Using energy consumption for self-adaptation in FaaS

Pablo Serrano-Gutierrez[1,2][0000−0002−5397−690X] and Inmaculada Ayala[1,2][0000−0002−5119−3469]

[1] Departamento de Lenguajes y Ciencias de la Computación, Universidad de Málaga, España
[2] ITIS Software, Universidad de Málaga, España
{pserrano,ayala}@lcc.uma.es

Abstract. One of the programming models that has been developing the most in recent years is Function as a Service (FaaS). The growing concern over data centre energy footprints has driven sustainable software development. In serverless applications, energy consumption depends on the energy consumption of the application's functions. However, measuring energy proves challenging, and the results' variability complicates optimisation efforts at runtime. This article addresses this issue by measuring serverless function energy consumption and exploring integration into an optimisation system that selects implementations based on their current energy footprint. For this, we have integrated an energy measurement software into a FaaS system. We have analysed how to properly process the data and how to use them to perform self-adaptation. We present a series of methods and policies that make our system not only capable of detecting variations in the energy consumption of the functions, but it does so taking into account the variability in the measurements that each function may present. Our experiments showcase proper integration in a self-adaptive system, showing a reduction up to 5% in energy consumption due to functions in a test application.

Keywords: sustainability · serverless · self-adaptive.

1 Introduction

The growing concern about the energy footprint of datacentres [11, 3] has encouraged the development of more sustainable software. Green Software engineering is a field that pursues software development and deployment more sustainably [8]. One of the most significant challenges in the field is measuring the energy consumption of the software. It can be measured with both hardware and software tools. Hardware tools provide greater accuracy but, on the other hand, are more expensive and challenging to implement. Furthermore, it is tough to isolate the energy consumption of a particular piece of software or process using this kind of solution because it measures the energy consumption of the whole system. Regarding software tools, we can find two types, some based on estimates

and others using hardware sensors. The latter has been developed recently since these hardware elements began to be incorporated into microprocessors just a decade ago.

Usually, to try to reduce the consumption of an application, optimisations are carried out or different configurations are tested, and for each case energy measurements are made at a global level to determine if consumption is reduced. The use of software tools allows us to not only measure overall consumption but also portions of the application, which can be used to optimise the different parts of it separately. Nevertheless, this is difficult to do in a traditional application, especially at runtime. However, due to their nature, serverless applications or those based on microservices are especially suitable for being optimised in parts.

Function as a Service (FaaS) environments are increasingly used since they save infrastructure maintenance and management costs. Serverless frameworks use virtual machines or containers to host serverless functions, which allows functions programmed in different programming languages to be used in the same application. This feature facilitates the integration of different work teams and software reuse. In addition, these frameworks support the automatic horizontal and vertical scaling of serverless functions based on demand.

Often, it is possible to implement a function in different ways. Also, when software is reused, we can have several implementations that suit the needs of the application. Each of these may be better suited to a given infrastructure or may have different performance from the point of view of user experience. For example, it is possible that an implementation is very energy efficient but needs a lot of available memory to work properly, so, in situations where memory is limited, it is possible that another implementation that initially was less energy efficient but does not need to use as much memory, behave better. So, an interesting approach to try to reduce consumption is to choose which of these functions are most appropriate to run in our infrastructure in terms of energy consumption.

Energy consumption of serverless applications is dynamic and challenging to predict at design time [5]. It depends on the infrastructure where the container is deployed and the execution conditions. In this work, we present the results of our experiments on energy saving for serverless applications. We have integrated an energy measurement software tool, Scaphandre [3], in a framework for the self-adaptation of serverless applications, and show how to use energy measurements to save energy of the serverless application at runtime. For this, we have defined procedures and rules that can be adapted to the necessities of an application. By executing the test cases, we verify that the system has self-adaptive behaviour, reducing the consumption of the application due to functions up to 5%.

This work is structured as follows: Section 2 presents some related work; Section 3 explains how we have integrated Scaphandre in the framework; Section 4 illustrates our approach for the analysis of energy measurements; Section 5 shows how to use it at runtime for energy saving; Section 6 presents the case study; and the paper finishes with some conclusions and ideas of future work.

