Improving the Energy Efficiency of IoT-Systems and its Software

Norbert Schmitt

University of Würzburg, Würzburg, Germany,
`norbert.schmitt@uni-wuerzburg.de`,
Homepage: `https://se.informatik.uni-wuerzburg.de`

Abstract. The growing amount of small and inexpensive Internet of Things (IoT) devices consumes energy. But also more opportunities are opened up to conserve energy on operating these devices, not only by monitoring the energy demand but by adapting the IoT system itself. Thus improving energy efficiency by moving computations to IoT or edge devices or the cloud that are suited for a given computation at a particular time. This work presents a vision on how such an IoT system can look and the challenges that are inherent to the problem of runtime self-adaption in the context of code offloading to preserve energy, including improving the energy efficiency of software at design-time through informed design choices.

Keywords: Energy Efficiency, Code Offloading, Self-Aware, Cloud, Green Software Engineering

1 Introduction

In 2011, Cisco estimated a total of 50 billion Internet of Things (IoT) devices operating by 2020 [1]. This number was later scaled down in 2016, with estimations ranging between 20 to 30 billion devices [2]. Even with the more conservative estimations, the amount of IoT devices is still growing rapidly and growth is assumed to continue in the future. All these devices produce data that needs to be processed to be of value. The processing is usually done by cloud data centers. Yet, according to a New York Times study from 2012[1] data centers already consume large amounts of energy. About 30 billion watts per hour worldwide. The U.S. Environmental Protection Agency estimated in 2010 that the U.S. used 3% of its total electricity to run data centers [3]. With the growing amount of data coming from IoT devices, more data centers would be necessary. Therefore making IoT systems consisting of many IoT devices and centers, more energy efficient, becomes more important.

Extensive research has been done to make data centers more energy efficient. Ranging from the machine level, with more efficient hardware [4], up to intelligently placing applications inside a data center [5]. And although the hardware

[1] https://www.nytimes.com/2012/09/23/technology/data-centers-waste-vast-amounts-of-energy-belying-industry-image.html

itself is becoming more energy efficient, there is still little awareness on the impact of software on energy efficiency and the energy efficiency of the software itself. A device's power consumption is indirectly controlled by the software that is defining what data and how data has to be processed [6]. With code offloading, the software not only decides what and how data has to be processed but also where. Most code offloading techniques are concerned mainly with improving the responsiveness of applications on devices with reduced processing power, like smartphones and other mobile devices. The second major topic for code offloading is conserving battery runtime. Regardless, no code offloading technique considers a bigger picture of many low-power interconnected IoT and edge devices. Thus, possible efficiency improvements by optimizing the global state, the time and location where computations take place, instead of single IoT or edge devices and reducing the computational load on the cloud are not considered.

Hence the major goal of this vision is improving the energy efficiency for IoT systems, many connected IoT and edge devices, and software in general. By leveraging edge computing, pushing computations to devices at the edge of a network, like switches and routers, the load on the IoT devices and the cloud can be reduced. Offloading data processing on edge devices should be done with care and only if an effect on the overall energy efficiency of the system, all interconnected devices, can be achieved. Furthermore, not all computations might be feasible for offloading. Depending on a device's capabilities, like available processing power, in combination with the constraints, like the size of the data to be processed and the time to result (timeliness), it can be beneficial or disadvantageous to use edge devices for computation. Thus this vision intends to employ the self-x attributes of Self-Aware Computing and Organic Computing [7] to increase energy efficiency. By adapting and optimizing the current state either by the system itself or controlled by a central entity. Through self-optimization and self-configuration, the system itself learns and decides autonomously where the data processing should take place, the IoT device, the edge or in the cloud. In case data processing is pushed to the cloud auto-scalers geared toward energy efficiency can help reduce energy consumption further. Auto-scalers can base their decision on energy consumption rather than resource usage. Additionally giving developers the right tools to make them more aware of the energy efficiency of their software and the implications of design decisions can further help in reducing energy consumption. This vision, therefore, proposes new research in code metrics that can represent the energy efficiency of software, starting at existing complexity metrics. This vision proposes a novel way to distribute computations between the IoT devices, edge devices and the cloud as well as novel code metrics to help make developers informed design decisions to increase energy efficiency.

The remainder of this paper is structured as follows: First, an overview of the current related work is presented. Chapter 3 describes the current state of the art in energy efficiency in data centers, code offloading and code metrics. In the following chapter 4, identified challenges and problems are explained together with possible approaches of how to solve these challenges. Finally, this paper concludes in Chapter 5.

