not directly produce work for the respective organization.

- Q9: What is the impact of external systems on the energy consumption of the application?

  The answer to this question will denote the impact that other software systems might have on the energy consumption of the software components that constitute an application. If system resources are not dedicated to the application, or the systems that are hosted by the infrastructure are excessive in number, then the probability that additional energy (compared to an isolated operation) would be required by the application components to deliver useful work will be higher due to resource contention events.

Metrics

In order to give an answer to the previous questions, the following metrics are used. The metrics are presented in order with the respective question that they are called to answer.

- [CNS] Consumption Near Sweet-Spot [4]: A percentage for every software component that depicts its efficiency with respect to its optimal efficiency when delivering work units.

- [PG] Proportionality Gap [100]: Deviation between the ideal power provisioning case and the actual one at distinct utilization levels, for every application component.
- [OPO] Operational Overhead: A percentage that denotes the energy overhead required by the infrastructure to process software component workloads.
- [ISO] Isolation: A percentage that denotes the impact that external applications have on the application energy footprint.

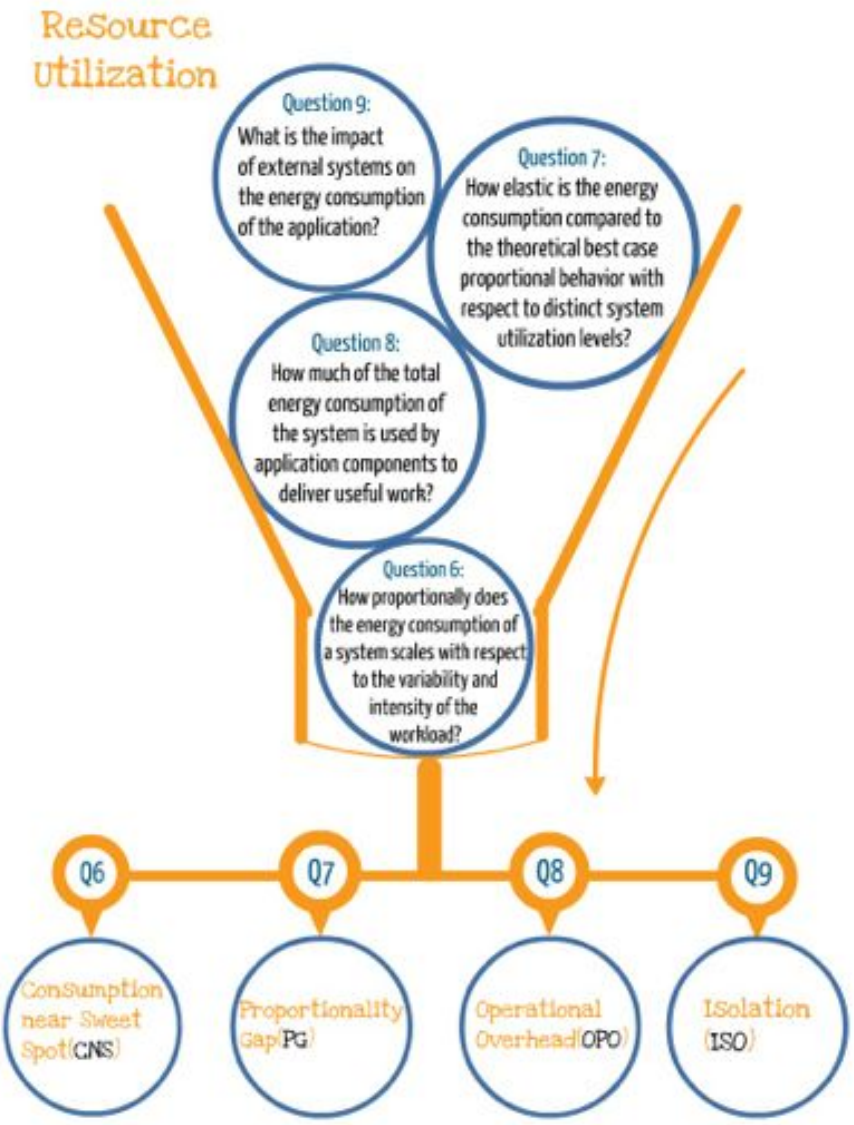


Figure 8: Goal-Question-Metric approach applied to the Resource Utilization sub-characteristic

6.2.3.1 *Metric Definition*

[CNS] Consumption Near Sweet-Spot

The `Consumption Near Sweet-Spot(CNS)` metric is developed by `SIG` [4] and aims at assessing the energy-efficiency of software systems, by evaluating how proportional their energy consumption is with respect to the variability of the workload.

The rationale behind this metric as is also described in the original paper [35] is the following. The efficiency of a system can be expressed by the amount of energy it requires to deliver a unit of work and will vary based on the intensity and variability of the workload at any given moment. Consequently, there will be some optimal level of energy consumption at a certain load level – *the sweet spot*. This consumption might deviate from the actual theoretical optimal but it can still act as a relative reference point to base optimizations on. Thus, a system can be considered energy-efficient if it operates consistently close to the sweet spot. The same indicator – as well as the rationale behind it – can be applied to individual software components as an efficient component should be constantly performing close or on his respective sweet-spot.

At any given moment `CNS` can be calculated as the ratio between a software component's average energy consumption per unit of work to its optimal one. This can be seen in the following equation:

$$CNS = \frac{CCUW_{opt}}{CCUW_{avg}} \tag{13}$$

Where $CCUW_{avg}$ is given by Equation 10, and expresses the average component consumption per unit of work during the measurement period and $CCUW_{opt}$ can be obtained simply by identifying the minimum observed energy consumption per work unit throughout the same period for the respective component. This value can typically be observed at a peak workload moment, during which, the resource will most probably run at its highest utilization resulting in high levels of efficiency. Since high intensity workloads might not occur in a production environment, an optional (and optimal case) is to obtain this value during system load/stress testing. By reaching the highest level of utilization this optimal efficiency value can be derived.

For the case study presented in this thesis a slightly different approach was followed in order to get a more accurate picture of how close to their optimal case the components operate. First, in order to derive $CCUW_{opt}$ for every component, the energy consumption of every unit of work that was delivered per software component was calculated for the entire measurement period and the respective minimum value was chosen per component. Second, `CNS` was not calculated based on $CCUW_{avg}$, but was based upon the respective energy consumption of every unit of work delivered per software component. In this case the `CNS` values will be as many as the units of work delivered by a software component and can be used to populate a quality profile per component.

Since the `CCUW` values that are used are on a per transaction basis (average consumption per transaction on the first case and the actual consumption per transaction if `CNS` is calculated for every unit of work), workload is factored out. Furthermore, by dividing observed consumption values to their relative optimal cases, system and component types are factored out, therefore `CNS` can be used to make direct comparisons between different system and component types.

Components that operate on highly fluctuating workloads or idle for large periods, will attain a good `CNS` value only in cases where they exhibit a very proportional behavior. In the opposite case, evaluators should direct their efforts, towards reducing the idle component consumption, or the energy consumption at low utilization levels. The

choice between increasing utilization or reducing the idle consumption, can be made based on the RIC metric (Equation 6.2.1.1). A high value in that indicator means that optimizations should be focused at reducing idle consumption, whereas in the opposite case optimizations should be targeted at increasing energy efficiency inside the low utilization spectrum. Viable optimization choices are workload migration or replacing parts of the infrastructure with resources that have low-power-active states capability for instance.

[PG] Proportionality Gap

The Proportionality Gap(PG) [100] is a metric that expresses the difference between the ideal energy consumption and the actual one, at distinct utilization levels for a component. Therefore PG combined with CNS can enable evaluators to more accurately distinguish the factors that cause disproportional behaviors by enabling them to quantify their impact on the overall energy consumption. PG is calculated using the following equation for a specific utilization level x:

$$PG_{x\%} = \frac{Power_{actual@x\%} - Power_{ideal@x\%}}{Power_{peak}} \tag{14}$$

In Equation 14, $Power_{actual@x\%}$ denotes the actual power that is drawn by a hardware resource when a software component operates at that utilization. $Power_{ideal@x\%}$ denotes the power that the host hardware resource should ideally draw at that utilization. $Power_{peak}$ denotes the maximum power that the host resource can provision. The values derived from this metric can be interpreted using the instructions from the original paper:

"For an ideal energy proportional server, the PG for all utilization levels is 0. For super-linearly proportional systems, like Server A, PG is very large at zero utilization and it continues to grow for some time before it starts to shrink. For sub-linearly proportional systems, like Server B, PG is very large at zero utilization but it continues to decrease with utilization."

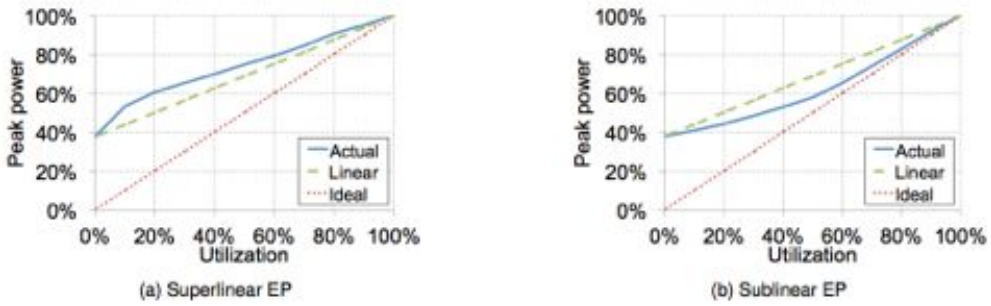


Figure 9: Energy Proportionality (EP) curve. The dotted, dashed, and solid lines shows the ideal, linear, and actual server EP curve, respec- tively. [100]

PG can also be used to identify disproportional behavior in the software component domain. In order to calculate PG the following methodology is used for every application

component. First, a linear model regarding the ideal energy consumption is created for every hardware resource that hosts some application component. Then the actual power consumption for all the utilization levels is measured for each hardware resource. The difference between the Ideal and the actual case can be seen in Figure 9. Finally, for each software component PG is calculated by using Equation 14, the average utilization of the component for the measurement period, and the power values of the host resource for the respective component utilization. For co-allocated software components the same correction methodology as the one mentioned in preceding metrics is used.

[OPO] Operational Overhead

When evaluating the energy efficiency of an application under a functional perspective it is important to assess the efficiency with which units of work are produced, as this production is the source of generated value for the application stakeholders. Therefore it is crucial to identify any energy overhead in order to eliminate or limit its impact.

Since the application is considered the only source that produces useful work for purpose of this model, any additional consumption apart from the one attributed to the application components could be considered a waste. Thus, it would be desirable that all the energy that was dissipated from the hardware was clearly used by the application to produce work. However this is rarely the case, as hardware resources usually host a wide spectrum of applications and thus, they should also provision energy to them. Therefore a portion of the total energy dissipation is used to handle workload caused by external applications (Figure 10). Consequently, it would be desirable that this portion is minimized.

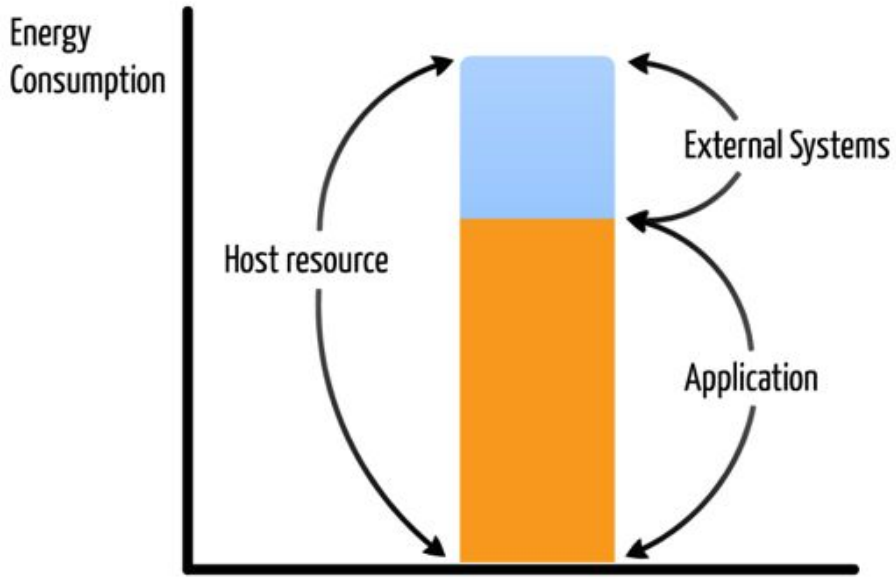


Figure 10: Total resource energy consumption in relation to application energy consumption

In order to calculate the Operational Overhead, the following methodology is used. First the total energy consumption for every hardware resource of the infrastructure is

calculated. To do so, a similar methodology as the one used for the calculation of the ACC is used (Section 6.2.1.1). Therefore for the entire measurement period the CPU utilization values of the resource are multiplied with the corresponding power consumption. Doing so for the whole period, produces the Total Energy Consumption(TEC) of every host resource.

Then, the calculated energy consumption of an application component ACC is divided by the total energy consumption of its respective host resources. This is done for all application components. The following equation can be used to calculate OPO for an application component x hosted by a resource i.

$$OPO_x = \frac{ACC_x}{TEC_i} \tag{15}$$

For co-allocated application components the same methodology as described in preceding metrics is used to correct the results.

[ISO] Isolation

Chen et al. [18] pointed out that when processors attempt to access shared resources simultaneously, the ones that cannot bind them due to contention will cause increased energy consumption and performance degradation. At a lower level, contentions occur when a function in some thread cannot access a resource and is forced to wait because another function in some other thread has already locked it.

In those cases the application that waits for the resource is considered active, but does not produce any useful work. Those *waits*, are caused by competing processes, thus hosting many application in the same platform configuration can be considered harmful from a performance and energy-efficiency perspective, if resource allocation techniques and appropriate scheduling mechanism are not carefully planned in a way that will minimize resource contention events in a shared environment.

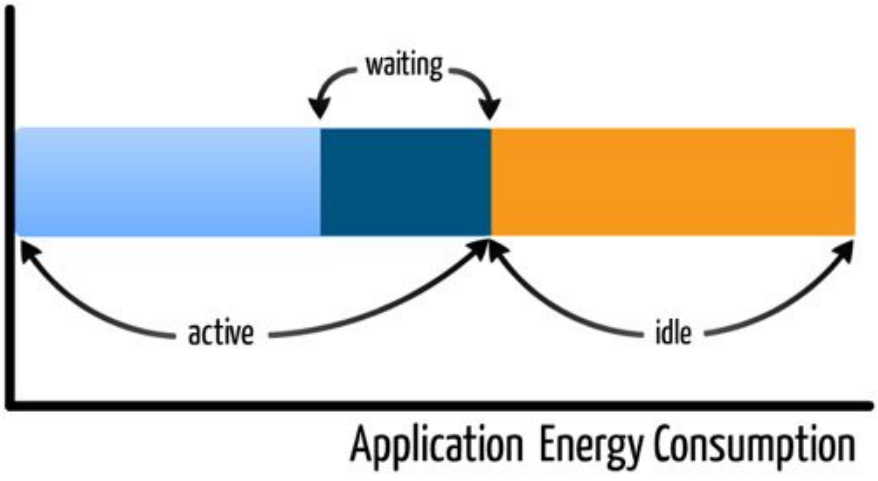


Figure 11: Division of the energy consumption of an application

Figure 11, presents the energy consumption of a hypothetical application. At a first level this consumption is divided into active and idle energy consumption. From these

two portions, the active one can be further broken down to actual active (used to produce work) and active wait(waiting due to external systems). Isolation, therefore tries to assess the extent of this active wait portion in order to determine the impact that external systems have on the overall energy footprint of the application under assessment. Of course, isolation can be evaluated at an application component level, by relating resource contention events to the respective software components and thus isolation can be expressed as: *"The extent at which the energy footprint of the application component is affected by external systems"*. Any results can be aggregated back to an application level.

Preventing and resolving resource contention incidents is one of the main operating system functions and various low-level mechanisms can be used, including locks, semaphores, mutexes and queues to insure that contention events are minimized. A second partial solution to contention problems is to introduce concurrency. Processing event streams on different threads or dynamically creating threads to serve incoming events, might yield significant energy related benefits as any blocked time can be potentially reduced if the proper load balancing mechanisms are used.

Virtualization can also be used to address contention issues as one of the key advantages of virtualized environments is improved fault and performance isolation between applications sharing the same hardware resources. Since a Virtual Machine is perceived as a dedicated physical server by the application running on it, aggressive consolidation and workload migration can be used in virtualized environments to increase energy-efficiency and enhance the isolation of co-allocated applications [9]. Nevertheless, virtualization is not a panacea, as some virtual machines might still not be able to allocate the required resources, as excesses of one `VM` can violate the isolation property of others causing a *"noisy neighbors effect"* [48]. This inevitably leads to performance degradation and increased latency [9].