[3] https://github.com/hubblo-org/scaphandre

2 Related work

Several studies analyse energy consumption in serverless systems. Moreno et al. [9] investigate the impact of cold-start techniques on response time and energy consumption, suggesting extending serverless platforms to the edge for latency reduction. Alhindi et al. [1] evaluate power efficiency of OpenFaaS [6] compared to Docker containers, highlighting energy efficiency of OpenFaaS in certain scenarios.

Other research focuses on energy-aware placement and scaling of serverless functions. Tsenos et al. [12] propose an Energy Efficient Scheduler to minimise energy consumption while meeting performance demands. FADE [13] is a tool proposed by Tzenetopoulos et al. that decomposes applications into serverless functions, considering energy consumption prediction for efficient node selection.

Other studies consider the architecture of the system. MicroFaaS [2] suggests energy-efficient architectures, utilising single-board computers to increase FaaS function energy efficiency by a factor of 5.6. Jiaxuechao et al. [5] analyse energy consumption breakdown in serverless computing, introducing energy fungibility and demonstrating its feasibility with a framework that can reduce function energy consumption up to 21.2%.

In conclusion, current work does not analyse the power consumption of functions individually and does not address consumption reduction based on the use of different implementations, which can be important in software reuse, as functions that have not been specifically designed to work on a specific system are often used.

3 Integrating energy measurement into FaaS

The solution presented in this section uses OpenFaaS [6], a framework for serverless functions, and Scaphandre, a tool to measure energy in containerised applications. We have selected OpenFaaS because it is a free-to-use serverless framework and energy efficient [1], but the approach can be easily adapted to other serverless frameworks.

As we have stated, Scaphandre is an energy measurement software tool which is currently under development. This tool bases its operation on RAPL (Running Average Power Limit) [4], a hardware element whose function is to limit consumption and which can be used as a source for these measurements. RAPL generates information about the system's energy consumption on a cumulative basis, and Scaphandre samples it and extracts information. As we know, *Energy=Power·time*, so it is easy for Scaphandre to calculate the power consumed by the system from the energy samples. However, to obtain the energy consumed by applications or a process, the procedure becomes complicated since there is no way for RAPL to know which process each instruction being executed on the microprocessor belongs to. Scaphandre performs this task using jiffies, which are the CPU ticks used by the application. However, the calculation is more complex since it must consider other parameters, such as the core on which it is executed.

OpenFaaS deploys Docker[4] containers for each function and orchestrates them using Kubernetes[5]. Therefore, to determine the consumption of the serverless functions, we must know the consumption of the associated containers. Scaphandre can obtain measurements per process, allowing us to get the consumption of serverless functions. Each replica has an associated container corresponding to a running process. The information that Scaphandre returns about the processes is the power in microwatts (*scaph_process_power_consumption_microwatts*). However, to compare the energy consumption of the functions, the metric of interest is energy. Two functions can consume the same power, but it consumes more energy if one takes longer to execute. Therefore, our solution also measures the execution times of the functions to estimate their energy.

Scaphandre has numerous exporters through which we can read the data it provides. In our approach, we use Prometheus[6], a powerful software used extensively for containerised applications and also used by OpenFaaS. We configure Scaphandre to serve the data to Prometheus. To do this, we made a custom installation of OpenFaaS, setting Scaphandre as an additional data source.

Scaphandre can be installed on Kubernetes and has an option called *containers* that links container names with the energy measurements of the processes facilitating their identification. This information is obtained from the data contained in *proc/PID/cgroup* directory of the container structure. However, this is not possible for containers created by OpenFaaS. There, we had two options to get energy measurements: we could modify these containers to contain the appropriate information or search for the container manually. We opted for the second option because it facilitates this solution's future integration with other FaaS frameworks. The search procedure is as follows. Firstly, we obtain the identifier of a container associated with the function of interest to us by querying Kubernetes. Subsequently, we obtain the identification of the process associated with the said container, i.e., the *PID*, and, finally, we make a query to Prometheus with said *PID*. This procedure must be repeated for each container associated with each function replica. Adding the energy consumption of all the replicas, we will obtain the consumption of the function.