2 Related Work

Energy efficiency is a growing concern. Therefore extensive research has been done to improve the energy efficiency. The release of the Server Efficiency Rating Tool (SERT) [8] 2.0 in 2017[2] also demonstrates a consistent interest in this topic. With the growth of data centers for cloud computing and virtualization, the degrees of freedom to conserve energy have further increased from machine level to whole data centers. This section gives a short overview of the related work, grouped into three topics, cloud energy efficiency, code offloading and software energy efficiency.

2.1 Cloud Energy Efficiency

The most common reduction in power consumption in cloud computing is through virtual machine (VM) migration [5]. A management software decides on which physical machine a VM gets deployed [9,10,11,12]. They focus on reducing the power consumption without or minimal impact on performance to increase the energy efficiency of cloud servers. Other examples are considering more constraints, such as quality of service (QoS) [13,12] or service level agreements (SLA) [14]. They all rely on common utilization power models or build their own model, such as Tian et al. [11]. Yet, servers are not the only power consumer in large data centers. By virtualizing formerly physical devices of the network infrastructure, researchers try to improve energy efficiency further [15,16]. This paper focuses less on cloud energy efficiency and the placement of applications inside a data center, but rather scaling the needed application instances inside the cloud efficiently with an energy-aware auto-scaler that is either reactive [17] or proactive [18].

2.2 Code Offloading

Code offloading is not a new technology and it has been argued that it can be beneficial under certain conditions [19]. Namely, the trade-off between computational and communication effort must be towards low communication and high computation. There is also a variety of frameworks available with different goals [20,21,22,23,24,25]. Yet, they either focus on scalability of the application, performance or conserving battery capacity for mobile devices. They also can only offload computations to the cloud and not between devices. As only the own individual devices are optimized for saving energy and are limited to cloud offloading, this work proposes an offloading technique that can share work between IoT, edge and cloud devices for the benefit of saving energy in the overall system.

[2] https://www.spec.org/sert/SERT-2.0-press-release.html

2.3 Software Energy Efficiency

Software developer choices can influence the energy consumption. The authors show that different implementations for the same task as well as different parameterization, in the case of Singh et al. the buffer size for reading and writing files, can change the energy consumption without changing the program's semantics [26]. Efforts were undertaken to determine the energy consumption resulting from executing a given source code, but most work relies to some degree on information about the hardware. Modeling a software's energy consumption is either done by training a model with measurements [27] or by defining a workload profile from user input together with expert knowledge of the hardware [28]. This paper tries to find new metrics that can reflect the energy consumption of a running software by source code analysis.

3 Foundation

This chapter gives a short introduction to the definitions and metrics used throughout this paper. This includes energy efficiency, the most efficient state for an IoT system, the cyclomatic complexity and Halstead complexity metrics.

3.1 Energy Efficiency

This section is divided into two parts. First, the definition of energy efficiency, followed by the definition of the most energy efficient state of an IoT system. The term energy efficiency in this work is used according to Equation 1, also used by the Standard Performance Evaluation Corporation (SPEC) [29]. The efficiency *eff* is the throughput t per watt (power p). The throughput is given in $\frac{1}{s}$, the number of finished function calls per second.

$$\text{eff} = \frac{t}{p} \quad \left[\frac{1}{Ws}\right] \tag{1}$$

An IoT system in this paper is defined as a network of IoT devices, from sensors to mobile devices, and also including edge devices and the cloud. The structure of the network itself is not relevant in this case and can be a mesh or hierarchy. The optimal state of such an IoT system is considered to be in such a way that not each individual device consumes the least amount of energy for a given computation but the combined energy consumption for all necessary computations in the system. Due to this definition, not each device's power draw is minimized. The definition allows individual devices to be less efficient if it benefits the overall system. An example is shown in Figure 1. Here an IoT system with different computations, A, B and C, is in an optimal state but changes due to Dev. 4 joining the network. As the additional device disturbs the current state, transfering it into a suboptimal state, the system must change to restore a better energy efficiency. The system than uses code offloading on methods A and C on Dev. 4 to return itself to a new optimal state including Dev. 4.