Unfortunately, in the context of this research we could not identify a feasible and universally applicable way of measuring the isolation property, therefore no measurement details can be given. We envision a low over-head measurement that will be able to assess the isolation property at a software component level.

6.3 AGGREGATION

This section presents the rationale that is used to classify metric results based on their values and the aggregation methodology that is followed to map metric results from a software component to a system level. In the model there are two types of metrics. `ACC`, that has a simple aggregation to the system level and metrics that have quality profiles. `ACC` is aggregated at a system level by summing all the component values. All other metrics are aggregated by following the steps of the methodology that is described next. This methodology is depicted in Figure 12.

Step 1 and 2

First, all the metrics are calculated for the software components that constitute the application that is under evaluation. These metrics are calculated on daily basis for every component during the entire measurement period. The second step is to classify those metrics based on their values. This is done based on thresholds that are used to distinguish between acceptable and not acceptable values. Those thresholds do not hold any statistical significance, as they are based on the literature survey or the rationale that is presented in the respective metric section of Section 6.5. Once a benchmark is available the thresholds can be calibrated. (These first two steps can be seen Figure 12.)

Steps 3 and 4

Once all the values of all the metrics are classified based on the thresholds that where set on the previous step, they are used to populate a quality profile for every metric. Those quality profiles that are formed summarize the distribution of the respective metrics. For instance the metric values for the RIC metric are classified between bad and good values and the respective quality profile of this metric has those two categories (step 3 of Figure 12). A percentage that denotes how much the values of each category are, compared to the total values that where derived during the measurement period is then calculated. In the example of Figure 12 (step 4) the quality profile for RIC is 67% bad and 33% good.

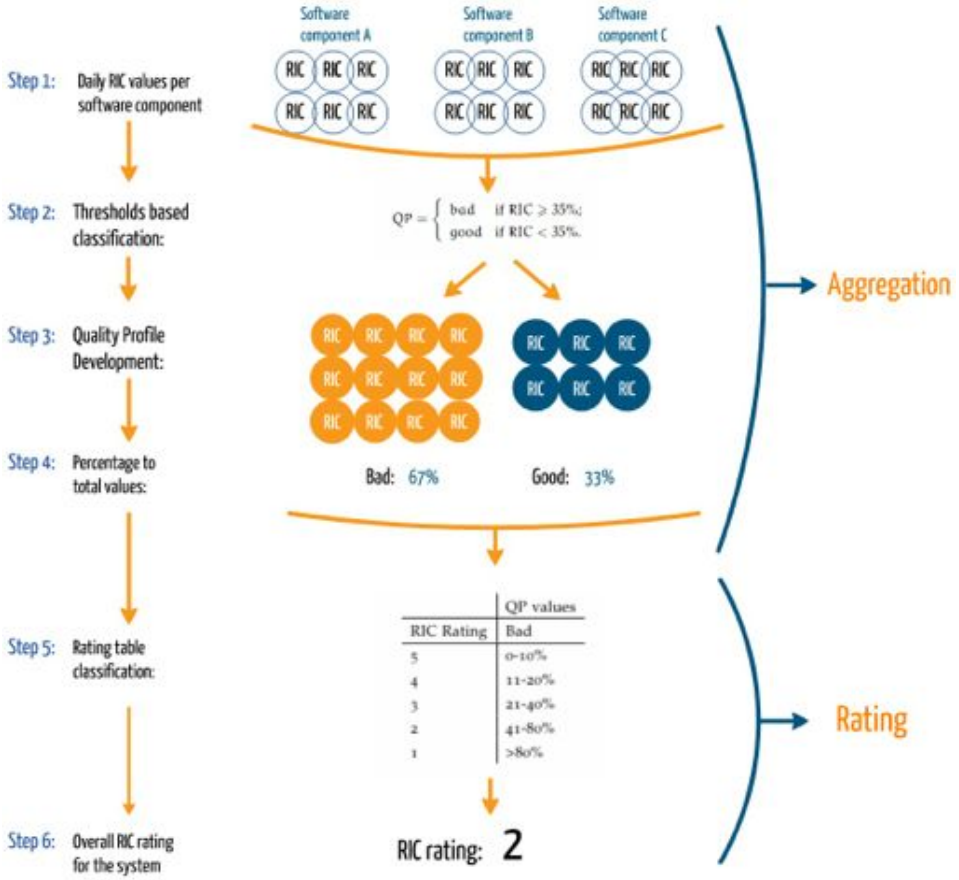


Figure 12: Example Aggregation and rating from software component level to system level for the RIC metric

6.4 RATING

This section describes the methodology that is used to derive system level ratings from the metric results that are obtained from the aggregation. Furthermore, the mapping of those ratings to energy efficiency sub-characteristics is presented. The steps presented here follow those of the preceding section.

Steps 5 and 6

When the quality profiles are obtained, a table is used to derive a rating of the respective metric at a system level. In Figure 12 for instance, the software component metric values for the RIC metric are considered 67% bad which denotes a score of 2 at the system level (steps 5 and 6 of Figure 12). A simple rating scheme was used for this purpose: 1 / 2 / 3 / 4 / 5, with 1 being the worst and 5 being the best rank. This step results in a system level rating for every metric that is used by the model.

After scoring individual metrics, a rating is derived for sub-characteristics of energy efficiency by aggregation according to the mapping of Figure 13. Since, the method that is presented here is indicative, we consider that all metrics share an equal weight and a system level score is derived by the average of each metric that is relevant to each sub-characteristic according to Figure 13. A final aggregation is then made to aggregate the ratings of each sub-characteristic to an overall energy efficiency score for the system. The average is also used here for the same reasons.

	ACC	RIC	CCUW	PGR	PRO	CNS	PG	OPO	ISO
Energy Behavior	X	X	X						
Capacity				X	X				
Resource Utilization						X	X	X	X

Figure 13: Mapping energy efficiency sub-characteristics to metrics. The rows in this matrix represent the three sub-characteristics of energy efficiency based on the conceptual division presented in the model. The columns represent the metrics that where used based on the GQM methodology

6.5 THRESHOLDS

This section presents the thresholds that are used for the aggregation of metrics as well the rating at the system level. In the beginning of each of the following sub-sections a brief elaboration on the threshold choice is given. After that, an array and a table is presented for all the metrics except the ACC. The array represents the metric thresholds whereas the table can be used to compute component rating based on the quality profile derived for every metric.

[ACC] Annual Component Consumption

The ACC metric is a volume based indicator as it expresses the total energy consumption of a software component for a certain period scaled yearly and as such it is difficult to derive relevant thresholds. Therefore, a more simple aggregation method is followed. All the ACC values of the software components are summed up and the results are aggregated to the system level based on the table bellow. Thus the ACC rating of a system is pro-

duced by the classification of the ACC sum of the software components under evaluation.

$$ACC_{agg} = \sum_{i=1}^{n} ACC_{component(n)} \quad (16)$$

ACC Rating	ACCagg(KWh)
5	0-5000
4	5001-20000
3	20001-80000
2	80001-160000
1	>160000

Aggregate at a system level, this metric demonstrated the overall yearly energy consumption of the software system.

[RIC] Relative Idle Consumption

The RIC metric scales from 0-100%. Therefore, the closer this indicator is to zero, the less energy is *"wasted"* due to idle software components. Obviously, it is desirable that this value is zero, or as close to it as possible for every software component. This in extent would mean that the respective components, either produce work constantly or that they consume zero energy when they idle.

The threshold that is used to distinguish between acceptable and unacceptable values for this metric, is similar to the one used by the Energy Star U.S. government certification. Energy Star certification is awarded to systems with "typical" power (a weighted idle, sleep, and standby power consumption) below 35% of the maximum power of the machine [76]. Even though RIC is derived through energy and not power consumption we consider it relatively suitable at that point, to classify RIC values.

$$QP = \begin{cases} bad & \text{if RIC} \geqslant 35\%; \\ good & \text{if RIC} < 35\%. \end{cases}$$

	QP values
RIC Rating	Bad
5	0-10%
4	11-20%
3	21-40%
2	41-80%
1	>80%

The above table can be used for aggregating RIC values of software components at a system level. A relatively low rating after the aggregation, denotes that the software components of the system consume a significant amount of energy when they idle and consequently, their RIC values should be examined more closely to identify optimization opportunities. Possible optimization paths would be for instance, to replace parts of the host resource of a problematic software component with others that have a lower idle power consumption, or by powering down the host resource if significant idle timeframes are available.

[CCUW] Component Consumption per Unit of Work

CCUW expresses the average energy that each component expends to produce each work unit an as such it is difficult to set a logical threshold. We, therefore, rely on the data collected in the case study (Chapter 7) to define the threshold. The threshold selected is the mean value of the available data set.

$$QP = \begin{cases} bad & \text{if } CCUW > mean(dataset); \\ good & \text{if } CCUW \leqslant mean(dataset). \end{cases}$$

	QP values
CCUW Rating	Bad
5	0-10%
4	11-25%
3	25-50%
2	51-70%
1	>70%

At an aggregation level CCUW expresses the average energy that is consumed by the system to deliver one work unit.

[PGR] Peak Growth

The PGR metric scales from 0% to 100% and reveals the deviation of the actual software component power demand is, from the theoretical achievable power provisioning limit that the host hardware can provide, at peak workload. Therefore PGR can be zero, only in cases where the power demand caused by a software component reaches the theoretical provisioning limit of its host resource.

For the purpose of the model we consider values that are between 25% and 10% a suitable threshold. Components that present PG values less than 10% operate very close to their the maximum provisioning capability of their hardware during peak load. This creates implications regarding energy efficiency as a slight increase in peak workload will signal the complete alignment of this two values. This in turn indicates that the respective component will not be able to produce any more useful work at that point, since the host resource cannot provision more, and thus additional latency will be imposed during workload, leading in extent to increased energy consumption. Furthermore, facilitating growth inside the specific host resource, might not be a viable choice without a significant degradation in other QoS like performance.

Resources that present a wide power gap (>25%), are clearly over-provisioned. This in itself might not be immediately inefficient, as provisioning capability is not directly translated to energy consumption. However, a substantial power gap implies that a specific resource operates at low – and thus less efficient – utilizations even at moments when the workload reaches its maximum limit.

$$QP = \begin{cases} good & \text{if } 25\% \leqslant PGR \leqslant 10\%; \\ bad & else. \end{cases}$$

	QP values
PGR Rating	Bad
5	0-15%
4	16-30%
3	31-60%
2	61-80%
1	>80%

Since PGR values point to components that operate close or far from their maximum capacity during peak load, aggregated at a system level PGR is able to potentially identify how much additional production the system can facilitate without damaging its energy footprint.

[PRO] Provisioning

Low percentages at PRO metric point to hardware resources that are rarely used by the application components to produce work and are probably underutilized or idle for a significant amount of time, whereas very high percentages point to the opposite. Both

these cases are harmful from an energy-efficiency point of view, as underutilized resources are a common source of inefficiency whereas resources that operate consistently over the range of 90% utilization although efficient, might lead to serious performance degradation and in extent energy efficiency if any spike in the workload occurs. Therefore values that are in the range of 60-90% are considered ideal for the purposes of this model.

$$QP = \begin{cases} good & \text{if } 60\% < PRO \leqslant 90\%; \\ bad & \text{else.} \end{cases}$$

	QP values
PRO Rating	Bad
5	0-15%
4	16-30%
3	31-60%
2	61-80%
1	>80%

Aggregated, at a system level, PRO can inform the evaluators as to how much energy from the total theoretical that the infrastructure can provide, is actually used by software components to produce units of work, since the idle consumption is not accounted in the calculations.

[CNS] Consumption Near Sweet-Spot

The optimal component consumption will be by definition smaller than or equal to all other cases, hence the values that are produced will be in the region of 0-100% with zero meaning that the respective component consumed significantly more energy compared to its optimal case and 100% indicating a component that operated on the sweet spot. For this metric we define two thresholds and ,thus , three categories in the quality profile.

$$QP = \begin{cases} bad & \text{if } CNS \leqslant 15\%; \\ medium & \text{if } 15\% < CNS \leqslant 40\%; \\ good & \text{if } CNS > 40\%. \end{cases}$$

	QP values	
CNS Rating	Bad	Medium
5	20%	10%
4	30%	15%
3	40%	25%
2	60%	30%
1	-	-

Aggregated at a system level, CNS denotes how close to its optimal energy consumption the system operates when it produces units of work since at first level it is calculated on a per unit of work basis.

[PG] Proportionality Gap

Since PG, denotes the extent at which the component deviates from the optimal proportional case, ideally we would want this value to be 0% or as close to zero as possible. For the purpose of setting a threshold value we chose 30% to distinguish between components. Thus, PG values that are higher than 30% denote components that on average operate at a utilization spectrum that is highly energy inefficient.

$$QP = \begin{cases} good & \text{if } PG \leqslant 30\%; \\ bad & \text{if } PG > 30\%. \end{cases}$$

	QP values
PG Rating	Bad
5	0-15%
4	16-30%
3	31-60%
2	61-80%
1	>80%

At a system level the PG rating quantifies the impact that the overall usage of the system has on its proportionality and in extent its energy efficiency levels.

[OPO] Operational Overhead

Values produced by OPO will never be 0% as at least a portion of the host resources energy consumption will inevitably be used by external components which are necessary for the operations of the software components such as the operating system for instance. However, this is also the point at which administrative choices are quantified by OPO as different operating systems will in most cases require a different amount of energy and therefore, will affect the OPO value in a divergent way. Consider an OPO value of 90%. This means that the respective application component needs a 10% of excess energy per unit of work that it delivers. In other words, for every 1 watt that is used by the application component to produce work an additional 10% (0.1 watt) is needed by the infrastructure to facilitate this production. Looking at the same value from a different perspective, a 90% OPO means that for every 1 watt that is consumed by the host resource, 0,9 watts are used by the application component to produce useful units of work.

From the above it is evident, that the higher the OPO value the less energy is consumed by the host resources for external to the software component operations, which for the purpose of this model, do not generate business value for the application stakeholders. Thus the threshold that is used for this metric is 95% and the categorization can be seen by the following array.

$$QP = \begin{cases} bad & \text{if OPO} \leqslant 95\%; \\ good & \text{if OPO} > 95\%. \end{cases}$$

	QP values
OPO Rating	Bad
5	0-10%
4	11-20%
3	21-30%
2	31-40%
1	>40%

Aggregated at a system level, the rating produced by OPO can inform evaluators as to what percentage of the total platform energy consumption is actually used for production. Therefore, it connects application efficiency with that of the host infrastructure and quantifies to some extent the impact that administrative choices have on energy consumption. By administrative choices, we refer to the choice of the operating system, firmware, or any kind of external to the application software that shares the infrastructure with the application, as well as the choice of hardware resources and the overall system deployment.

[ISO] Isolation

This research did not identify any feasible path to measure the Isolation indicator. Therefore thresholds as well as an aggregation table is not presented, since we cannot be sure beforehand what values such an indicator would produce. We envision a similar aggregation methodology such as the one presented in this chapter, which would map the Isolation rating to the resource utilization sub-characteristic.

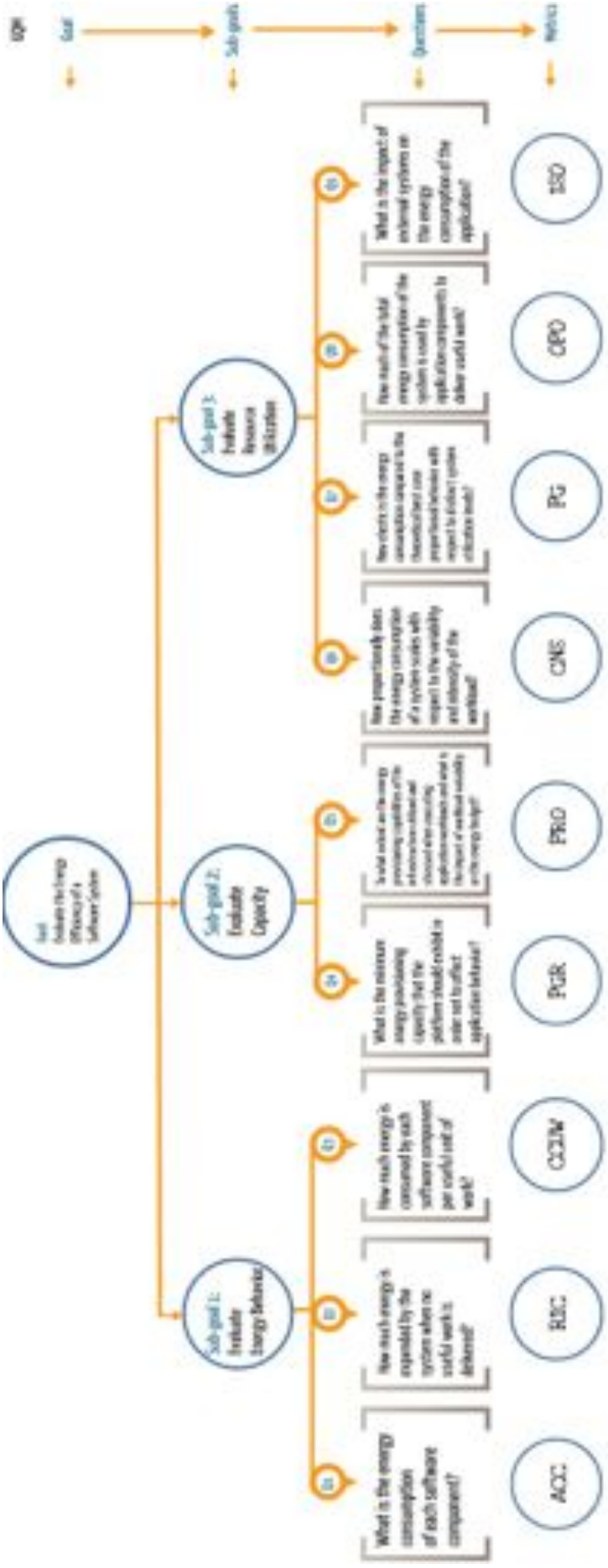


Figure 14: Overview of Goal-Question-Metric approach

7 CASE STUDY

This section presents the information regarding the case study that was conducted in order to evaluate the model that was proposed in the previous section.