4 Analysing energy measurements

Before using the Scaphandre measurements to save energy, it is necessary to analyse the data obtained. After measuring the energy consumed by serverless functions, we observed significant variability. Suppose we want to compare the energy consumed by two implementations statically. In that case, it is not a problem since we can perform measurements over long periods and assign average values to each. However, if we want the system to react to changes that may occur in consumption over time, what interests us is that it can do so in a short period of time.

[4] https://www.docker.com/
[5] https://kubernetes.io/
[6] https://prometheus.io/

Function	Description	CPU usage	Memory usage	Ext. requests
f1	Get node info	Low	Low	No
f2	Text printing	Low	Medium	No
f3	Mathematical calculations	High	Low	No
f4	Search and manipulation of data	Medium	Medium	Yes
f5	Data compression	High	Medium	Yes
f6	OpenCV image analysis	High	High	Yes

Table 1. Serverless functions analysed

The fluctuations in serverless functions' energy consumption behaviour can cause unnecessary application adaptations. In this context, comparisons with alternate implementation become even more challenging. To avoid this, we can use several methods. One consists of waiting for the energy values to remain at a certain level for a time. That is, if the measurement presents a peak, it would not be taken as the new energy level, but rather, it should remain stable at that new value to be considered the new level of energy consumed. This presents the problem that peaks in the opposite direction can prolong the waiting time. Another method is calculating the average value of all the energy samples from the beginning. This has the problem that the longer the system has been active, the longer it will take for a change in the measured levels to be reflected. That is why moving averages are more useful. In this way, the n last samples obtained during a certain period of time are considered. Depending on the chosen period, we can smooth the energy curve more or less. However, we will also increase the time necessary to calculate, limiting our system's reaction time.

We have conducted tests with different serverless functions to analyse the functions' energy behaviour. The objective is to be able to check if the range of variation of the measurements is different depending on what the function performs. We have used six functions that perform calculations of different complexity, use different amounts of memory, and work locally or make external requests. Disk or database access should not be considered among these parameters since serverless functions do so through external requests. These functions are presented in Table 1. We have chosen them so that they are representative of the different types of functions considering the different combinations of values for these parameters, ranging from the simplest (*f1*) to the most complex (*f6*). Logically, other functions would have a different energy consumption, but what we are interested in is studying if there are differences in the variability in energy consumption depending on the type of processing that the function performs. We must keep in mind that our objective is to compare behaviours; we are not interested in specific energy measurements since these may vary depending on the infrastructure in which we deploy the application, the system load, etc.

The results obtained show that there are differences in the variability of power consumption measurements for each function. The simplest ones present less variability, while the more complex ones that include external requests are more variable. To analyse the differences, we will focus on those that present

the least and greatest variability (see Fig. 1). We can decrease the variability by calculating the moving averages for more extended periods, as shown in Fig. 2.

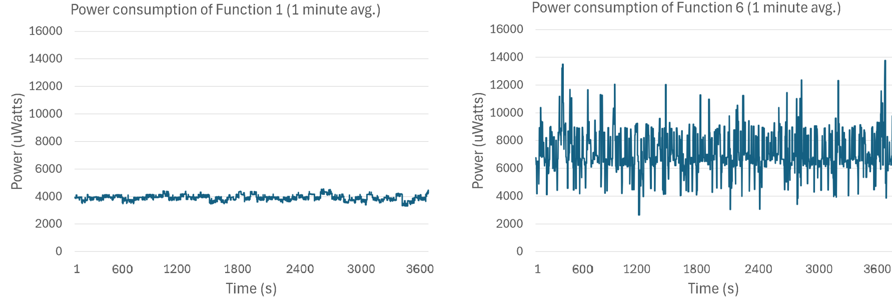

Fig. 1. Power consumption measurements obtained for *f1* and *f6*

We could evaluate an energy objective function for the application to trigger the system's self-adaptation process. However, to be able to react to changes in functions individually, it is more convenient to introduce action policies based on the energy consumed by each function. Once the policy is activated, the optimisation process determines the optimal configuration of the system at that moment. Thus, we can consider a threshold to trigger the self-adaptive process. Since the energy consumed by the functions is variable, it is most appropriate to define them in percentage. The chosen value must be greater than the maximum level of variability of the energy measurement we use since, otherwise, the normal variation of the measurements would unnecessarily launch a self-adaptation process. The maximum variation percentages of the mentioned functions are shown in blue in Fig. 3.