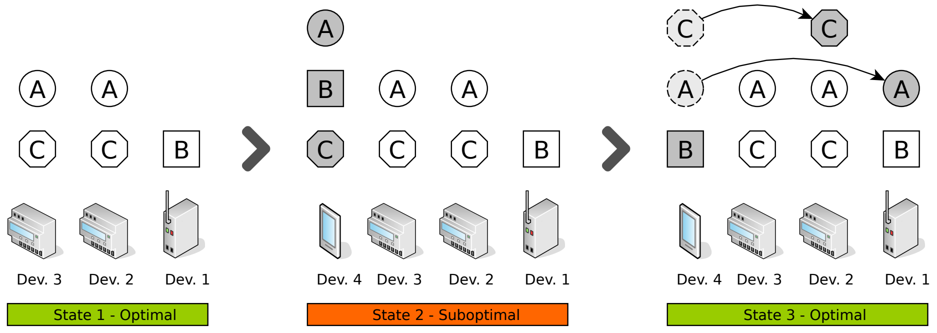

Fig. 1. Resuming to an optimal, more energy efficient, state for the overall system after a mobile device (Dev. 4) joins the system.

3.2 Software Metrics

For preliminary measurements, two well known software metrics were used. The *cyclomatic complexity* and the *Halstead complexity metrics*. The cyclomatic complexity represents the program in the form of a control flow graph. Nodes (N) in the graph represent locations in the code at which a decision must be made. The directed edges E show the flow of the program after a decision is made. The cyclomatic complexity C is then calculated as follows:

$$C = E - N + 2P \tag{2}$$

E is the number of edges in the control flow graph and N is the number of nodes. The value P represents the number of connected components of the graph. For the preliminary measurements containing only single functions and no real world program, connected components P is equal to 1. Thus simplifying the equation to $C = E - N + 2$.

The Halstead complexity metrics build upon four measures. They include:

- N_1: Total number of operators
- N_2: Total number of operands
- η_1: Number of distinct operators
- η_2: Number of distinct operands

From these measure, several metrics can be calculated to express a program's complexity and size. By adding the distinct and total counters to the vocabulary $\eta = \eta_1 + \eta_2$ and length $N = N_1 + N_2$ of a program, several metrics can be calculated shown in Equation 3.

$$\begin{aligned} \hat{N} &= \eta_1 \cdot \log_2 \eta_1 + \eta_2 \cdot \log_2 \eta_2 \qquad & V &= N \cdot \log_2 \eta \qquad & D &= \frac{\eta_1}{2} \cdot \frac{N_2}{\eta_2} \\ E &= D \cdot V & T &= \frac{E}{18} & B &= \frac{E^{\frac{2}{3}}}{3000} \end{aligned} \tag{3}$$

- $\hat{N}$: The calculated program length (not lines of code).
- V: The program volume. A second measure, next to $\hat{N}$, of a program's size.
- D: The difficulty to write the program, how complex is the code.
- E: The effort to write the program, how much work must be put in to produce the code.
- T: Calculated time to write the program, how long it will take to write the code.
- B: The number of delivered bugs.

4 Challenges

In the following chapter, the challenges identified in this paper are presented together with first ideas for a solution and preliminary measurements if available. The challenges are divided in runtime challenges and design-time challenges. Runtime challenges include problems arising during operation of an IoT system while the design-time challenges are concerned with developing energy efficient software.

4.1 At Runtime

IoT systems consist of a multitude of different devices, as shown in Figure 2, each with their capabilities regarding computational power, network, and energy consumption. IoT devices can have very little to no computational power and act just like data sources. Other devices can have more computational power and therefore higher energy consumption but can do a certain amount of data processing themselves without the cloud. The workload that a device can stem, is not only constrained by its hardware but also its energy source and computation speed. These constraints can be, but are not limited to:

- Time and space complexity of the algorithm to compute.
- Size of the data set.
- Energy consumption per computation (energy efficiency).
- Timeliness of the result, including latency.
- Locality of data (privacy).

For example, a battery powered surveillance camera could provide face recognition itself as long as the battery holds enough charge and offload the face recognition to an edge device or cloud server when the battery runs low. Additionally, devices can be mobile. While stationary devices often communicate with fixed devices after setup, moving devices switch between communication partners frequently to stay available in the network to transmit data.