7.1 THE SYSTEM

One of the main services of SIG is the Software Risk Monitoring service. This service takes as input source code snapshots of an application or a portfolio of applications, analyzes them by applying suitable metrics and presents the results via a website. SIG consultants then use the results of the analysis in order to come to recommendations and actionable advice regarding aspects of the application that are the main risk and cost drivers in order for the clients to be able to optimize their systems in a feasible way. The clients on the other hand, can use the web service that SIG offers to track the process of optimizations and monitor their systems with respect to the various measurements that SIG deploys.

In the context of this thesis the software components that are used to sustain the main functionality of the Software Risk Monitor where chosen as a source to evaluate the model.

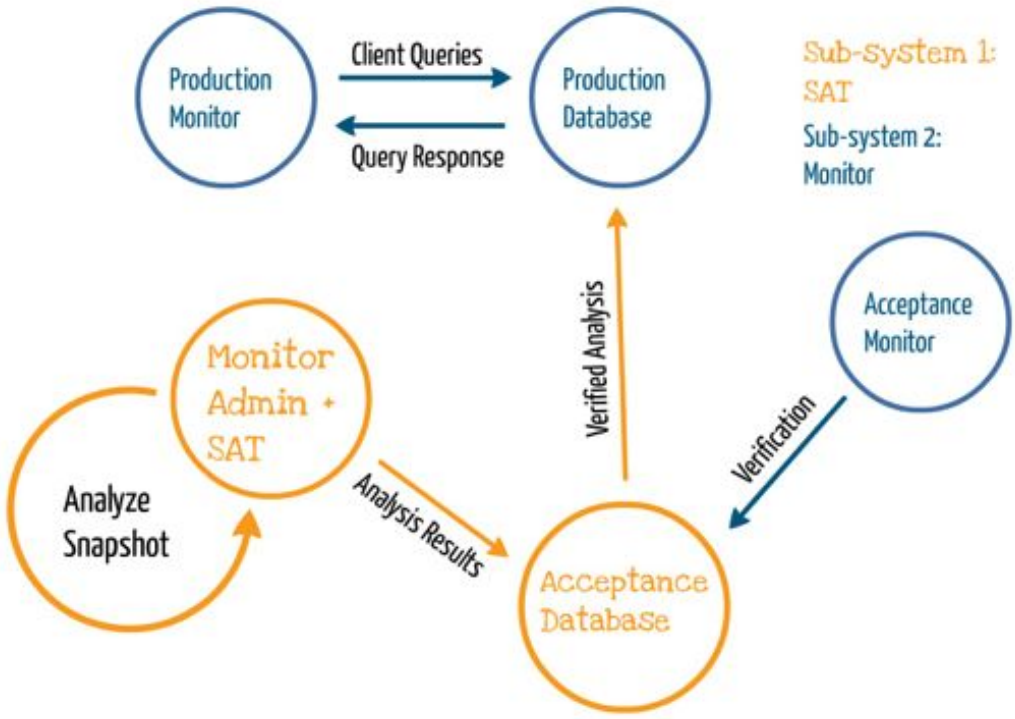


Figure 15: Case study System Overview

As can be seen by Figure 15 the system is consisted of five software components which are hosted by five hardware resources , respectively. Since, direct energy measurements where not feasible in the context of this case study and since the machines that are used to host these components are similar, we assume that all the hardware resources are the same regarding their technical characteristics.

This system can be further divided into two sub-systems for the purpose of determining a useful unit of work for every software component. The first system consist of the monitor admin + SAT and acceptance database. It is used to analyze source-code snapshot and provide data to sub-system 2. We will refer to sub-system 1 as the SAT. The second is consisted of the production monitor, production database and acceptance monitor and can be used by clients to track their system's evolution and current state. We will refer to this sub-system as the Monitor.

It is important to note here that the distinction between sub-systems is only made in order to attribute a useful unit of work to software components based on the functionality that those components are called to sustain. Thus, this classification between those two sub-systems does not hold any merit for the purposes of aggregation.

7.2 SUB-SYSTEM 1: SAT

On a weekly basis, SIG clients send source-code snapshots of their software and the monitor admin component is responsible for pulling the scheduled snapshots and starting the analysis which is done by the SAT through applying the necessary metrics and indicators. In order to reduce the overhead we attribute all the analysis of the SAT to the monitor admin component which means that we consider those two components as one. For the rest of this document we will use the name monitor admin to denote both the monitor admin as well as the SAT component.

The monitor admin is a computationally intensive component that works almost constantly. When a snapshot is calculated the results are forwarded to the acceptance database component. The monitor admin component then pulls the snapshots of the next application that is scheduled for analysis and continues the same routine.

The acceptance database component is responsible for storing analysis results, coming from the monitor admin component. For every application that is analyzed a separate database is maintained by the acceptance database component and every new analysis is appended to the respective database.

Of course this is a simplistic overview of the main functionality of the system as the system has more elaborate functions to support its main operations. However they are considered auxiliary to the analysis.

7.3 SUB-SYSTEM 2: MONITOR

The second sub-system is the Monitor which is can be used by clients to monitor the evolution of their applications. This sub-system consists of the acceptance monitor, production monitor and production database software components.

The acceptance monitor component is used by SIG developers and technical consultants, in order to verify that new analysis results are suitable to be forwarded into the production database. The acceptance monitor component is used to facilitate this verification.

Once an analysis is considered suitable for the production environment, the data is forwarded to the production database. Therefore, the production database, is essentially

a verified copy of the acceptance database component, which is used to store all the concrete analysis results for all the systems that are part of the Software Risk Monitoring service.

Clients then use the production monitor component in order to view the analysis results regarding their application portfolio. The production monitor queries and retrieves the suitable analysis results based on the client and application portfolio, from the production database component.

7.4 UNITS OF WORK

Since it was decided that the system should be conceptually divided into two subsystems based on the functionality that it provides, it is also necessary to distinguish between two functional work units.

For the SAT sub-system the unit of work that is set in the context of this case study, is the successful analysis of a set of snapshots that correspond to the source-code of a client application. Accordingly, a client session is considered as a unit of work, for the monitor sub-system.

For the purpose of calculating the model indicators, the work unit that will be used for each application component, corresponds to the respective useful unit of work of the sub-system that it is part of. Therefore the useful unit of work categorization is the following:

- Unit of Work: User Session for software components:
  - Production Monitor
  - Production Database
  - Acceptance Monitor
- Unit of Work: Application Analysis for software components:
  - Acceptance Database
  - Monitor Admin

7.5 DATA COLLECTION AND ANALYSIS

The model requires four distinct inputs. The first is more conceptual in nature and was described above. It is the system architecture and deployment which is necessary in order to target the measurements correctly, but also to define the most suitable work units for the system. The other three are presented in the following section. An example overview of the data collection and analysis methodology for one software component is given in Figure 16.

7.5.1 *Utilizations*

The second input that is required by the model, is the host resource and application component utilizations for all the resources and components that where identified as part of the system in the previous step. Following, an explanation on how the required data to derive those utilization values is given, based on the case study that was conducted.

All the software components that are in scope, are hosted in machines running some Linux distribution. Therefore, and in order to collect the necessary data to derive host resource as well as software component utilization a script was developed.

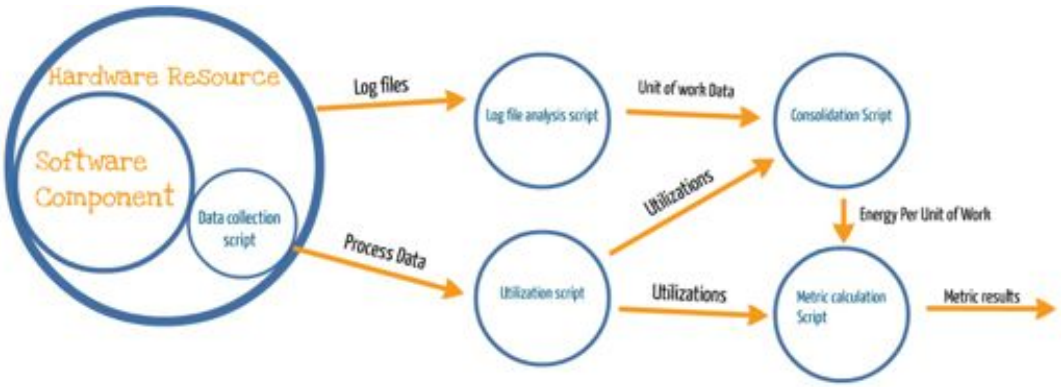


Figure 16: Data Collection and Analysis Overview

The main purpose of the script is to track and collect data regarding the processes and sub-processes that originate from the preceding software components and statistics regarding the CPU of their respective hardware resource. Therefore the script needs to be deployed in all five host hardware nodes of the system. More information on this script can be found in Section A.1.

For every hardware resource and every application component at least two files (one for the CPU and at least one for the application component) where generated every second during the course of a seven day measurement period. Therefore this methodology produces at least 1 million files per component, assuming that the component is active during the entire measurement period.

In retrospective, the script could have been modified to execute some of the necessary parsing, that will be explained below, and some computations directly on the host resource before storing the files. We chose to avoid this since the measurements took place on an actual production environment and new additional load could potentially create implications.

Nevertheless, since during this case study the script generated an enormous amount of data, a compressing functionality was implemented to avoid swarming the file system of the hosts. This was done by periodically compressing the respective folder in which the output data was stored and emptying it afterwards to clear it for the forthcoming data.

The generated data from the preceding script was collected and analyzed on a daily basis. To perform the analysis and derive utilization results for the application components as well as their host resources an R script was developed. This script takes a folder with data that is generated by the previous script as an input and produces a table that depicts total CPU as well as application component utilization for every second of the measurement period. More information regarding the implementation details of the script are given in Section A.2.

During the measurements it was observed that in some cases some samples where missing. In order to fill this small gaps the utilizations for the missing seconds was calculated by taking the average utilization of their preceding and their following second.

7.5.2 *Units of work*

The third input that is required by the model is time and duration statistics regarding the useful unit of work production. For this case study two distinct units of works were chosen based on the elaboration that was presented before. Therefore, the suitable log

files should be parsed for the period in order to derive the amount of units of work and their duration. In this regard, two scripts where developed for to parse the respective log files that are generated by the two sub-systems in order to find correct key words and timestamps to distinguish the starting and ending points for the system analysis (SAT components) and the user session (Monitor components) work units. These log files where analyzed on a daily basis and produced a data set with starting and ending timestamps of work units of the respective sub-systems.

7.5.3 *Power Estimation*

The final input that is required by the model is power consumption information for the hardware that hosts application components. Since direct power measurements where not an option in the context of this thesis a linear power model – since we assume that all the host resources have the same configuration – was used based on measurements that where done on the servers used for experimenting at SEFLab[1]. This model was used to derive the energy consumption of software components and host resources instead of direct power measurements. The linear equation that was used is the following:

$$Power = ((Power_{max} - Power_{idle}) \times Utilization) + Power_{idle} \quad (17)$$

In accordance to SEFLab measurements, $Power_{max}$ was set at 220W and $Power_{idle}$ was set at 160W, to reflect the maximum and idle power consumption of the resource. $Utilization$ is the CPU utilization value of the hardware resource or the software component based on the needs of the respective metric.

In retrospective, in order to acquire more accurate measurements regarding energy consumption two methods can be used. One is to measure the actual power of host resource per second for the entire period, however this requires additional hardware instrumentation as there are no APIs to derive such measurements for most hardware resources. The second is to use power models for the actual or from similar hardware. The power model that was used is linear which is not valid for most modern resources. Also, the same power model was used for all hardware since we assume they are exactly the same.

7.5.4 *Combining information*

In order to calculate the metrics of the model, there are some certain steps to be followed. First, software component as well as hardware resource utilization data has to be related to their energy consumption. Thus, a script was developed to map utilizations to the respective power consumption based on Equation 17. This was done for both host resource as well as the application components utilization and produced a table with utilizations and respective power consumption for all hardware resources and software components. The next step is to derive the per unit of work consumption from that table. More information on this script can be found in Section A.3

1 SEFLab [28] was deployed by SIG, in order to enable researchers to study the energy footprint of software applications. The setup of the lab includes infrastructure that enables measurements at hardware and software level and alignment of the resulting measurement streams, so that software behavior can be effectively mapped to resource usage and energy consumption.

7.5.5 *Metric and Rating*

A final script was used to calculate the metric results. This script take as input the data stream and calculates the metrics for all the application components and hardware resources. Metrics are then aggregated and a system level rating is calculated.

7.6 RESULTS

This section presents the results of the case study. The analysis and interpretation follows in the next chapter. Tables 1 to 6 and 9 to 10 display daily data for each software component under evaluation, for the units of work, `ACC`, `RIC`, `CCUW`, `PGR`, `PRO`, `PG` and `OPO` respectively. For this set of tables, except Table 1 the first set of rows represents the results of the repsective metric whereas the last set of rows represent the aggregation and the rating at a system level.

In the case of `CNS`, only the aggregated data is presented in Table 8. This is because there are too many data-points to fit in a table since `CNS` was calculated on a per unit of work per software component basis. Table 7 presents an alternative aggregation method that could have been used in order to obtain a rating at the system level. This table is only presented here for reasons of consistency. Table 11 displays the rating per model metric. Finally, Table 12 presents the rating per sub-characteristic and the overall energy efficiency rating of the system.

Date	10-07-2013	11-07-2013	12-07-2013	13-07-2013	14-07-2013	15-07-2013	16-07-2013
Acceptance Database	19	15	38	52	49	58	29
Acceptance Monitor	46	40	40	6	4	63	72
Monitor Admin	19	15	38	52	49	58	29
Production Database	46	40	40	6	4	63	72
Production Monitor	46	40	40	6	4	63	72

Table 1: Units of Work delivered by software components.

ACC(J)	10-07-2013	11-07-2013	12-07-2013	13-07-2013	14-07-2013	15-07-2013	16-07-2013
Acceptance Database	719,455,571	719,710,107	719,761,718	719,310,028	719,401,025	719,942,831	719,967,799
Acceptance Monitor	719,164,758	719,214,070	719,229,347	719,155,164	719,056,426	719,276,872	719,243,570
Monitor Admin	729,256,905	724,781,707	720,503,614	731,178,145	742,223,256	734,266,715	736,036,671
Production Database	719,131,492	719,233,463	719,148,496	719,095,546	719,459,584	719,915,320	719,450,911
Production Monitor	719,923,673	719,898,636	719,940,146	719,891,998	719,924,081	719,980,244	720,077,140
Aggregate(J):	25,265,207,025						
Aggregate(KWh):	7,013						
Rating:	4						

Table 2: [ACC] Annual Component Consumption metric results, aggregation and rating. The results are presented in Joules whereas the aggregation is done in KWh.

RIC	10-07-2013	11-07-2013	12-07-2013	13-07-2013	14-07-2013	15-07-2013	16-07-2013
Acceptance Database	67.44%	68.85%	61.21%	67.49%	68.41%	61.79%	62.93%
Acceptance Monitor	48.76%	49.35%	48.14%	49.32%	49.84%	46.16%	44.66%
Monitor Admin	0.002%	0.011%	0.023%	0.007%	0.006%	0.001%	0.001%
Production Database	89.89%	91.02%	90.23%	92.01%	81.68%	74.83%	85.40%
Production Monitor	66.74%	65.72%	64.44%	66.23%	66.20%	64.80%	63.49%
Overall:	Bad	Good					
Aggregation:	80%	20%					
Rating:	2						

Table 3: [RIC] Relative Idle Consumption metric results, aggreagation and rating.