The network itself can be organized in a hierarchical or a mesh structure. Even in the case the system already reached the most energy efficient state, disturbances by a change in the environment, either by a defect, by sensors moving in and out of reach of base stations or by sensors with critically low

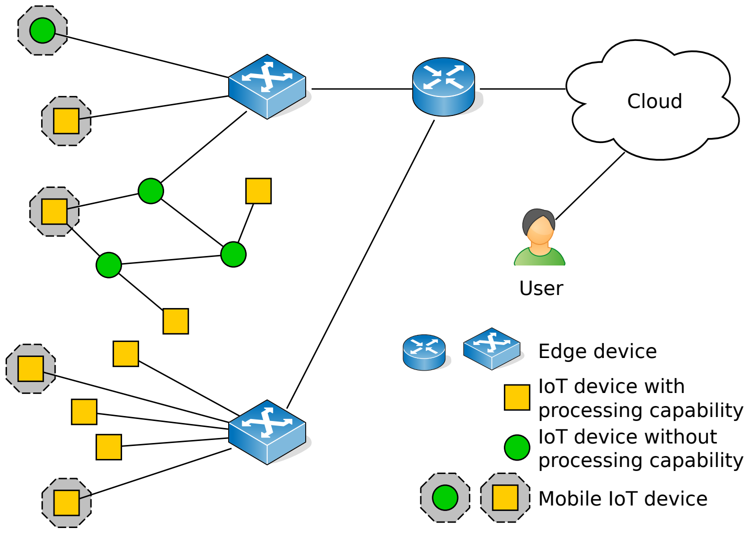
Cloud
User
Edge device
IoT device with processing capability
IoT device without processing capability
Mobile IoT device

Fig. 2. Internet of Things System Example

battery charge, can result in a suboptimal state. For this reason, an automatic offloading of computations to other devices should be triggered. The offloading can be done either by a central instance or by the device that needs offloading if the optimal state is disturbed.

The current state developing for IoT devices is, that each device must be programmed and set up independently and is usually limited to this setup. If the programming and deployment are not done by an expert, it is most likely not in an optimal state regarding energy efficiency. The vision, therefore, aims to remove expert knowledge and rely on self-improvement. It is divided into two parts, the first, making and keeping the IoT system more energy efficient by reactively or proactively distributing computations among IoT devices, edge devices and the cloud. Secondly, improving energy efficiency of the software, primarily computations performed on IoT devices.

Under these circumstances, the following challenges were identified.

C1: *How can the energy efficiency of the system be modelled?*

Before the system can make any meaningful decision about itself, it must know its state. Hence a model of the system must be created from which decisions can be derived. The current assumption is that there exist two different decisions that must be made possible by such a model. The first one, is an immediate decision that must not be optimal but quick to react to sudden changes in the environment like a defect. The second decision is a long term planning to optimize the system. An idea is to minimize the amount of immediate decisions and proactively shifting computations by predicted disturbances in the environment. The model can be based on hardware specifics or the running workload. For the hardware specific model, performance counter have proven to be suited well as a multitude of models show [30,31,32,33,34,35,36,37]. Performance counter have the downside that expert knowledge about a device must be available to be useful. Other higher-level metrics such as CPU usage are also feasible options. For models not dependent on hardware, request rates or similar metrics can act as a basis. Both options should be explored to overcome Challenge C1.

C2: *When should the computation be offloaded from a device?*

After determining the current state in Challenge C1, the next challenge can be adressed. For Challenge C2, the idea is to show that each device has certain trade-offs regarding its constrains mentioned earlier. As stated in Chapter 2, offloading might only be beneficial under these constraints. First, a decision algorithm must be derived if code offloading should actually take place, taking into account the battery state (if any), the size of the data to transmit compared to the network bandwidth, the timeliness including possible network latencies and the energy consumption of the devices involved in the offloading process. To derive a decision algorithm, first a careful selection of devices with different characteristics and capacities has to be made to allow for a general answer. Furthermore, not only the hardware is important but also the software. To show that different computations are more suitable to run on edge, IoT devices, or the cloud, a selection of different algorithms with different time and space complexities must be selected as synthetic workloads. For example, different sorting algorithms like randomized quicksort with $\mathcal{O}(n \log(n))$ and insertion sort with $\mathcal{O}(n^2)$ for sorting data. Real world workloads should also be included like face recognition. It can then be determined how the energy efficiency behaves in relation to increasing computational complexity, increasing size of data sets and time-to-result.

C3: *Where should the computation take place?*

As a next step for the Challenge C3, a deployment model must be derived, based on the results of C2 and the state of the art in placement algorithms. This includes knowing what data size is to expected for a specific computation.

For all devices in the IoT system, its optimal and maximal computational capacity must be known. While the capacity for a specific device can be read from data sheets or measured, the requirements of the algorithms must be estimated,

Application	Simulation Time (s)
Media Store	25.6
SPECjEnterprise2010	55.6
Process Control System	65.4
Business Reporting System	587.6

Table 1. Solving time for different performance models from Brosig et al. [38].

simulated or modelled, as running them on each device can quickly become infeasible due to the large variety of devices.

C3 will be addressed by first deriving a simple decision tree, checking if all requirements of an algorithm are met for deployment on a certain device. If not all requirements are met, the workload cannot be computed on the device in question and another one must be chosen. This can later be extended to more complicated and better solutions if necessary. However, more important is to identify the resource and time-to-result requirements of the algorithm. It is intended to use static code analysis combined with machine learning to estimate the resource requirements. The time-to-result requirements must be provided by the developer as identical algorithms can have different requirements.

C4: *How to design an auto-scaler to make informed decisions about energy efficiency?*

Auto-scalers are used in cloud environments to automatically reduce or increase the number of resources, normally virtual machines or container. This is mostly used to save money as only the necessary amount resources are used and paid for. An auto-scaler could also be used to conserve energy by deploying, shutting down and moving application instances. By giving the auto-scaler a model of the IoT system designed in Challenge C1 it can make informed decisions about energy efficiency. However, solving a model can take large amounts of time, especially for IoT and edge devices with limited computational capacity. Table 1 shows the time to solve performance models of increasing size. As the typical cloud is used by many customers, solving a model for each customer to determine the best solution is time consuming. A cloud with 1000 customers, in this case IoT systems, and an average solving time of 30s needs over 8 hours to solve all models, not including the time to reach a decision from the model's output.

4.2 At Design-Time

With the previous challenges focusing on the runtime aspect, the remaining challenges focus on the design time of the system. To improve energy efficiency further, the developers should be more aware if and how certain design decisions and programming patterns can influence energy efficiency. To be helpful for the developer, a quick response is key or otherwise the information will not be integrated in the final program. If a developer must wait several minutes or even hours to know the impact of a small code change will discourage its usage. With the added complexity that the information should be as hardware independent as possible, the energy efficiency must be estimated without actually benchmarking the code on real hardware. Requirements on the system can change over time. While the devices can be exchanged and the system should adapt itself, the software on the other hand must be adapted manually. By changing the software, it can direct how energy is consumed on a device without changing the software's semantics. Determining how energy efficient a piece of software can run from the sources alone is challenging.

C5: *Are current metrics suitable to determine the runtime energy efficiency behavior of software?*

To overcome Challenge C5, first, it must be surveyed at what level design decisions can be made that negatively affect energy efficiency. This could include design patterns, programming language keywords and others. Their influence on energy efficiency must be identified to give valuable feedback to the developer. The first idea is to check existing code metrics and how they relate to energy efficiency. As a first impression, seven algorithms with different runtime behaviors $\mathcal{O}\left(n^2\right)$, $\mathcal{O}\left(n \cdot \log\left(n\right)\right)$, $\mathcal{O}\left(n\right)$ and $\mathcal{O}\left(\log\left(n\right)\right)$ were selected and measured. The algorithms are:

– Linear search
– Binary search
– Mandelbrot escape time algorithm
– Sieve of Eratosthenes (prime number calculation)
– Insertion Sort
– Merge Sort
– Quick Sort

All were measured with data sizes ranging from 400 to $4096 \cdot 10^2$ bytes. The size was doubled after each measurement of 240 seconds, resulting in 11 total measurements for each algorithm per measurement run. The only exception is the Mandelbrot escape time algorithm with only 9 measurements per measurement run. For the escape time algorithm, image sizes starting at $100x100$ were calculated. First the x value is doubled. Afterwards the y value is doubled up to the maximum image size of $1600x1600$. Larger image sizes were not applicable. Each measurement run is repeated eight times. From the data, the average bytes processed per Ws is calculated with Equation 4 with $n = 8$. The number of iterations (how often the algorithm could be executed in a measurement) after