CCUW(J)	10-07-2013	11-07-2013	12-07-2013	13-07-2013	14-07-2013	15-07-2013	16-07-2013
Acceptance Database	728,193	922,771	364,023	266,017	282,339	238,707	477,432
Acceptance Monitor	300,654	345,755	345,783	2,304,984	3,457,002	219,559	192,105
Monitor Admin	738,114	929,207	364,627	457,538	291,296	243,457	488,240
Production Database	300,640	345,785	345,744	2,304,793	3,458,940	219,754	192,161
Production Monitor	300,971	346,105	346,125	2,307,346	3,461,173	219,774	192,328
Overall:	Bad	Good					
Aggregation:	22.86%	77.14%					
Rating:	4						

Table 4: [CCUW] Component Consumption per Unit of Work metric results, aggregation and rating. The results are presented in Joules

PGR	10-07-2013	11-07-2013	12-07-2013	13-07-2013	14-07-2013	15-07-2013	16-07-2013
Acceptance Database	25.22%	24.21%	22.00%	24.33%	24.17%	21.14%	20.65%
Acceptance Monitor	6.22%	3.36%	5.71%	6.26%	4.57%	5.89%	5.69%
Monitor Admin	0.61%	1.70%	2.06%	1.86%	2.14%	2.42%	2.23%
Production Database	23.71%	22.82%	23.08%	23.28%	23.08%	23.67%	23.14%
Production Monitor	7.51%	13.37%	7.28%	14.35%	15.94%	8.77%	8.03%
Overall:	Bad	Good					
Aggregation:	54.28%	45.72%					
Rating:	3						

Table 5: [PGR] Peak Growth metric results, aggregation and rating.

PRO	10-07-2013	11-07-2013	12-07-2013	13-07-2013	14-07-2013	15-07-2013	16-07-2013
Acceptance Database	23.70%	22.68%	28.26%	23.67%	23.01%	27.85%	27.01%
Acceptance Monitor	37.28%	36.86%	37.74%	36.87%	36.50%	39.18%	40.27%
Monitor Admin	75.36%	75.98%	76.86%	76.09%	76.11%	76.31%	75.94%
Production Database	7.35%	6.54%	7.11%	5.81%	13.34%	18.33%	10.63%
Production Monitor	24.22%	24.97%	25.90%	24.60%	24.62%	25.64%	26.60%
Overall:	Bad	Good					
Aggregation:	80.00%	20.00%					
Rating:	2						

Table 6: [PRO] Provisioning metric results, aggregation and rating.

Components	Acceptance Database			Acceptance Monitor			Monitor Admin			Production Database			Production Monitor		
	B	M	G	B	M	G	B	M	G	B	M	G	B	M	G
10-07-2013	47.36	21.07	31.57	76.08	19.58	4.34	42.1	26.33	31.57	76.08	19.58	4.34	76.08	19.58	4.34
11-07-2013	60	20	20	67.5	30	2.5	66.67	13.33	20	67.5	30	2.5	67.5	30	2.5
12-07-2013	44.63	21.16	34.21	82.5	17.5	0	42.10	26.33	31.57	82.5	17.5	0	82.5	17.5	0
13-07-2013	11.53	32.71	55.76	100	0	0	NA	NA	NA	100	0	0	100	0	0
14-07-2013	14.28	53.07	32.65	100	0	0	16.32	55.11	28.57	100	0	0	100	0	0
15-07-2013	31.03	44.84	24.13	57.14	41.268	1.58	31.03	41.39	27.58	57.14	41.268	1.58	57.14	41.268	1.58
16-07-2013	55.17	6.90	37.93	50	48.612	1.38	51.72	3.46	44.82	50	48.612	1.38	50	48.612	1.38

Table 7: Alternative CNS aggregation.

Table 7 presents an alternative aggregation, that could have been followed for the CNS metric, for reasons of consistency. The values in the cells are percentages. The columns represent the score of each component if the aggregation was made on a daily basis. Here the quality profiles are formed on a per component, per transaction, daily basis and a different aggregation and rating scheme could be followed to derive a system level rating from those results.

CNS	Bad	Medium	Good
Aggregation:	56.17%	35.36%	8.46%
Rating:	1		

Table 8: [CNS] Consumption Near Sweet-Spot metric aggregation and rating.

The aggregation and rating that is presented in Table 8 is based on the results of this metric on a per transaction per component basis for the entire measurement period. This is the approach that was followed for obtaining a system level rating for this metric.

PG	10-07-2013	11-07-2013	12-07-2013	13-07-2013	14-07-2013	15-07-2013	16-07-2013
Acceptance Database	72.56%	72.49%	72.38%	72.48%	72.47%	72.33%	72.33%
Acceptance Monitor	72.63%	72.61%	72.61%	72.63%	72.64%	72.60%	72.61%
Monitor Admin	65.91%	64.33%	62.11%	64.24%	64.24%	63.58%	64.48%
Production Database	72.65%	72.62%	72.64%	72.65%	72.55%	72.43%	72.55%
Production Monitor	72.44%	72.44%	72.42%	72.44%	72.43%	72.42%	72.39%

Overall:	Bad	Good
Aggregation:	100%	0%
Rating:	1	

Table 9: [PG] Proportionality Gap metric results, aggregation and rating.

OPO	10-07-2013	11-07-2013	12-07-2013	13-07-2013	14-07-2013	15-07-2013	16-07-2013
Acceptance Database	99.06%	99.22%	98.91%	99.18%	99.23%	98.88%	98.82%
Acceptance Monitor	99.12%	99.14%	99.12%	99.15%	99.16%	98.87%	99.13%
Monitor Admin	94.34%	93.70%	93.06%	93.83%	94.41%	93.73%	93.76%
Production Database	99.13%	99.15%	99.10%	99.16%	98.50%	98.14%	98.87%
Production Monitor	99.03%	98.99%	99.02%	99.05%	99.05%	99.01%	99.05%
Overall:	Bad	Good					
Aggregation:	20%	80%					
Rating:	4						

Table 10: [OPO] Operational Overhead metric results, aggregation and rating.

Metrics	ACC	RIC	CCUW	PGR	PRO	CNS	PG	OPO	ISO
Energy Behavior	4	2	4						
Capacity				3	2				
Resource Utilization						1	1	4	NA

Table 11: Mapping of Aggregated Results to energy efficiency sub-characteristics.

The rows of Table 11 represent the sub-characteristics of energy efficiency. The columns represent the rating of all model metrics.

	Rating
Energy Behavior	3.33
Capacity	2.5
Resource Utilization	2
Aggregate:	2,61
System Rating:	3

Table 12: Rating per sub-characteristic of energy efficiency.

The first rows of Table 12 represent the aggregated average values per energy efficiency sub-characteristic from the metric rating presented in Table 11. The last row presents the overall energy efficiency rating of the system which is derived by rounding the aggregate value of the previous row.

8 ANALYSIS AND DISCUSSION

8.1 INTERPRETATION

This section interprets and discusses the results that were obtained from the case study. A general interpretation of all the metrics was given in the Section 6.3, to justify the thresholds that were set as well as the aggregation method that was followed.

Ideally, optimizations should at a first step focus on the sub-characteristics of energy-efficiency that are ranked lower at a system level. Table 12 presents the aggregated system rating for the case study. The lowest ranked sub-characteristic of energy efficiency is resource utilization and thus optimizations should first focus at that characteristic. Then examining the values per metric will reveal those with the lowest score at the system level. Table 11 reveals that `CNS` and `PG` are the metrics that score lower at the system level. Subsequently, navigating to the underlying level for those metrics, will identify those components that have the lowest value and thus optimizations should focus on improving the energy footprint of the respective worst contributors. This should be done for all the metrics that have a relatively low score.

Findings, will be discussed in order of the tables that were presented in Section 7.6. Wherever necessary, metric information will be combined to come to a more complete evaluation regarding the energy efficiency levels of the software components. Thus, the model indicators will be discussed at a software component level. The Isolation metric is not presented here since it was not measured.

8.1.1 *[ACC] Annual Component Consumption*

In general, `AAC` enables evaluators to identify the biggest energy consumers among a set of software components. This in turn, enables them to more accurately target optimization efforts towards the most costly ones. The value of `AAC` can act as a milestone which will enable system administrators to spot energy fluctuations, by analyzing how it evolves. Consequently, `ACC` can continuously be monitored to enable stakeholders to track the energy behavior of software components and spot any deviations or trends.

Analyzing the results of the case study of Chapter 7 provides some more insight on how to interpret this metric. At a first stage, the `ACC` values (Table 2) point out that the monitor admin component is the highest energy consumer of the system but with little difference. The relatively uniform consumption among the software components can be partially attributed to the use of the identical linear power models to derive energy consumption results and since this component presents the higher utilization for the period (11.84%) it is expected to draw more energy in total. Furthermore, the dynamic range of the power model that was used to derive energy consumption (220W peak power, 160W idle power, dynamic range 60W) is relatively small[1]. Since for a linear power model, power consumption scales proportionally to the utilization levels of the component a small dynamic range in the power consumption means that the differences in utilization levels of the components will not have a significantly visible impact on

1 According to [59] a typical blade server – based on HP's c-series half-height blade designs – has an active power consumption of 450W an idle power consumption of 270W with a dynamic range of 180W

the consumption of the host resources. Nevertheless, since the monitor admin is the component that arguably exhibits the highest functional complexity – analyzing client snapshots – among the set of the components, this increased consumption is to some extent justified.

8.1.2 *[RIC] Relative Idle Consumption*

`RIC` is comparable between software components and as such, at a first level it can be used for making direct component comparisons regarding their idle energy consumption which evaluators can then use to identify the biggest idle consumers of the system.

In the results of the case study (Table 3), we observed that the highest contributor is undeniably the production database as the largest portion of its energy consumption occurs when the component idles. Since the percentages are quite high, optimizations should focus at reducing the idle consumption of this component. Nevertheless, since the production database is a component that should be operational 24/7 due to availability constraints, powering down the resource for certain time frames for instance, might not be a feasible optimization choice and thus any efforts to increase the energy efficiency of this component should be done by maintaining any availability requirements. Feasible optimizations in that case would be to host the production database to a hardware resource that has hardware components with lower-power idle states.

On the other hand the monitor admin component seems to be operating almost continuously during the entire period and thus its `RIC` values are significantly small. This was expected to some extent since the monitor admin is a component that is continuously analyzing client source-code. Thus, by interpreting `RIC` and `ACC` in combination, it is obvious that although the production database has relatively similar `ACC` values compared to the other components, this consumption is mainly attributed to the idle state and thus this portion of energy is wasted since it is not used for work production.

8.1.3 *[CCUW] Component Consumption per Unit of Work*

The `CCUW` indicator (Table 4) can be used to identify temporal trends in consumption with respect to workload variability and the unit of work that was set for each component.

If the components are classified based on the sub-system that they belong to, a first observation that can be made is that components that belong to the same sub-system share a similar distribution of `CCUW` values. Thus, their per unit of work consumption changes similarly throughout the measurement period.

A second observation that can be made is that the two sub-systems exhibit a high divergence in their `CCUW` values, as the distribution of those values for the Monitor subsystem (acceptance monitor, production database, production monitor) is by far different than the one observed in components that belong to the SAT sub-system (acceptance database, monitor admin). At some extent, this is because those two sub-systems produced a different amount of work units throughout the measurement period.

Another observation that can be made is the tremendous rise of `CCUW` values between 13-07-2013 and 14-07-2013 for the Monitor sub-system components (Figure 17). This is understandable, since the unit of work that was used for this set of components was a client session and therefore a significant decrease during that period which was a weekend, was expected. Consequently, a portion of any optimization efforts should be targeted at reducing the energy consumption of those components during the weekend, since this consumption is not justified by the work they produce, compared to their aver-

age case. Virtualization can be used for instance, to host some components on the same hardware resource in order to reduce the overall energy consumption of the components.

Figure 17: CCUW values for production monitor for the measurement period

Naturally, different units of work will present huge differences in their energy consumption values. For instance, a highly complex transaction (e. g.calculating a three-day weather forecast) is expected to consume more energy than a static web-search. Therefore, this indicator cannot be used for making direct comparisons between different component classes at a software component level. However, in cases where similarities exist (in terms of component type and functional complexity), this indicator can be used for efficiency comparisons.

Furthermore, expressing energy costs through business-relevant workloads also enables an organization to plan and make more accurate administrative decisions since it is much more convenient to project future demand – and in extent work output – and optimize for efficiency when it is expressed in terms that make business sense. Consequently, this indicator can be used to make a straightforward estimation of the energy demand based on the expected intensity of forthcoming workload, but can also act as milestone to spot consumption trends and deviations in relation to the work output of the system. Since `CCUW` is calculated for each software component, planning can also be done more accurately since forthcoming workload intensity might be different for some parts of the system.

8.1.4 *[PGR] Peak Growth*

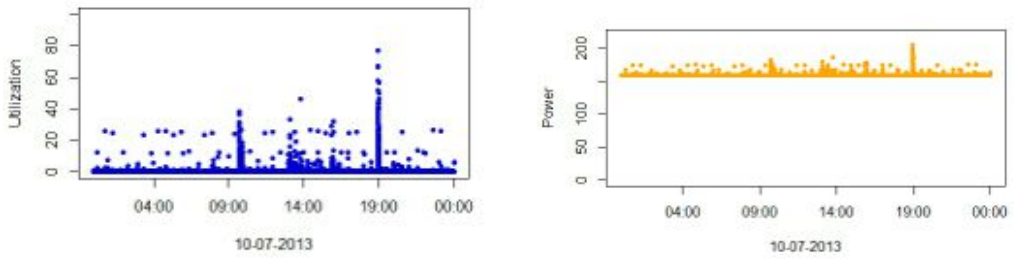
A high amount of `PGR` values outside the acceptable threshold region denotes that many application components might operate very close or very far from their hardware resource's power provisioning capacity during peak load. On the former case, growth might not be able to be facilitated inside the given infrastructural context of the system. On the later case, parts of the system will be underutilized and thus inefficient even at their peak load.

The first observation that can be made based on the `PGR` metric (Table 5) is the very small values that the monitor admin component exhibits. This indicates that this component reaches the maximum power provisioning capacity of its host resource on a daily basis. Since the maximum power that is drawn by each component is related to the utilization levels that it exhibits, it is clear that the monitor admin operates at very high utilizations at least for some minimal time-frame on a daily basis.

At a first level, further investigation is needed to determine the impact that these spikes in utilization have on performance of this component, since any additional latency that is caused by performance degradation is translated to increased energy consumption. Consequently, any increase in the peak load of the monitor admin component, can potentially have significant impact on its energy footprint, as the host resource does not seem capable of providing the required power. To alleviate that problem, it would be possible to facilitate additional load by changing the time disposition of the current workload – if possible – in order to stress the component more homogeneously so that during peak load the PGR values can be within the acceptance thresholds.

Observations can be made based on the values derived for the other components. For instance, the two databases both exhibit sustainable low PGR values. This denotes that even during their peak load, their host resources have further provisioning capacity for facilitating additional workload. Furthermore, it might be feasible for instance, to virtualize those components and host them on the same hardware, if their peak load moments have different time disposition, in order to improve the overall energy footprint of the system (availability, security etc. requirements are external to the model).

Further observations can be made by examining the peak utilization of the components in order to make a more accurate interpretation of the PGR metric. The acceptance monitor component for instance, presents very small PGR values throughout the period. By examining the energy consumption of this component, for the first measurement day, it is clear that it operated at peak workload only for a small time frame, with very low utilizations throughout the duration of the day (Figure 18). This type of information can be used to more accurately target any optimization efforts.



(a) Utilization of acceptance monitor during 10-07-2013

(b) Power consumption of acceptance monitor during 10-07-2013

Figure 18: Utilization and Power Consumption for Acceptance Monitor(10-7-2013)

Generally, at a software component level, if PGR values are above the acceptance threshold, possible solutions to alleviate that problem, would be to alter certain workload characteristics or restructure the flow of the work unit, so that growth can be facilitated without the need for additional hardware resources. If one server for instance has a very high PGR value and another has a very low, any transactions during peak workload can be potentially routed to the second server so that growth can be facilitated without any negative impact in the energy footprint of the system. Moving portions of workload to different time frames to reduce utilization is also a commonly used and viable strategy in some cases.

In the case that PGR values are below the acceptance threshold, a suitable optimization path would be to consolidate servers, and move software components with very low PGR values to the same hardware.

8.1.5 *[PRO] Provisioning*

The PRO indicator demonstrates how much of the total energy consumption that the host resource could have provided on a daily basis, is actually used for work production by the respective software component. Therefore, apart from informing evaluators about the average energy usage of a software component compared to the maximum theoretical provisioning capability of its host resource, this metric can also help them to more accurately understand the impact of the workload to the overall energy consumption the system, enabling them in extend to follow the more accurate optimization path.