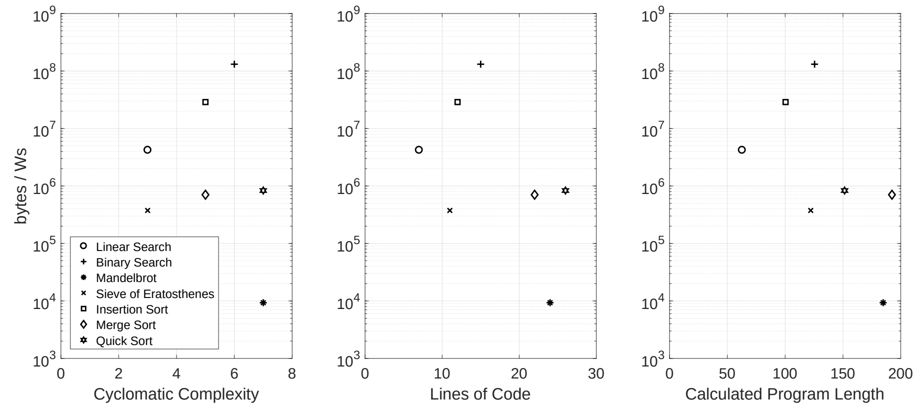

Fig. 3. Static code metrics and the amount of data (bytes) processed for each Ws consumed by different algorithms.

240s is recorded to calculate the number of bytes processed, or generated in the case of the escape time algorithm, by multiplying it with the known data size for each iteration. The idle power p_{idle} is subtracted from the measured average power p_{avg} to only show the power consumed by the active application.

$$\frac{1}{n} \cdot \sum_{i=1}^{n} \frac{iterations_i \cdot bytesPerIteration_i}{(p_{avg} - p_{idle}) \cdot 240s} \tag{4}$$

From the measurement data, the average bytes processed per Ws is presented in Figure 3 to show how energy efficient the implementation is. The calculated complexity metric is on the x-axis and average bytes processed is on the y-axis of Figure 3.

It seems odd that insertion sort has better energy efficiency in this scenario then the divide-and-conquer sorting algorithms. This behavior could be due to the linear iteration over the data array making better use of the optimizations in hardware like memory prefetchers.

The figure also shows two groups of algorithms. Group 1 with the linear search, binary search and insertion sort, group 2 with the sieve of Eratosthenes, merge and quick sort and mandelbrot escape time algorithm as a possible outlier. Table 2 shows the Pearson correlation after applying the log function to both groups. The measurement hints that arranging algorithms into groups can give reasonably accurate results with existing code metrics to represent energy efficiency of an implementation. While the correlation is very high, the values should be taken with skepticism as only three data points are available for each group and further measurements must be made to support the first impression. Especially as the correlation including all algorithms does not support the use of existing complexity code metrics, shown in Section 3.2.

The preliminary measurements show that code metrics can possibly be used to give developers a base for comparison of different implementations but further

Metric		C	LoC	η	N	$\hat{N}$	V	D
Correlation	All	−0.1996	−0.5142	−0.5868	−0.3832	−0.5888	−0.4263	−0.0293
	Group 1	0.9925	0.9971	0.9996	0.9944	0.9989	0.9923	0.9957
	Group 2	0.9484	0.9982	0.7185	0.9562	0.6820	0.9360	0.9534

Table 2. Correlation coefficients for different algorithms and software metrics.

work is necessary to identify the code constructs that are responsible for making an algorithm run energy efficient or not.

C6: *How can the energy efficiency behavior of the source code be communicated to the developer?*

As a second part to convey energy efficiency of software to developers is the introduction of a energy efficiency regression test, similar to unit testing. As DevOps is growing among software developers, the time between a new version of software and deployment is reducing. To ensure that energy efficiency remains constant or improves, developers need to be aware of the implications of changes in the source code to the energy efficiency. Based on the results from Challenge C5, a regression test could be implemented to show if the energy efficiency of the software has improved over time. While the metrics in C5 can be calculated on the developers machine to provide quick feedback, a regression test can run predefined workloads on the application on a reference machine. Although this machine could possibly be a virtual machine, it is highly depending on the metrics, as some metrics could be dependent on hardware information such as performance counter that are not available in a virtualized environment.

5 Conclusion and Future Work

This paper introduces the vision of making IoT systems and their software more energy efficient. Six high-level challenges in Chapter 4 are presented that need to be overcome. The challenges include finding suitable modelling techniques, monitoring parameters for code offloading between devices and application placement, cloud auto-scaling. Additionally, static code metrics representing energy efficiency are discussed, making developers more aware of energy efficiency in their software design choices. Ideas and possible solutions, where to execute certain computations in an IoT system, are presented. The preliminary measurements for static code metrics show that they might be usable if algorithms or even full applications can be sorted into groups with different energy efficiency behavior.

As a first step towards more energy efficient IoT systems, an abstract model representation according to Challenge C1 is vital. After the state of an IoT system can be determined, progress in the following runtime challenges can be achieved. In Challenge C3, after building a decision model, with a decision tree as a first choice, current suitable solutions to the placement problem must be surveyed and evaluated.

The design time challenges in Chapter 4.2 can be addressed simultaneously. First deriving a new static code metric to allow comparison of semantically identical piece of source code and secondly, given developers a tool to raise awareness that design time decisions can impact energy efficiency. This could be achieved with a energy efficiency regression testing or other options.

References

1. Evans, D.: The internet of things - how the next evolution of the internet is changing everything. Technical report, Cisco Internet Business Solutions Group (IBSG) (April 2011) White Paper.
2. Nordrum, A.: The internet of fewer things [news]. IEEE Spectrum **53**(10) (2016) 12–13
3. Lange, K.D., Tricker, M.G.: The Design and Development of the Server Efficiency Rating Tool (SERT). In: Proceedings of the 2nd ACM/SPEC International Conference on Performance Engineering. ICPE '11, New York, NY, USA, ACM (2011) 145–150
4. Rotem, E., Naveh, A., Ananthakrishnan, A., Weissmann, E., Rajwan, D.: Power-management architecture of the intel microarchitecture code-named sandy bridge. IEEE Micro **32**(2) (March 2012) 20–27
5. Jin, Y., Wen, Y., Chen, Q.: Energy Efficiency and Server Virtualization in Data Centers: An Empirical Investigation. In: 2012 IEEE Conference on Computer Communications Workshops. (March 2012) 133–138
6. Calero, C., Piattini, M.: Green in software engineering. Springer (2016)
7. Kounev, S., Kephart, J.O., Milenkoski, A., Zhu, X., eds.: Self-Aware Computing Systems. Springer Verlag, Berlin Heidelberg, Germany (2017)
8. Lange, K.D., Tricker, M.G.: The Design and Development of the Server Efficiency Rating Tool (SERT). In: ICPE 2016. ICPE '11, New York, NY, USA, ACM (2011) 145–150
9. Hiltunen, M.A., Schlichting, R.D., Jung, G., Pu, C., Joshi, K.R.: Mistral: Dynamically managing power, performance, and adaptation cost in cloud infrastructures. In: 2010 IEEE 30th International Conference on Distributed Computing Systems(ICDCS). Volume 00. (06 2010) 62–73
10. Urgaonkar, R., Kozat, U., Igarashi, K., Neely, M.: Dynamic resource allocation and power management in virtualized data centers. In: Network Operations and Management Symposium (NOMS), 2010 IEEE. (April 2010) 479–486
11. Tian, Y., Lin, C., Yao, M.: Modeling and analyzing power management policies in server farms using stochastic petri nets. In: Future Energy Systems: Where Energy, Computing and Communication Meet (e-Energy), 2012 Third International Conference on. (May 2012) 1–9
12. Beloglazov, A., Buyya, R.: Energy Efficient Resource Management in Virtualized Cloud Data Centers. In: 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing, IEEE (2010)