From the values presented in Table 6, it is clear that the monitor admin uses his hardware extensively whereas the production database does not. At an initial stage this raises some questions, since the ACC results (Table 2) demonstrate that at some extent, all software components have a similar energy consumption. However, since the PRO indicator is calculated based on the active consumption of the respective component (that is the ACC value of the component minus its idle consumption), the very low values of the production database are justified since according to RIC results (Table 3), it uses most of its energy at the idle state and thus the PRO indicator for that component will be significantly low. Consequently, optimizations with respect to PRO results should at a first stage focus on the production database component since it is the one that uses its hardware for work at a very low rate consistently.

Feasible optimizations would be to host the production database component along with other software components on the hardware since at least for the measurement period the production database does not present any significant energy demand. Of, course other components also present a relatively low PRO indicator and thus virtualization can be deployed so that less hardware can be used in order to sustain the operations of the components.

8.1.6 *[CNS] Consumption Near Sweet-Spot*

CNS (Table 8) demonstrates in a quantitative nature, how elastic the energy consumption of a software component is with respect to the variability and intensity of the workload. Since CNS is calculated on a per unit of work and per software component basis, meaning that the energy expenditure during every unit of work is divided by the optimal energy consumption that was observed during the measurement period per software component, only the averages for every component are presented here (Table 13).

CNS	Avg. for period
Acceptance Database	20.81%
Acceptance Monitor	13.35%
Monitor Admin	29.70%
Production Database	13.35%
Production Monitor	13.36%

Table 13: Average CNS values for software components

From Table 13, it is obvious that all the components that belong to the Monitor sub-system present almost an identical scalability. This can be partially justified by a closer look at component utilization. For the first measurement day for instance, the acceptance and the production monitor components present almost the same average utilization levels (0.13% and 0.39%, respectively, with their maximum utilization reaching 77.19% and 72.45%). Furthermore, the units of work that where produced during that day where 46 in number with an average duration of 32 minutes, occupying a large period of the day (units of work can be concurrent). Consequently, the components are expected to present similar `CNS` values since they have very similar average utilizations. Furthermore, the first ten units of work with the smallest energy consumption for each component of the Monitor sub-system, where omitted when calculating `CNS`. This is because when parsing the log files we observed that some units of work had a very small duration and thus energy consumption. After a closer inspection we identified that this was caused by two reasons. First there were some user sessions that could not be justified (a user session that lasted 3 seconds for instance). Second, because the energy consumption per unit of work was calculated on a daily basis, if there was some user session that started relatively late in a day (e. g.23:59:30), its ending point was set at 23:59:59 although its duration might have been larger. Thus it presented a very low energy consumption which was not representative to the actual case. We chose to filter out the preceding units of work since they would not represent an accurate value for the optimal energy consumption of each component. As a result, all the components of that sub-system presented almost the same optimal energy consumption per unit of work.

Finally, the relatively similar `CNS` values for components that belong to the same sub-systems can be attributed to the way the correlation between unit of work duration and respective energy consumption was made. The duration and time information regarding the work units were based on the log files of the monitor admin and the production monitor component since log files where not available for the other software components of the system. Obtaining additional information for those components would require further software instrumentation which was not possible due to time constraints and because we wanted to keep any kind of additional software instrumentation as minimal as possible.

Choosing to use the log files of the monitor admin and the production monitor as a source to derive unit of work information for the other software components of the system does not seem to hold any merit. This choice was partially justified since the acceptance monitor, the production database and the acceptance database are counted as auxiliary components to the two main components of the system (monitor admin and production monitor), since those two are the ones responsible for the delivering the main functionality of the two work units that where set for this case study. However for instance, when the monitor admin component finishes analyzing a snapshot the acceptance database does not produce any useful work. Its contribution starts when the new snapshot is forwarded to it and thus work for that component starts some time after the time that was used for calculating the `CNS` indicator. Consequently, the `CNS` values for the auxiliary components do not present – with significant accuracy – their respective scalability.

Since the log files of the monitor admin and the production monitor where used to derive unit of work information, the `CNS` indicator can only be used to make direct comparisons between those two components. From their respective values, it is obvious that both of them present a very low scalability with respect to their best case energy consumption and thus both should be optimized to some extend in order to operate closer to that point.

The first candidate for optimizations is the production monitor since it presents a lower CNS value on average and consequently it does not scale its energy consumption according to the variability of the workload. Usually, proportionality issues as the ones that are depicted in the CNS values are caused by very high consumption at the idle state or at lower utilizations. Since CNS was calculated based on a per unit of work basis, it indicates that the above software components usually operate at lower utilizations when they produce work compared to their best case, or that their work units presents a high deviation in their duration and thus more energy is consumed for units that last longer. Of course, both these cases can happen simultaneously.

Indicative optimizations to increase the energy efficiency of those components, would be to virtualize them, and use a software driven workload placement and Virtual Machine migration approach in order to increase the utilization of some host resources close to the peak, while suspending other machines during low workload periods [90][15] [92][32][74][30]. Of course, this requires careful planning to avoid resource contention incidents, which as was previously explained, result in energy waste since they impose additional latency.

8.1.7 *[PG] Proportionality Gap*

The PG indicator (Table 9) aims at evaluating how elastic the energy consumption of the components is, with respect to their average usage. Thus, it captures both the idle as well as low utilizations spectrum inefficiencies. Since at a software component level PG is calculated based on their respective average utilization level that they operate upon, by looking at PG values, evaluators can identify for every component, utilization regions in which the PG values are relatively high and try to shift resource utilization to levels that exhibit narrower gaps. This can be mainly done through resource consolidation or workload scheduling and migration, although other techniques such as the use of hardware resources with lower-power active states, for the software components that exhibit high PG values, is a feasible optimization path.

Table 14 gives an indication of the average utilization and the corresponding power that the monitor admin and the production monitor operated upon, compared to their maximum case for the respective day. It is obvious that both software components operated at highly inefficient utilization regions. The production monitor was idle most of the time based on its RIC values for that day, whereas the monitor admin, although active most of the period, operated at a very low and thus very inefficient utilization spectrum.

At a first glance the values of the PG indicator may seem contradictory to those of PRO however the low utilization of software components is attributed to their active consumption. Thus if a component operates at utilizations in the 0-1% (approximately 160 watts) region for instance, this consumption is counted as active and thus it is not removed as idle for the calculation of the PRO indicator.

Indicative optimizations with respect to the PG indicators would be to host components at hardware that exhibits a lower idle power consumption, or virtualize and host components on the same resource so they exhibit a lower PG indicator combined. This should be done, by taking under consideration their capacity indicators in order to identify the best suitable deployment.

In general, discovering the most suitable utilization level to operate is not trivial, as higher utilization equals higher efficiency. However, aiming for the highest utilization possible might not be a feasible goal in many cases. Thus, this metric can be used in or-

	Average Utilization and Power consumption	Maximum Utilization and Power consumption
Production Monitor	0,39% util / 160,234 Watts	72,45% util / 203.47 Watts
Monitor Admin	9,36% util / 165,616 Watts	97,76% util / 218.656 Watts

Table 14: Average usage, Peak usage and corresponding Power consumption of the components for 10-07-2013

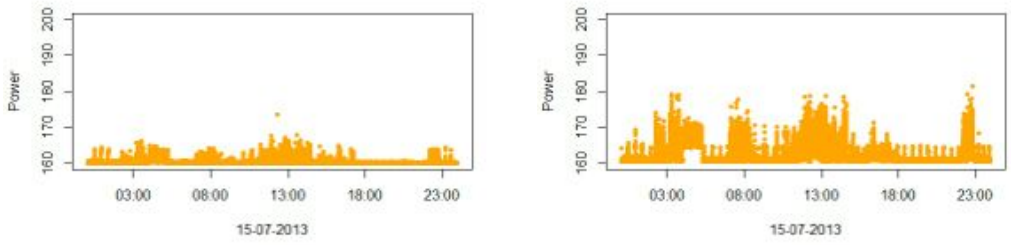
der to identify the utilization level that signals a reduction in PG – (*the "knee" in the curve* (*Figure 9*) –, which can constitute a more feasible optimization goal. Consequently, evaluators can aim at increasing the average resource utilization of inefficient components at least at that point before aiming for higher utilization levels.

8.1.8 *[OPO] Operational Overhead*

When calculating OPO workload as well as system or component type is factored out, therefore OPO values can be used for direct comparisons.

From the results that were obtained by the case study of Chapter 7, OPO (Table 10) demonstrates that the majority of host resources are almost dedicated in serving software component workloads. This is natural since the host resources are only used to facilitate software component work and thus the energy overhead for facilitating component production is minimal for the SIG systems.

To some extent however this is due to the nature of the indicator. Since during the calculation of the ACC for instance, idle software components are attributed 160 watts by default and since the dynamic range of the power consumption model is relatively small, it is expected that the difference between hardware and software component energy consumption will not be significantly large. For instance, if a software component idles (160 watt) but the hardware is executing external workload at 50% utilization (190 watts) the OPO value will be 84%. Thus, either a more proper calculation or more suitable aggregation values are needed in order to accurately quantify the energy overhead of hosting the respective software components.



(a) Power drawn by Acceptance Database component

(b) Power drawn by host resource

Figure 19: Acceptance Database and host resource power usage for 15-07-2013

In general, by looking at the software component OPO values evaluators can identify hardware resources that provision more than what their respective hosted software components need. Optimizations with respect to the OPO value mainly focus on areas that are

external to the application *per se*. Therefore feasible optimizations would be to reduce the number of external components that share hardware resources with application components, to replace *"necessary"* software components with others that are more efficient, or even substitute hardware resources with alternatives that will cause a smaller energy excess.

Finally, OPO can act as a reference value that can be used to assist application stakeholders, in monitoring the effectiveness of any change that is made within a given system or even a hardware resource hosting an application component, regarding energy efficiency.

8.2 EVALUATION

The modeling effort that was presented in this thesis, will be evaluated based on the goal that was set for this research (Section 2.1) and the requirements that where set for the model (Section 6.1) as it was also stated in Section 4.4 .

8.2.1 *Evaluation based on the Research Goal*

With respect to the main goal of the research – *Create a practical model that will be able to help stakeholders and SIG consultants to evaluate the energy efficiency of a software system* – we consider that it was fulfilled despite the lack of validation. The model seems able to identify sources of energy waste and point to optimization opportunities and therefore an assessment regarding the energy footprint of the system is produced. Nevertheless, all the thresholds that were set for the indicators as well as the aggregation method where indicative since we did not have a benchmark to derive them in a statistically sound way. Therefore, only a justification regarding the thresholds as well as indicative ways on how to act upon the values that are derived by the metrics and the model are given.

The reasons that we believe that the model can partially act as first step in assessing the energy efficiency of modern software systems are discussed next.

First, the model managed to identify parts of the SIG system where energy waste occurred. For instance, the RIC metric clearly indicates that the production database is a major source of energy waste, as this component spends only a small portion of the energy it expends for production of work, whereas the PG indicator clearly identifies that all the software components of the system operate in very low and inefficient utilization regions.

Second, the results that the model produced at least at a software component level, can point out to clear optimization opportunities. For example the CCUW metric indicates that weekends are a very inefficient period for the acceptance monitor, production monitor and production database components. This is because the units of work that are delivered by those components during that period, do not justify their energy consumption compared to their CCUW values on weekdays. Consequently, a clear optimization opportunity with respect to this indicator, is to decrease the energy consumption those components during the weekends.

Nevertheless, not all the indicators seem to point out to clear optimization opportunities. For instance, OPO does not provide any significant lead on how to act upon improving its value at least at this point. Consequently, more experimentation is needed to understand the strengths and limitations of the metrics.

A third reason that makes us believe that this modeling effort is promising, is that optimization activities can be partially prioritized at least at a software component level. Ideally, if the aggregation method that was presented was based on a formal statistical analysis, potential evaluators should focus on those sub-characteristics of energy efficiency that presented the lowest rank at a system level. Since this isn't possible, prioritization of optimization activities can be achieved on a per metric basis. For instance, evaluators can identify the software component that presents the lowest score on a respective metric and focus their optimization efforts there.

Overall, the results that we obtained by the case study indicate that this modeling effort has potential, although significantly more evaluations that will probably lead to further refinements are needed.

8.2.2 *Evaluation based on the Model Requirements*

Based on the requirements that where set in Section 6.1 the following evaluation is made regarding the modeling effort that was presented.

Practical

On the one hand, the model is relatively small as it is consisted only of nine metrics which are fairly easy to calculate once the relevant data is available. Furthermore, we consider that the input that is required by this model is relatively minimal, as in order to apply the model an evaluator only requires:

- An architectural overview of the system.
- Utilization data of software components and host resources.
- Unit of work information.
- Power consumption information.

Based on the research that was conducted, we considered that all of the above can be considered a fairly minimal input in order to produce an accurate evaluation regarding the energy efficiency of a software system.

On the other hand, during the case study the following observations where made that might impose limitations to the practicality of the model:

- Calculating utilizations from the data that is generated by the tracking script that was used needs a considerable amount of time. In retrospective, this script can be modified to perform some of the parsing or computations on the host on which it executes, so that utilization information can be derived with a smaller overhead once the data is obtained. In that case the additional utilization that the script will generate should cause minimal interference with the system.
- Creating an accurate power model for every hardware resource of a system might impose a significant amount of overhead depending on the system type and scale. Nevertheless, acquiring direct energy measurements might be a more impractical goal since it requires a power meter on every resource and additional time to process this type of data. Therefore, we consider that power modeling holds some merit, as indicative power models can be created for similar classes of machines and then used in retrospective by future evaluations.

- Evaluating the energy efficiency of a system should be done `in situ` since energy efficiency is determined by the combination of specific software, hardware and real time workload. This creates implications regarding future benchmarking efforts as it hinders an evaluator of replicating a system in the lab in order to apply the model with ease. Consequently, this method can be considerably difficult to benchmark, as a significant amount of evaluations of systems *in situ* are needed to derive accurate thresholds and a proper aggregation methodology. Nevertheless, we believe that evaluations that are not done *in situ* do not have much merit when assessing the energy-efficiency levels of a software system.

Holistic

The model is holistic since it provides an evaluation regarding the energy efficiency levels of a complete system. This is accomplished by relating the energy consumption of hardware resources to the activity levels of the major software components of the system. Consequently, at a software component level, holistic evaluations are provided as both these layers as well as the intermediate ones are assessed as a whole.

Nevertheless, a concrete answer at a system level cannot be given as the aggregation method that was followed is indicative. However, since all the major system components are assessed to some degree, the model seems able to provide visibility at the system level which is the level of architectural design and of management decisions that have the most influence on the majority of modern software systems. Consequently, evaluations and in turn the optimization opportunities that these evaluations point to, will have a biggest impact at that level. If for instance, the two databases of the `SIG` system are hosted on the same hardware resource, since they use their respective hardware resources at a minimal level, it is expected that the energy savings from this move will be marginally higher than if simply the `CPUs` of the hardware resources that host those components where replaced by more efficient ones.

Actionable

The model is actionable since it can:

- Identify parts of the system that are not energy-efficient.
- Discover optimization opportunities.
- Prioritize – to some degree – optimization activities.

For the purpose of assessing how actionable this model is, the same rationale as presented in Section 8.2.1 applies.

Comparable Results:

A concrete answer regarding this requirement cannot be given since only 1 system was evaluated.

Initially, all model indicators seem relevant to some degree to all modern software systems since they are called to assess properties that apply to all of them (e. g.how much those components idle or how close to their best case they operate). At least at a software component level, heterogeneous components (e. g.a computationally intensive application and a database for instance) where evaluated during the case study that was conducted and all the metrics seem to produce meaningful and relevant results among

that particular component set. Nevertheless, the comparability of the results still remains vague to some extent.

On the one hand, some of the indicators that where used are immediately comparable between every kind of software component or system since they are relativistic. For instance, since the CNS indicator factors out the workload as well as the software component type, it can be used for making direct comparisons.

On the other hand, some of the indicators are not immediately comparable and therefore a significant amount of further evaluations are needed in order to derive threshold values through statistical analysis that will enable direct comparisons. Consequently, it might be necessary to benchmark some indicators based on the component and system types. For instance, the CCUW metric is expected to produce marginally higher values based on the functional complexity of the work unit and probably this indicator needs a more elaborate thresholds and aggregation methodology. Nevertheless, deriving concrete threshold values for the indicators as well as the aggregation methodology seems feasible, although it was know before hand that it would be out of scope.

Understandability

We consider that the model is relatively easy to understand since most of the indicators are fairly simple and straightforward to use and interpret. For instance, the metrics that are used to assess the energy behavior sub-characteristic do not require some elaborate explanation to understand and communicate. Therefore, ACC simply points out which software component is the main energy contributor of the system, RIC identifies how much of the energy contribution of each component is wasted due to the idle state and CCUW relates the energy consumption to production by simply demonstrating how much energy on average each unit of work requires. Nevertheless since we could not measure the Isolation property we cannot be sure regarding the understandability of any potential metric that will be used in that case.

The aggregation method that was used was indicative, therefore we envision an equally understandable top-down approach for interpreting model results. A potential evaluator should be able to navigate from a problematic sub-characteristic at the system level, to the lower ranked indicator of this sub-characteristic and finally to the component that scored lower on that indicator.

Internal View

The model as well as the indicators that are used are impartial regarding any external requirements. For instance, as it was demonstrated in the results section, any availability requirements that the system should exhibit (the production monitor and database should be operational 24/7) were not taken under account when calculating metric and model results. Those requirements are only used to contextualize evaluation results, identify a suitable optimization path and make the necessary trade-offs between energy efficiency and other system qualities.

8.3 THREATS TO VALIDITY

We have identified the following sources of problems in our research.

CONCEPTUAL DIVISION. The conceptual division of energy efficiency that we presented is based upon the literature survey and since there is currently no such division of this quality, some aspects may have been left unidentified.

METRICS. Most of the metrics are contributions of this research, hence more experimentation is needed to understand their strength and limitations.

THRESHOLDS AND AGGREGATION. Since we didn't have a large data set at our disposal, the thresholds that where set for those metrics where based on the literature survey or on the elaboration that was presented in Section 6.3 and as such they where not statistically derived. The aggregation methodology that was used suffers from the same limitation.

VALIDATION. A quantitative validation for this model was infeasible within the scope of this project for the reasons described in Section 4.4. As such the model is not validated.

POWER ESTIMATION. The case study is also subjected to a variety of threats. First and foremost, no actual energy measurements took place due to time and resource constraints, but also because a production system *in situ* was measured and thus any external interference should be minimal. Therefore, a linear power model was used for calculating energy consumption. This creates some implications with the metric values as in the majority of cases, consumption does not scale linearly with utilization as can be seen in Figure 9. Based on this figure we used the linear case which deviates from the actual. If different linear models where used (different Pmax and Pidle values), it is expected that some metrics especially the volume based ones such as the `ACC` might present high deviations. On the positive side of this convention, using the same linear model for all hardware resources limits to some extend the impact that different hardware configurations might have on the experiment and thus it allows us to gain a more clear (although to some extend distorted) visibility to software component energy consumption.

IDENTICAL POWER MODELS. Another convention that was made is that all the software components of the case study operated on hardware resources that shared the same characteristics and configuration, which is not accurate.

VIRTUALIZATION. Another convention that was made is that although when the scripts for the experiment where developed and tested the software components of the `SIG` systems used dedicated host hardware resources, at some point a migration occurred and they got virtualized. Our data collection methodology captures multiple instances of the same component running on a specific hardware resource but it does not track components if any instance of them spawns at hardware resource, other than the one that we track the specific component on. In retrospective the data gathering methodology could be modified to take under account those cases since every software component can be tracked on every hardware resource that belongs to the resource pool. However, due to time constraints we weren't able to modify our data gathering methodology.

DATA COLLECTION AND ANALYSIS. Other threats to validity originate from the data collection and analysis methodology that was used. First, since a script is needed to extract data from hardware resources, it inevitably creates some minimum implications regarding the utilization of those results as it generates some minimal utilization over-

head by itself. Since it is impossible to measure the utilization of the resource without generating some utilization this was considered acceptable.

Second, since all the scripts that where used during the data collection and analysis where custom made some interference with the results is expected. For instance, there were some implication regarding the sampling, as two times the script seized working when tracking the monitor admin component. This lead to some data-loss for the respective time frames which was inevitably replaced by utilization data from the other days. Fortunately, this incident only occurred for the monitor admin component and as such it did not create more serious implications regarding the rest of the case study. Nevertheless, an experiment should be re-run using well know tools such as Linux Top, and the model can be re-applied to validate the outcome.

UNIT OF WORK. This model tries to assess a system regarding its energy-efficiency levels by assessing the the energy-efficiency of this system's software components that participate in work production. Therefore, a convention that is made when calculating the indicators, is that only one application (or a set of software components such as in the case study) generates useful work and all other systems that might reside in the same hardware configuration are auxiliary to the application under evaluation. However, this might not be entirely accurate as for instance an organization might host a set of components that belong to different web-services inside the same hardware resources, all of which generate some equally important useful work from the perspective of the organization. Nevertheless, in those cases, the model can still be adapted to assess the overall efficiency of the system by making the correct aggregations. For instance from components, to web-services, to sub-systems, to the final system.

9 CONCLUSION

This chapter states the conclusions drawn from the results of the experiment and proposes future work in order to further improve the model.

9.1 CONCLUSION

In this thesis a conceptual division, a model and a set of metrics that aim at evaluating the energy efficiency of software systems was proposed. This model was applied to the systems of the Software Improvement Group with promising results, as it identified clear energy wastes and optimization opportunities. Nevertheless, significantly more evaluations are needed to understand the strengths and limitations of our modeling approach.

As it was stated in Chapter 4, it was known beforehand that a sufficient validation regarding the model and the underlying methodology was not feasible. Furthermore, there was no proper data-set available and the duration of this research was relatively short in order to evaluate more than one system *in situ*. Consequently, the model that we presented needs more validation.

Answer to sub-research question 1

We chose to base our conceptual division on the corresponding division of performance efficiency from `ISO/IEC 25010` [44] for the reasons that are presented in Section 5.1. Consequently, with respect to the first sub-question of this research – **What are the aspects that determine the energy efficiency levels of a software system?** – (Section 2.4) an answer was given at some extent. Of course this question delves in the conceptual layer of energy efficiency and thus some aspects may have been left unidentified. Other aspects of energy efficiency (e. g.environmental) were out of scope and where implicitly ignored with respect to the conceptual division that we propose. Nevertheless, since this division seems sufficient to act as a basis for the development of a practical model, at least at the state that the current research on the topic is, a satisfactory answer was given.

Answer to sub-research question 2

A difficulty that this research had to surmount was that many of the current metrics regarding energy efficiency aim at assessing efficiency aspects at lower system levels and hence they where not suitable for our purpose. Consequently, half of the metrics that were used by this research had to be developed by us and although simple in nature, they had to be justified to some degree. Therefore, more experimentation is needed in order to identify their strengths, limitations and their suitability across a wide spectrum of different system types.

Nevertheless, most of the metrics seem to serve their purpose at least in the context of the case study, thus they can be used to quantify the sub-characteristics of energy efficiency based on the conceptual division that was created. The evaluation of the metrics was promising since clear energy wastes were identified, however a significant larger amount of evaluation is needed in order to come to conclusions with a high degree of

certainty since the thresholds that where used where not calibrated. Consequently, with respect to the second sub-question that was formulated by this research – **How can these aspects be quantified in a meaningful way?** – (Section 2.4) we partially gave an answer as we demonstrated that the results derived by those metrics can indeed at some level evaluate the sub-characteristics of energy efficiency.

Answer to sub-research question 3

With respect to the third sub-question that was raised by this research – **How is it possible to combine and aggregate evaluation results so that they can produce actionable advice** – (Section 2.4) only a partial answer was given. The aggregation was indicative and as such it holds no statistical value. On the other hand, as it was demonstrated in Section 8.1, the results that this indicators produced, seem to answer all the question that were set in the `GQM` approach. Thus they succeed in fulfilling their goal which is to assess the sub-characteristics of energy efficiency. Consequently, a combined evaluation is indeed produced by the model. Furthermore, since clear inefficiencies where identified during the evaluation of the `SIG` systems, this model seems capable of being used for evaluations as it can produce actionable advice.

Answer to the Research Question

With respect to our main research question – **How to develop a practical model that will have the ability to evaluate the energy efficiency of software systems?** – (Section 2.4), a conclusive answer cannot be given. Although the model reveals inefficiencies, it is not able to produce a validated, absolute number regarding the energy efficiency levels that a system exhibits. Nevertheless, such number was not expected from the beginning of the research. We believe, that the fact that this model is able to identify spots in a system where energy waste occurs and reveal clear optimization opportunities is a considerable achievement and a satisfactory answer to our research question.

Lessons Learned

One of the lessons learned regarding the practicality of the model, is that even though it required a fairly small amount of inputs, deriving the necessary utilization statistics introduced a significant amount of overhead. We consider that more extensive evaluation frameworks can potentially create serious implications regarding the applicability of an evaluation based on this observation. Therefore, we believe that practical solutions are more promising.

A second lesson that can be drawn is that although the model seems suitable for smaller scale systems, it might possibly need further refinement in order for it to be equally applicable to large-scale systems. In an ideal case an automated monitoring solution that would perform the necessary computations and derive real time metric results *in situ* could alleviate that problem, although we cannot be entirely sure regarding the feasibility of this effort.

Another observation that can be made based on the above, is that there seems to be clash between the practicality and the actionability of the model. We consider that the model requires a minimal amount of input in order to be actionable and practical. Nevertheless, as the system under evaluation grows in scale in order to preserve the actionability of the model, its practicality inevitably decreases as volume of inputs rises. The opposite also applies, as in order to preserve the practicality of the model it might be necessary, that a simpler approach be followed which would inevitably limit the

actionability of the model. Consequently, the sliver line between those two qualities might be entirely dependent on system scale.

Another lesson that can be drawn, is that energy efficiency is still a very undeveloped topic and from a software perspective its even more immature as the ICT field only recently started to admit and realize the undeniable impact that source code execution has on the energy consumption of hardware resources. We believe that a centralized and widely accepted conceptual division in the form of an ISO for instance, can help significantly, as it can act as a solid basis that future research can more accurately build upon in order to be justified to some degree.

From this thesis we can draw that a model to measure the energy efficiency of software systems is possible, although validating that the model actually assesses energy efficiency remains an unsolved challenge. Therefore, we can only conclude that this model assesses the overall energy footprint of a system in an effort to identify energy wastes and produce actionable advice in order to eliminate them. Based on the results that where derived from the case study, we believe that this approach holds merit. Nevertheless, significant refinements and improvements are needed which are stated in the next section.

We expect that energy efficiency will constitute a major design consideration in the imminent future and thus practitioners in the field should start treating this quality to a level that is appropriate to the implications that it can potentially create.

9.2 FUTURE WORK

Since this model is a first effort in quantifying the energy efficiency levels of software systems in the form of a practical model, there are are lots of opportunities for future work.

VALIDATION/EVALUATION. The first and most important is the validation of the model, as in the context of this thesis it was applied only on one system (Chapter 7) and therefore further evaluation is necessary. This can be done by applying the model to a wider set of systems in order to gain more experience regarding its structure and the metrics that where used. This will also help in identifying how suitable the model is to evaluate other system types. A wide amount of evaluations will probably reveal hidden limitations that will lead to model refinement both at a conceptual (energy efficiency conceptual division and metrics used) as well at an operational (measurement methodology and data analysis) level.

ACCEPTANCE THRESHOLDS/AGGREGATION. The collection of a large number of evaluations and energy profiles through model application will also pave the way in establishing acceptance thresholds for the metrics and will lead to a more suitable aggregation method. This fact will enable more concrete and accurate comparison among systems from different functional domains. Therefore, one avenue of further work is model benchmarking.

ENERGY MEASUREMENTS. Furthermore, since in the case study (Chapter 7) that was conducted no direct energy measurements took place, future research should explore a feasible way to obtain accurate energy measurements with a reasonably low overhead.

ISOLATION. Since this research did not manage to measure the Isolation property (Equation 6.2.3.1), investigating the impact that external systems might have on the energy footprint of an application is important. This can be done by identifying metrics that will be able to measure such property in a feasible way without violating the requirements that where set for the model (Section 6.1). Nevertheless, application components can also affect the energy footprint of one-another considerably (an application component might cause others to be at an `Wait` state for instance) and thus an internal type of isolation – *how much the energy footprint of an application component is affected by other internal to the application elements* – might be an interesting path for future work.

METRIC PERSPECTIVE. Another promising avenue for future research would be, to investigate `Resource Utilization` and `Capacity` sub-characteristics, by evaluating the proportionality of the application as well as the limits of the host infrastructure through different perspectives. This could be done for instance, by examining proportionality through evaluating the scalability of the system when data increases compared to its scalability when the number of transactions rises.

DISTRIBUTED COMMUNICATIONS. Another possible course for future work, would be to investigate metrics targeted at assessing the energy impact of distributed communications caused or targeted to application elements as they can play an important role in determining energy costs [55]. More specifically, this type of consumption can be heuristically measured by the number of bytes transmitted through the network and the type of communication means (optical, fabric cable etc.), when the application uses communication channels. These, can be either internal (e. g.Lan) or external (e. g.Internet) to the system. In the latter case, since it might be difficult to acquire accurate data regarding the energy characteristics of the communication channel, estimates such as the ones described in [85] can be used.

REDUNDANCY. Another intriguing domain of research could be exploring the relationship between energy efficiency and redundancy. Redundancy is inevitable due to the way modern software systems are deployed and operate – since it is crucial to sustain high levels in other system qualities – but it can also create additional energy costs. Therefore, an avenue of future work would be to incorporate measurements that target to assess the impact that any form of duplication (e. g.data, computational) might have on the energy footprint of an application.

LOWER LEVEL MEASUREMENTS. In the context of this model, an application is broken down to software components and all assessments take place at that level, by associating hardware resource energy consumption to them. A similar division can be followed for the hardware resources that are used by those components, by dividing them into their internal components (CPU, memory, disks etc.) and associating energy consumption at that level to the application.

Following this approach would enable a more low-level assessment of the energy footprint of software components as it would enable the correlation of their energy consumption to that of low level hardware components individually. For instance the Annual Component Consumption (`ACC`) (Section 6.2.1.1) could be calculated as the annual energy consumption of the application component caused by the host resource's CPU, memory, disks etc. revealing in that sense hidden optimization opportunities. This would also pave the way to extending the model with measurements targeted at *lower level* system events such as Cache miss ratio, no. of Page Faults per sec, Processor in-

terrupts, Context Switches etc. that play a role in the energy footprint of an application [40][89].

ASPECTS OF ENERGY EFFICIENCY. Another route that future researchers could follow, would be to enhance the model with measurements targeted at other aspects of energy consumption and therefore efficiency, such as the heat dissipation of the host platform as well as the cooling energy costs required to counteract those thermal emissions. Moreover, extending the model with other sustainability indicators such as for instance CO2 emissions caused by a system might be feasible. Furthermore, it would be interesting to evaluate energy efficiency not only from an operational perspective, but through a wider prism by assessing other phases of the application life-cycle. Furthermore, weighting energy consumption and model indicators based on the type of energy (e. g.renewable, nonrenewable) that is used to sustain the operations of the application is also an interesting topic of research as the costs between different energy types vary.

OPTIMIZATIONS. Finally, creating a decision tree or table, that will guide evaluators through the optimization process based on the values that are produced by the model indicators, would add significant value to it, since at this point it is focused on the evaluation domain. Creating an optimization path by using the model indicators and an optimization decision tree could be the most suitable step to extent this research.

Of course, it is rather trivial if all of the above can be incorporated in the model in a feasible way or if they can be applied without violating some of the model requirements (Section 6.1) as some of them are expected to increase the overhead of applying the model and were not pursued mainly for that reason and due to time constraints. Therefore, it is crucial to investigate the preceding areas without limiting the applicability of the model. After all, one of the most important requirements set for this research was for the model to be practical and for that reason it should be easily applicable with a minimal overhead and with the least possible amount of input data.

A

SCRIPT IMPLEMENTATION DETAILS

A.1 DATA COLLECTION SCRIPT

Gathering the necessary CPU data is achieved by periodically copying the file residing in the /proc/stat directory of the resource. This file keeps track of a variety of different system statistics since its last boot. The proc file system is used as an interface to kernel data structures and is commonly mounted at /proc.

In order to derive process statistics a slight more elaborate methodology is used. The main difficulty that requires a more complex approach, is that the processes are assigned with a process identifier (PID) every time that they are created. This means that every time the application component is restarted, its main process is dynamically assigned a new PID. Also, every time a certain sub-process is invoked it is also assigned a new PID. Therefore, it is not viable to follow the same methodology as the one that was followed for the system CPU and simply copy the /proc/<PID>/stat directory of the system for certain predetermined PIDs. Consequently, a certain tracking mechanism should be developed to track all the processes and sub-processes that originate from the respective application components.

This is achieved by periodically querying the process tree (pstree) of the system for processes that originate from a certain User (application component). The script therefore, queries the process tree, tracks certain processes that are generated by the software component by name, checks their PID value and then copies the respective /proc/<PID>/stat file to the output folder. This is done for all the processes of the component as well as any sub-process that might be created during the course of the seven day measurement period.