13. Beloglazov, A., Abawajy, J., Buyya, R.: Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing. Future generation computer systems **28**(5) (2012) 755–768
14. Chase, J.S., Anderson, D.C., Thakar, P.N., Vahdat, A.M., Doyle, R.P.: Managing energy and server resources in hosting centers. ACM SIGOPS operating systems review **35**(5) (2001) 103–116
15. Botero, J.F., Hesselbach, X., Duelli, M., Schlosser, D., Fischer, A., de Meer, H.: Energy Efficient Virtual Network Embedding. IEEE Communications Letters **16**(5) (March 2012) 756–759
16. Bolla, R., Bruschi, R., Lombardo, C., Mangialardi, S.: DROPv2: Energy-Efficiency through Network Function Virtualization. IEEE Network **28**(2) (April 2014) 26–32
17. Chieu, T., more: Dynamic Scaling of Web Applications in a Virtualized Cloud Computing Environment. In: IEEE ICEBE 2009, IEEE (2009) 281–286
18. Bauer, A., Herbst, N., Spinner, S., Ali-Eldin, A., Kounev, S.: Chameleon: A Hybrid, Proactive Auto-Scaling Mechanism on a Level-Playing Field. IEEE Transactions on Parallel and Distributed Systems (September 2018) Preprint.
19. Kumar, K., Lu, Y.H.: Cloud computing for mobile users: Can offloading computation save energy? Computer **43**(4) (2010) 51–56
20. Cuervo, E., Balasubramanian, A., Cho, D.k., Wolman, A., Saroiu, S., Chandra, R., Bahl, P.: Maui: making smartphones last longer with code offload. In: Proceedings of the 8th international conference on Mobile systems, applications, and services, ACM (2010) 49–62
21. Chun, B.G., Ihm, S., Maniatis, P., Naik, M., Patti, A.: Clonecloud: elastic execution between mobile device and cloud. In: Proceedings of the sixth conference on Computer systems, ACM (2011) 301–314
22. Kosta, S., Aucinas, A., Hui, P., Mortier, R., Zhang, X.: Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading. In: Infocom, 2012 Proceedings IEEE, IEEE (2012) 945–953
23. Gordon, M.S., Jamshidi, D.A., Mahlke, S.A., Mao, Z.M., Chen, X.: Comet: Code offload by migrating execution transparently. In: OSDI. Volume 12. (2012) 93–106
24. Flores, H., Srirama, S.: Adaptive code offloading for mobile cloud applications: Exploiting fuzzy sets and evidence-based learning. In: Proceeding of the fourth ACM workshop on Mobile cloud computing and services, ACM (2013) 9–16
25. Schafer, D., Edinger, J., Paluska, J.M., VanSyckel, S., Becker, C.: Tasklets:" better than best-effort" computing. In: Computer Communication and Networks (ICCCN), 2016 25th International Conference on, IEEE (2016) 1–11
26. Singh, J., Naik, K., Mahinthan, V.: Impact of developer choices on energy consumption of software on servers. Procedia Computer Science **62** (2015) 385–394
27. Seo, C., Malek, S., Medvidovic, N.: Component-level energy consumption estimation for distributed java-based software systems. In: International Symposium on Component-Based Software Engineering, Springer (2008) 97–113
28. Hao, S., Li, D., Halfond, W.G., Govindan, R.: Estimating mobile application energy consumption using program analysis. In: Proceedings of the 2013 International Conference on Software Engineering, IEEE Press (2013) 92–101
29. von Kistowski, J., Lange, K.D., Arnold, J.A., Sharma, S., Pais, J., Block, H.: Measuring and benchmarking power consumption and energy efficiency. In: Companion of the 2018 ACM/SPEC International Conference on Performance Engineering. ICPE '18, New York, NY, USA, ACM (2018) 57–65
30. Isci, C., Martonosi, M.: Runtime power monitoring in high-end processors: Methodology and empirical data. In: Proceedings of the 36th Annual IEEE/ACM Interna-

tional Symposium on Microarchitecture. MICRO 36, Washington, DC, USA, IEEE Computer Society (2003) 93–
31. Contreras, G., Martonosi, M.: Power prediction for intel xscale/spl reg/ processors using performance monitoring unit events. In: ISLPED '05. Proceedings of the 2005 International Symposium on Low Power Electronics and Design, 2005. (Aug 2005) 221–226
32. Singh, K., Bhadauria, M., McKee, S.A.: Real time power estimation and thread scheduling via performance counters. SIGARCH Comput. Archit. News **37**(2) (July 2009) 46–55
33. Chen, X., Xu, C., Dick, R.P., Mao, Z.M.: Performance and power modeling in a multi-programmed multi-core environment. In: Proceedings of the 47th Design Automation Conference. DAC '10, New York, NY, USA, ACM (2010) 813–818
34. Lim, M.Y., Porterfield, A., Fowler, R.: Softpower: Fine-grain power estimations using performance counters. In: Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing. HPDC '10, New York, NY, USA, ACM (2010) 308–311
35. Bircher, W.L., John, L.K.: Complete system power estimation using processor performance events. IEEE Transactions on Computers **61**(4) (April 2012) 563–577
36. Rodrigues, R., Annamalai, A., Koren, I., Kundu, S.: A study on the use of performance counters to estimate power in microprocessors. IEEE Transactions on Circuits and Systems II: Express Briefs **60**(12) (Dec 2013) 882–886
37. Tsafack Chetsa, G.L., Lefèvre, L., Pierson, J.M., Stolf, P., Da Costa, G.: Exploiting performance counters to predict and improve energy performance of HPC systems. Future Generation Computer Systems **vol. 36** (July 2014) pp. 287–298
38. Brosig, F., Meier, P., Becker, S., Koziolek, A., Koziolek, H., Kounev, S.: Quantitative Evaluation of Model-Driven Performance Analysis and Simulation of Component-based Architectures. IEEE Transactions on Software Engineering (TSE) **41**(2) (February 2015) 157–175