A.2 UTILIZATION SCRIPT

The script parses the folder and distinguishes between CPU and application component files based on their filename. For the CPU files only the first row of the file is needed in order to derive utilization results. This first row holds information about the amount of time, measured in units of USER_HZ(1/100ths) of a second on most architectures, that the system spent in various states. Thus, the first row is stored from every file and the right values are parsed. This row is then summed. Value 4 of this row represents the idle time of the CPU. Consequently, the following equation can be used in order to derive the CPU utilization of the host resource for a second n.

$$CPU_{util} = \frac{((Sum.row_n - Sum.row_{n\text{-}1}) - (Value.4_n - Value.4_{n\text{-}1})}{Sum.row_n - Sum.row_{n\text{-}1}} \times 100 \qquad (18)$$

By parsing all the CPU files and using the above equation the per second utilization for every host resource is obtained.

Parsing the files that are generated by the processes that originate from this application components is again a bit more elaborate since every second more than one files might be generated by the component based on the processes and sub-processes that it creates. Thus, the script produces the application component utilization by aggregating the statistics that are derived by all its process and sub-process files for every second.

These files hold status information about the respective process or sub-process and only two values (14 and 15) from the 44 total are needed to correctly derive software component utilization. Value 14 holds information regarding the amount of time that this process has been scheduled in user mode, measured in clock ticks and Value 15 holds information regarding the amount of time that this process has been scheduled in kernel mode, also measured in clock ticks. Consequently, all the Values 14 and 15 are for the processes and sub-processes of a component are summed up for every second and are used by the following equation to derive the component utilization for a second n:

$$Soft.Component_{util} = \frac{((Sum.Values.14_n + Sum.Values.15_n) - (Sum.Values.14_{n-1} + Sum.Values.15_{n-1})}{Sum.row_n - Sum.row_{n-1}} \times 100 \quad (19)$$

The $Sum.row_n$ is the the same as described in Equation 18.

Thus for every second, the utilization of all processes and sub-processes is calculated and the results are aggregated into an overall software component utilization. The preceding analysis produces the `CPU` and the software component utilization per second for the entire measurement period.

A.3 CORRELATION SCRIPT

This script is used to correlate the per second power consumption of the software components with the information about the units of work derived by the respective log files. Thus, the correlation script, is also used to align unit of work and respective power consumption data sets. This is done by correlating the starting timestamp of the unit of work table with corresponding timestamp of the power consumption table. After that and until the ending timestamp of the work unit the power drawn by this unit of work is calculated. This is done for every second of the work unit duration. Since two work units might exist in parallel, for instance for the `Monitor` Sub-system, two clients might be using the system simultaneously, the power should be fairly attributed between them. Thus if two units of work share the same timestamp, the power of the application component is divided and attributed equally to them. The same methodology is used if more than two units of work happen simultaneously. The sum of the per second power consumption of a work unit produces the energy consumption of that work unit and thus the energy consumption for all the units of work delivered by the respective software component is calculated.

B

EFFICIENCY AND OTHER QUALITY ASPECTS

Attaining higher levels of energy efficiency is becoming an increasingly pressing matter for modern organization. However, this quality conflicts with various other desired system properties and trade-offs need to be made in order order to achieve high efficiency levels. Some of the most apparent candidates that are expected to clash with energy-efficiency are quality aspects such as performance, availability, usability etc.[74]. Nevertheless, in many cases the relationship between efficiency and those quality attributes is highly complex and not easily understood.

A straightforward way to interpret this association is by examining which of those qualities depend on any kind of redundancy mechanism (data, computational, hardware resource etc.) in order to be promoted. By definition, redundancy clashes with the overall sense of economy that stems from efficiency, therefore is contradictory to the main efficiency goal which is: finding the cheapest optimal solution that is needed to achieve energy efficiency. Therefore, promoting qualities such as security or availability, that usually require some levels of duplication –either computational or data – is expected to damage at least to some extend the overall energy footprint of the system.

Thus, any resource (hardware or software) that increases energy consumption but does not contribute to the main useful work that the system delivers, or is not crucial for system operations can be removed to enhance efficiency levels.

Even though resource consolidation is more easily perceived conceptually in the hardware resource domain, discarding software components in order to increase efficiency levels is also a viable choice. Software that does not contribute to system operations or is not an integral part of the system, will inevitably bind system resources, increasing energy consumption in return.

Furthermore, as previously described, not all quality aspects are enhanced through careful and systematic planning and hence, many of them have long-gone been promoted in an energy agnostic way. Therefore, many of the trade-offs that have to be made still remain trivial.

`McCall et al.` in [58], presents a brief elaboration on the trade-offs between a general efficiency quality aspects and other quality aspects. We believe that those trade-offs also apply to some extend in the more narrowly scoped energy efficiency domain. Therefore, they are presented in conjunction with well known tactics that are used to attain them according to `Bass et al.` [8], in order to provide help to practitioners that are called to interpret evaluation results but also to increase the clarity with which those are presented and conveyed:

Security, Integrity, Non-repudiation, Accountability and Authenticity vs Efficiency

`McCall et al.` [58]:*"The additional code and processing required to control the access of the software or data usually lengthens run time and require additional storage".*

Usually and in order to achieve higher levels of Security (Integrity is a sub-characteristic of security according to ISO) additional processing or storage resources are used.

An example of those is maintaining and operating an intrusion detection system. Such systems function by persistently comparing network traffic patterns to historic ones of

known attacks that are maintained in a database [8]. The extra processing (comparisons) and storage (database) by definition require additional energy to operate.

Maintaining audit trails (transaction copies) in order to enhance Non-repudiation and Accountability is another commonly used security technique that produces supplementary energy overhead. The same also applies for more widely and frequently used techniques that aim at augmenting security levels, such as maintaining redundant copies of administrative data like passwords and user records.

Finally, techniques that are used for reinstating the system to a consistent state after an attack, usually require additional processing and storage capacity and are further discussed in later in the Availability section, as they fall under the jurisdiction of Recoverability.

Usability, Learnability, Operability, User Error Protection and User Interface aesthetics vs Efficiency

`McCall et al.` [58]: *"The additional code and processing required to ease an operator's task or provide more usable output usually lengthen run time and require additional storage".*

The above is also evident, in the context of energy efficiency, if one considers the mechanisms which are used in order to amplify usability levels. According to [8] there are three distinct techniques that are used to promote usability:

- Maintaining a model of the task, in which the task model is stored and used by the system in order to provide some level of assistance to the user. e. g.Auto-correction in text editors.
- Maintaining a model of the user, in which specific user properties, such as system knowledge is stored, to enable predictions about expected user behavior. e. g.Pace scrolling speed.
- Maintain a model of the system, in which the system maintains information about its state and expected behavior in order to provide appropriate feedback to the user. e. g.Predicting and displaying the time needed to finish a file transfer.

All of the above significantly increase usability levels through binding additional processing and storage resources, escalating in extend the energy costs of the system.

Additionally, in most cases these mechanisms are paired with other frivolous features in order to provide higher aesthetics to the system. Consider for instance, the commonly know flying pages when one copies files on his personal computer. Clippy which was used at the MS Office is another example, of a superficial feature used towards increasing usability.

Moreover, and in many cases providing better aesthetics is not done to promote any of the above mechanisms. User Interface aesthetics is a sub-characteristic of Usability according to the ISO and is defined as: *"degree to which a user interface enables pleasing and satisfying interaction for the user."*

In order to increase the degree of this property *"expensive"* features are added to the system such as various forms of multimedia and high resolution images, while cheaper forms of the same formats might have worked without any-notable difference or aesthetic degradation. Consider the case, in which a `JPEG` file of 1900x1200 resolution is used by a web-site to depict a company logo. The same aesthetic results however could have been probably achieved by a lower resolution (more energy efficient) image without any notable impact to the end-user perception.

At a first glance reducing image resolution might not seem like a practice capable of increasing-efficiency levels at a satisfactory degree. However, examples as the above make a lot more sense when they are not considered in isolation. When the above image is placed at a popular web-site the energy waste is multiplied by the number of web-site visits. It is evident that the energy waste can scale to significant numbers depending on the system context and type.

Maintainability and Testability vs Efficiency

`McCall et al.` [58]:*"Optimized code, incorporating intricate coding techniques and direct code, always provides problems to the maintainer. Using modularity, instrumentation, and well commented high level code to increase the maintainability of a system usually increases the overhead resulting in less efficient operation".*

Optimization routines are a common method for enhancing energy efficiency levels. Usually, performance related optimizations and libraries are used in order to reduce execution time and therefore decrease energy consumption. However, they also induce increased complexity which inevitably has a negative impact on maintainability.

Testability on the other hand requires the development of test cases, which can easily scale in number and volume and which usually require additional storage capacity in order to be kept and reused in latter development stages as the process evolves in time. The increased amount of source code by each-self as well as the task of testing the system against those test cases at a regular basis, inevitably imposes an additional energy burden.

Flexibility, Re-usability, Modularity and Modifiability vs Efficiency

`McCall et al.` [58]: *"The generality required for a flexible system increases overhead and decreases the efficiency of the system".*

We as software engineers have learned not to reinvent the wheel. One of the first things we have been tough is to abstract and create general, modular, re-usable solutions whenever and wherever possible in order for them to be easily usable by future and possibly entirely heterogeneous software products, or be conveniently modified in existing ones, if deemed necessary. This fundamental way in the way we develop and think, creates many implications in the context of energy efficiency as principal techniques that are commonly used to promote the preceding qualities, such as abstractions.

According to [74] the main reason for this phenomenon is that abstraction layers, or source-code constructs, such as deep inheritance in Object Oriented languages, that are used to promote the aforementioned qualities, usually make program execution less efficient from a performance and energy viewpoint, *"as calls have to be dynamically resolved and routed through these abstraction layers and additional library and framework code needs to be executed resulting in energy waste."*

Creating re-usable and easily modifiable solutions through modularization, information hiding and encapsulation, inevitably involves some level of generality. Nevertheless, general purpose solutions that are developed to be used in a broad context far deviate from the specific tailored made implementations that are required in order to attain high levels of efficiency (consider optimizations discussed in the previous section, or the extend at which libraries-which are by definition designed to serve a wide context-are used in modern software development).

A more thorough discussion as to how generality affects efficiency and in extend energy efficiency can be found in [74][11][33].

Consequently, dealing with the energy efficiency problem might require a drastic and considerable shift in the way we think about, plan and develop software, both at an individual but also on a higher level of the Software Engineering stack.

Portability and Interoperability vs Efficiency

McCall et al. [58]: *"The added overhead for conversion from standard protocol and standard data representations, and the use of interface routines decreases the operating efficiency of the system".*

The same rationale also applies in this case as a higher processing demand equals additional energy cost.

Reliability, Availability, Fault tolerance and Recoverability vs Efficiency

According to Bass et al. [8] in order to attain high levels of Availability – and in extend Reliability, Fault tolerance and Recoverability – redundancy mechanisms are commonly deployed. However, in order to achieve satisfactory level of the preceding qualities through redundancy implies a contemporary need for synchronization (to ensure that the redundant copy can be used if the original fails).

Furthermore, redundancy is often used in the form of various structured mechanisms such as voting to enhance Fault Recovery and Reliability levels. According to this technique as described by Bass et al. [8] Redundant processes run on separate processors, taking equivalent input and computing an output which is sent to the voter. This method is used in order to detect and correct deviating operations of algorithms or processor failures.

Furthermore, in distributed system, redundancy may be deployed on the network, through redundant components residing in parallel communication paths. Synchronizing duplicate components, thus is achieved by sending similar sets of messages to each and everyone.

According to Barosso et al. [6] in order to increase the levels of that quality in the data center domain, data and computations are usually spread across multiple hardware nodes. This technique also enhances Fault tolerance and Recoverability levels, however it implies that hardware resources should be available at all times, even during lower load periods, and therefore operate at lower efficiency regions.

From the above it is clear, that redundancy in order to attain higher levels of the preceding qualities has a substantial impact on the energy footprint of a software system. On the processing domain, executing redundant computations on different processors increases energy consumption due to parallel and duplicate processing, while the constant need for synchronization imposes an apparent burden in networking costs and energy consumption.

In the data domain, redundancy can cause excess use of additional storage, increased time in processing queries, superfluous network usage and also introduces the possibility of inconsistencies between duplicate data as synchronization must be assured between redundant copies. If one considers the ancillary mechanisms which affect the aforementioned factors then the cost of data duplication increases significantly.

For instance, a synchronous mechanism for maintaining duplicate data copies (for the purposes of increasing Recoverability for instance) – one that must assure that data is copied to a second storage device before completing the transaction – will create additional energy consumption, compared to an asynchronous one, because it will increase the time that the main transaction needs in order to be processed. This in extend will

lead to resources being binded for larger periods, resulting in an excess in energy consumption.

Of course redundancy is not the only way of increasing the aforementioned qualities as more *"brute-force"* and inefficient ways are often used depending on the system type and the general operating context. For instance, in order to increase the availability levels of a web service, the host infrastructure might remain operational 24/7, even though demand might be minimal to zero during night hours.

All of the above, are translated into palpable energy costs. Consequently reducing qualities such as availability to a more realistic level, may yield significant benefits in reducing energy costs.

Conclusion

The above conflicts do not constitute an exhaustive list but where presented in order to give a feel to evaluators, that will potentially use this model, about the relationship between energy efficiency and other well known and valued system qualities.

Therefore, the interpretation of the model results and the advises and optimization suggestions made after system evaluation should take under account the level that the preceding qualities should exhibit, evaluate how realistic and pragmatic this level is, in order to make the necessary trade-offs so as to attain high efficiency levels.

BIBLIOGRAPHY

[1] Susanne Albers. Energy-efficient algorithms. *Commun. ACM*, 53(5):86–96, May 2010.

[2] Danilo Ardagna, Cinzia Cappiello, Marco Lovera, Barbara Pernici, and Mara Tanelli. Active energy-aware management of business-process based applications. In *Proceedings of the 1st European Conference on Towards a Service-Based Internet*, ServiceWave '08, pages 183–195, Berlin, Heidelberg, 2008. Springer-Verlag.

[3] M. Armand and J. M. Tarascon. Building better batteries. *Nature*, 451(7179):652–7, February 2008.

[4] Jeroen Arnoldus, Joris Gresnigt, Kay Grosskop, and Joost Visser. Energy-efficiency indicators for e-services. *2nd International Workshop on Green and Sustainable Software*, 2013.

[5] Luiz André Barroso. The price of performance. *Queue*, 3(7):48–53, September 2005. ISSN 1542-7730.

[6] Luiz André Barroso and Urs Hölzle. The case for energy-proportional computing. *Computer*, 40(12):33–37, December 2007.

[7] Victor R. Basili, Gianluigi Caldiera, and H. Dieter Rombach. The goal question metric approach. In *Encyclopedia of Software Engineering*. Wiley, 1994.

[8] Len Bass, Paul Clements, and Rick Kazman. *Software Architecture in Practice*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2 edition, 2003. ISBN 0321154959.

[9] Anton Beloglazov, Rajkumar Buyya, Young Choon Lee, and Albert Y. Zomaya. A taxonomy and survey of energy-efficient data centers and cloud computing systems. *CoRR*, abs/1007.0066, 2010.

[10] Pat Bohrer, Elmootazbellah N. Elnozahy, Tom Keller, Michael Kistler, Charles Lefurgy, Chandler McDowell, and Ram Rajamony. Power aware computing. chapter The case for power management in web servers, pages 261–289. Kluwer Academic Publishers, Norwell, MA, USA, 2002.

[11] D. J. Bradley, R. E. Harper, and S. W. Hunter. Workload- based power management for parallel computer systems. *IBM J. Res. Dev.*, 47(5-6):703–718, September 2003.

[12] David Brooks, Vivek Tiwari, and Margaret Martonosi. Wattch: a framework for architectural-level power analysis and optimizations. *SIGARCH Comput. Archit. News*, 28(2):83–94, May 2000.

[13] David J. Brown and Charles Reams. Toward energy-efficient computing. *Commun. ACM*, 53(3):50–58, March 2010. ISSN 0001-0782.

[14] Eugenio Capra, Chiara Francalanci, and Sandra A. Slaughter. Is software "green"? application development environments and energy efficiency in open source applications. *Inf. Softw. Technol.*, 54(1):60–71, January 2012.

[15] Gong Chen, Wenbo He, Jie Liu, Suman Nath, Leonidas Rigas, Lin Xiao, and Feng Zhao. Energy-aware server provisioning and load dispatching for connection-intensive internet services. In *Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation*, NSDI'08, pages 337–350, Berkeley, CA, USA, 2008. USENIX Association.

[16] Hui Chen, Meina Song, Junde Song, A. Gavrilovska, K. Schwan, and M. Kesavan. Cacm: Current-aware capacity management in consolidated server enclosures. In *Proceedings of the 2011 International Green Computing Conference and Workshops*, IGCC '11, pages 1–6, Washington, DC, USA, 2011. IEEE Computer Society.

[17] Hui Chen, Shinan Wang, and Weisong Shi. Where does the power go in a computer system: Experimental analysis and implications. In *Proceedings of the 2011 International Green Computing Conference and Workshops*, IGCC '11, pages 1–6, Washington, DC, USA, 2011. IEEE Computer Society.

[18] Yan-Wei Chen, Mei-Ling Chiang, and Chieh-Jui Yang. Energy-efficient scheduling based on reducing resource contention for multi-core processors. In Jeng-Shyang Pan, Ching-Nung Yang, and Chia-Chen Lin, editors, *Advances in Intelligent Systems and Applications - Volume 2*, volume 21 of *Smart Innovation, Systems and Technologies*, pages 553–562. Springer Berlin Heidelberg, 2013.

[19] TeamQuest Corporation. Capacity planning, discipline for data center decisions, 2004.

[20] Stephen Dawson-Haggerty, Andrew Krioukov, and David E. Culler. Power optimization - a reality check. Technical report, EECS Department, University of California, Berkeley, Oct 2009.

[21] Dell. Understanding Data Center Energy Intensity. Technical report, 2010.

[22] Thanh Do, Suhib Rawshdeh, and Weisong Shi. pTop: A Process-level Power Profiling Tool. In *HotPower '09: Proceedings of the Workshop on Power Aware Computing and Systems*, New York, NY, USA, October 2009. ACM.

[23] Dimitris Economou, Suzanne Rivoire, and Christos Kozyrakis. Full-system power analysis and modeling for server environments. In *In Workshop on Modeling Benchmarking and Simulation (MOBS*, 2006.

[24] EPA. EPA Report to Congress on Server and Data Center Energy Efficiency. Technical report, U.S. Environmental Protection Agency, 2007.

[25] Xiaobo Fan, Wolf-Dietrich Weber, and Luiz Andre Barroso. Power provisioning for a warehouse-sized computer. *SIGARCH Comput. Archit. News*, 35(2):13–23, June 2007.

[26] Michel Feidt. Energy efficiency and environment. *U.P.B. Scientific Bulletin, Series C*, 72(1), 2010.

[27] Wu-chun Feng. Making a case for efficient supercomputing. *Queue*, 1(7):54–64, October 2003.

[28] Miguel A. Ferreira, Eric Hoekstra, Bo Merkus, Bram Visser, and Joost Visser. Seflab: A lab for measuring software energy footprints. 2013.

[29] Jason 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, pages 2–, Washington, DC, USA, 1999. IEEE Computer Society.

[30] Anshul Gandhi, Mor Harchol-Balter, Rajarshi Das, and Charles Lefurgy. Optimal power allocation in server farms. *SIGMETRICS Perform. Eval. Rev.*, 37(1):157–168, June 2009.

[31] Matthew Garrett. Powering down. *Commun. ACM*, 51(9):42–46, September 2008.

[32] Inigo Goiri, Ferran Julia, Ramon Nou, Josep Ll. Berral, Jordi Guitart, and Jordi Torres. Energy-aware scheduling in virtualized datacenters. In *Proceedings of the 2010 IEEE International Conference on Cluster Computing*, CLUSTER '10, pages 58–67, Washington, DC, USA, 2010. IEEE Computer Society.

[33] Naga Govindaraju, Jim Gray, Ritesh Kumar, and Dinesh Manocha. Gputerasort: high performance graphics co-processor sorting for large database management. In *Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, SIGMOD '06, pages 325–336, New York, NY, USA, 2006. ACM.

[34] The G. Grid. The Green Grid Data Center Compute Efficiency Metric: DCcE. Technical report, 2010.

[35] Kay Grosskop. Pue for end users - are you interested in more than bread toasting? 2012.

[36] Kay Grosskop and Joost Visser. Identification of application-level energy optimizations. 2012.

[37] Kay Grosskop and Joost Visser. Energy efficiency optimization of application software. *Green and Sustainable Computing: Part II, 1st Edition*, 2013.

[38] Jorge Guerra, Wendy Belluomini, Joseph Glider, Karan Gupta, and Himabindu Pucha. Energy proportionality for storage: impact and feasibility. *SIGOPS Oper. Syst. Rev.*, 44(1):35–39, 2010.

[39] P.K. Gupta and G. Singh. User centric framework of power schemes for minimizing energy consumption by computer systems. In *Radar, Communication and Computing (ICRCC), 2012 International Conference on*, pages 48–53, 2012.

[40] Sudhanva Gurumurthi, Anand Sivasubramaniam, Mary Jane Irwin, N. Vijaykrishnan, Mahmut Kandemir, Tao Li, and Lizy Kurian John. Using complete machine simulation for software power estimation: The softwatt approach. In *Proceedings of the 8th International Symposium on High-Performance Computer Architecture*, HPCA '02, pages 141–, Washington, DC, USA, 2002. IEEE Computer Society.

[41] Stavros Harizopoulos, Mehul A. Shah, Justin Meza, and Parthasarathy Ranganathan. Energy efficiency: The new holy grail of data management systems research, 2012.

[42] Taliver Heath, Eduardo Pinheiro, Jerry Hom, Ulrich Kremer, and Ricardo Bianchini. Application transformations for energy and performance-aware device management. In *Proceedings of the 2002 International Conference on Parallel Architectures and Compilation Techniques*, PACT '02, pages 121–130, Washington, DC, USA, 2002. IEEE Computer Society.

[43] Urs Holzle and Luiz Andre Barroso. *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines*. Morgan and Claypool Publishers, 1st edition, 2009.

[44] ISO/IEC. ISO/IEC 25010 - Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - System and software quality models. Technical report, 2010.

[45] Timo Johann, Markus Dick, Stefan Naumann, and Eva Kern. How to measure energy-efficiency of software: Metrics and measurement results. In Rick Kazman, Patricia Lago, Niklaus Meyer, Maurizio Morisio, Hausi A. Muller, Frances Paulisch, Giuseppe Scanniello, and Olaf Zimmermann, editors, *GREENS*, pages 51–54. IEEE, 2012.

[46] Koomey Jonathan. Growth in data center electricity use 2005 to 2010. *CA: Analytics Press*, 2011.

[47] Aman Kansal and Feng Zhao. Fine-grained energy profiling for power-aware application design. *SIGMETRICS Perform. Eval. Rev.*, 36(2):26–31, aug 2008.

[48] Aman Kansal, Feng Zhao, Jie Liu, Nupur Kothari, and Arka A. Bhattacharya. Virtual machine power metering and provisioning. In *Proceedings of the 1st ACM symposium on Cloud computing*, SoCC '10, pages 39–50, New York, NY, USA, 2010. ACM.

[49] Alexander Kipp, Tao Jiang, Mariagrazia Fugini, and Ioan Salomie. Layered green performance indicators. *Future Generation Computer Systems*, 28(2):478 – 489, 2012.

[50] Jonathan G. Koomey. Estimating total power consumption by servers in the U.S. and the world. Technical report, Lawrence Derkley National Laboratory, February 2007.

[51] Belady C. L. In the data center, power and cooling costs more than the it equipment it supports. *Electronics Cooling Magazine*, 13(1):24–27, 2007.

[52] Richard P. Larrick and Kirk W. Cameron. Consumption-based metrics: From autos to it. *IEEE Computer*, 44(7):97–99, 2011.

[53] Petter Larsson. Energy-efficient software guidelines, 2012.

[54] Etienne Le Sueur and Gernot Heiser. Dynamic voltage and frequency scaling: the laws of diminishing returns. In *Proceedings of the 2010 international conference on Power aware computing and systems*, HotPower'10, pages 1–8, Berkeley, CA, USA, 2010. USENIX Association.

[55] Priya Mahadevan, Sujata Banerjee, and Puneet Sharma. Energy proportionality of an enterprise network. In *Proceedings of the first ACM SIGCOMM workshop on Green networking*, Green Networking '10, pages 53–60, New York, NY, USA, 2010. ACM.

[56] Sara S. Mahmoud, Imtiaz Ahmad, and Others. Green performance indicators for energy aware IT systems: Survey and assessment. *Journal of Green Engineering*, 3 (1):33–69, 2012.

[57] Robert N. Mayo and Parthasarathy Ranganathan. Energy consumption in mobile devices: why future systems need requirements–aware energy scale-down. In *Proceedings of the Third international conference on Power - Aware Computer Systems*, PACS'03, pages 26–40, Berlin, Heidelberg, 2004. Springer-Verlag.

[58] Jim A. Mccall, Paul K. Richards, and Gene F. Walters. Factors in Software Quality. Volume I. Concepts and Definitions of Software Quality. Technical report, GENERAL ELECTRIC CO SUNNYVALE CALIF, November 1977.

[59] David Meisner, Brian T. Gold, and Thomas 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, pages 205–216, New York, NY, USA, 2009. ACM.

[60] David Meisner, Christopher M. Sadler, Luiz André Barroso, Wolf-Dietrich Weber, and Thomas F. Wenisch. Power management of online data-intensive services. *SIGARCH Comput. Archit. News*, 39(3):319–330, June 2011.

[61] Microsoft. Energy smart software. *white paper*, 2010.

[62] J. Mitchell-Jackson, J. G. Koomey, B. Nordman, and M. Blazek. Data center power requirements: measurements from Silicon Valley. *Energy*, 28, 2003.

[63] Gordon E. Moore. Cramming more components onto integrated circuits. *Electronics*, 38(8), April 1965.

[64] Navid Behravan Morteza Jamalzadeh. An exhaustive framework for better data centers' energy efficiency and greenness by using metrics. *Indian Journal of Computer Science and Engineering*, 2(6):813–822, 2012.

[65] San Murugesan. Harnessing green it: Principles and practices. *IT Professional*, 10 (1):24–33, January 2008.

[66] Liam Newcombe, Zahl Limbuwala, Payl Latham, and Victor Smith. Data centre fixed to variable energy ratio metric dc-fver, an alternative to useful work metrics which focuses operators on eliminating fixed energy consumption. *BCS Data Centre Specialist Group*, page 29, 2012.

[67] Bruce Nordman and Kenneth J. Christensen. Greener pcs for the enterprise. *IT Professional*, 11(4):28–37, 2009.

[68] Adel Noureddine, Aurelien Bourdon, Romain Rouvoy, and Lionel Seinturier. A preliminary study of the impact of software engineering on greenit. In *GREENS*, pages 21–27. IEEE, 2012.

[69] C. D. Patel, C. E. Bash, R. Sharma, and M. Beitelmal. Smart cooling of data centers. In *Proceedings of IPACK*, 2003.

[70] Barbara Pernici, Danilo Ardagna, and Cinzia Cappiello. In Antonino Mazzeo, Roberto Bellini, and Gianmario Motta, editors, *E-Government Ict Professionalism and Competences Service Science*, volume 280 of *IFIP International Federation for Information Processing*, pages 195–203. Springer US, 2008. ISBN 978-0-387-09711-4.

[71] Meikel Poess and Raghunath Othayoth Nambiar. Energy cost, the key challenge of today's data centers: a power consumption analysis of tpc-c results. *Proc. VLDB Endow.*, 1(2):1229–1240, August 2008.

[72] Meikel Poess and Raghunath Othayoth Nambiar. Power based performance and capacity estimation models for enterprise information systems. *IEEE Data Eng. Bull.*, 34(1):34–49, 2011.

[73] Parthasarathy Ranganathan. Recipe for efficiency: principles of power-aware computing. *Commun. ACM*, 53(4):60–67, April 2010. ISSN 0001-0782.

[74] Parthasarathy Ranganathan, Suzanne Rivoire, and Justin D. Moore. *Models and Metrics for Energy-Efficient Computing*, volume 75. 2009.

[75] Charles Reams. Modelling energy efficiency for computation. 2012.

[76] S. Rivoire, M.A. Shah, P. Ranganathan, C. Kozyrakis, and J. Meza. Models and metrics to enable energy-efficiency optimizations. *Computer*, 40(12):39–48, 2007.

[77] Kurt W. Roth, Fred Goldstein, and Jonathan Kleinman. Energy Consumption by Office and Telecommunications Equipment in Commercial Buildings Volume I: Energy Consumption Baseline. Technical report, US Department of Energy, January 2002.

[78] Cagri Sahin, Furkan Cayci, Irene Lizeth Manotas Gutierrez, James Clause, Fouad Kiamilev, Lori Pollock, and Kristina Winbladh. Initial explorations on design pattern energy usage. In *First International Workshop on Green and Sustainable Software (GREENS)*. ACM, Jun 2012.

[79] Eric Saxe. Power-efficient software. *Queue*, 8(1):10:10–10:17, January 2010.

[80] Greg Schulz. *The Green and Virtual Data Center*. Auerbach Publications, Boston, MA, USA, 1st edition, 2009.

[81] Mircea R. Stan and Kevin Skadron. Powe-aware computing, guest editors' introduction, 2003.

[82] Bob Steigerwald and Abhishek Agrawal. Developing green software, 2011.

[83] Bob Steigerwald, Rajshree Chabukswar, Kathik Krinshnan, and Jun De Vega. Creating energy-efficient software, 2007.

[84] Alexander Kipp Tao Jiang, Cinzia Cappiello, Mariagrazia Fugini, G.R. Gangadharan, Alexandre Mello Ferreira, Barbara Pernici, Pierluigi Plebani, Ioan Salomie, Tudor Cioara, Ionut Anghel, Wolfgang Christmann, Ealan A. Henis, Ronen I. Kat, Marilena Lazzaro, Andrei Ciuca, and Doru Hatiegan. Layered green performance indicators definition. Technical report, Green Active Management of Energy in IT Service centres, June 2010.

[85] Cody Taylor and Jonathan Koomey. Estimating Energy Use and Greenhouse Gas Emissions of Internet Advertising: Working Paper Prepared for IMC2. Technical report, IMC2, February 2008.

[86] The Climate Group. SMART 2020: Enabling the low carbon economy in the information age. Technical report, 2008.

[87] Narendran Thiagarajan, Gaurav Aggarwal, Angela Nicoara, Dan Boneh, and Jatinder Pal Singh. Who killed my battery?: analyzing mobile browser energy consumption. In *Proceedings of the 21st international conference on World Wide Web*, WWW '12, pages 41–50, New York, NY, USA, 2012. ACM.

[88] Shobhit Tiwari. Need of green computing measures for indian it industry. *Journal of Energy Technologies and Policy*, 1(4):18–24, 2012.

[89] Vivek Tiwari, Sharad Malik, Andrew Wolfe, and Mike Tien chien Lee. Instruction level power analysis and optimization of software. *Journal of VLSI Signal Processing*, 13:1–18, 1996.

[90] Niraj Tolia, Zhikui Wang, Manish Marwah, Cullen Bash, Parthasarathy Ranganathan, and Xiaoyun Zhu. Delivering energy proportionality with non energy-proportional systems: optimizing the ensemble. In *Proceedings of the 2008 conference on Power aware computing and systems*, HotPower'08, pages 2–2, Berkeley, CA, USA, 2008. USENIX Association.

[91] Dimitris Tsirogiannis, Stavros Harizopoulos, and Mehul A. Shah. Analyzing the energy efficiency of a database server. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, SIGMOD '10, pages 231–242, New York, NY, USA, 2010. ACM.

[92] Rahul Urgaonkar, Ulas C. Kozat, Ken Igarashi, and Michael J. Neely. Dynamic resource allocation and power management in virtualized data centers. In *NOMS*, pages 479–486. IEEE, 2010.

[93] Vijay R. Vasudevan. Energy-efficient data-intensive computing with a fast array of wimpy nodes. Technical report, Carnegie-Mellon University Pittsburgh PA, School of Computer Science, Oct 2011.

[94] Marc A Viredaz, Lawrence S Brakmo, and William R Hamburgen. Energy management on handheld devices. *Queue*, 1(7):44–52, October 2003.

[95] Mike Wagner. The efficiency challenge: Balancing total cost of ownership with real estate demands. Technical report, Cherokee International, 2008.

[96] Andreas Weissel, Björn Beutel, and Frank Bellosa. Cooperative i/o: a novel i/o semantics for energy-aware applications. In *Proceedings of the 5th symposium on Operating systems design and implementation*, OSDI '02, pages 117–129, New York, NY, USA, 2002. ACM. ISBN 978-1-4503-0111-4.

[97] Mathias Weske. *Business Process Management - Concepts, Languages, Architectures, 2nd Edition*. Springer, 2012.

[98] Joseph Williams and Lewis Curtis. Green: The new computing coat of arms? *IT Professional*, 10(1):12–16, January 2008.

[99] Niklaus Wirth. A plea for lean software. *Computer*, 28(2):64–68, February 1995.

[100] 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*, pages 119–130, 2012.

[101] Andrew J. Younge, G. von Laszewski, Lizhe Wang, and Geoffrey C. Fox. *Providing a Green Framework for Cloud Data Centers*, chapter 17. Chapman and Hall/CRC Press, 01/02/2012 2012.

[102] Lide Zhang, Birjodh Tiwana, Zhiyun Qian, Zhaoguang Wang, Robert P. Dick, Zhuoqing Morley Mao, and Lei Yang. Accurate online power estimation and automatic battery behavior based power model generation for smartphones. In *Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, CODES/ISSS '10, pages 105–114, New York, NY, USA, 2010. ACM.