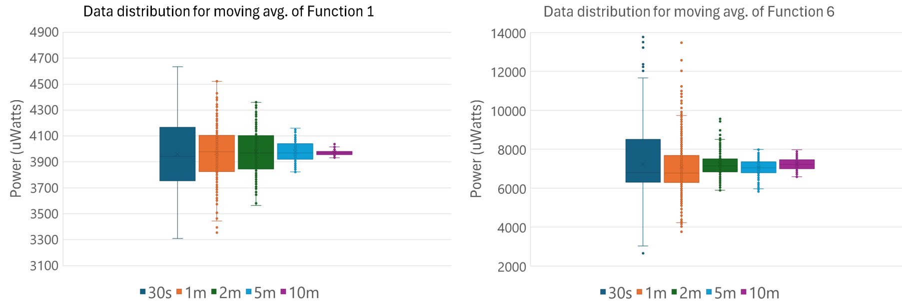

Fig. 2. Moving averages of *f1* and *f6* considering different periods

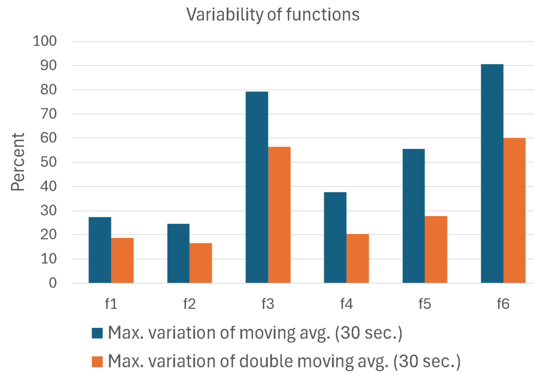

Fig. 3. Variation from average values of the test functions

In the case of the most variable function, the percentages are pretty high, so it would be interesting to improve the measurement method. To do this, we propose using the double moving average, a technique used in other fields of statistical analysis, such as financial analysis, in which it is used to study trends. The results are interesting since they notably smooth the energy curve (see Fig. 4). We can appreciate the reduction in variability by comparing them with the results for the moving average (Fig. 2) of the same functions. Besides, the results are improved compared to the moving average applied for the same total period of time. That is, the percentage variation of the double moving average for t is better than the moving average for *2t*. For example, in our tests, the maximum variation of the moving average of *f6* for 10 minutes was 9.6%, and the maximum variation of the double moving average for 5 minutes was 7.7%. The new maximum variation percentages of the tested functions are shown in orange in Fig. 3.

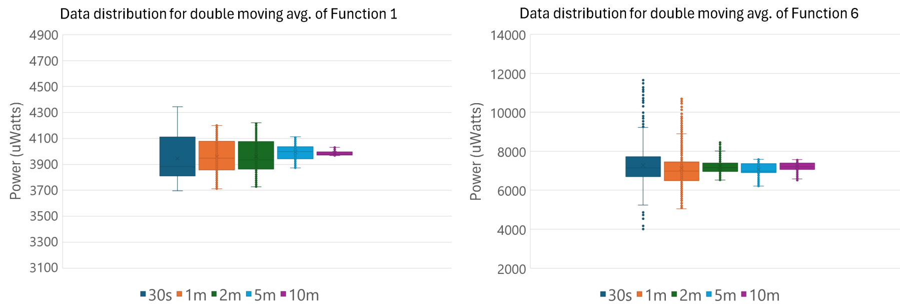

Fig. 4. Double moving averages of *f1* and *f6* considering different periods

5 Building the self-adaptive system

To build our self-adaptive system, we use a framework we previously developed [10]. It is designed to find optimal configurations by considering quality of service requirements. This framework is integrated with OpenFaaS and Scaphandre to constitute our complete system, whose architecture is detailed in Fig. 5.

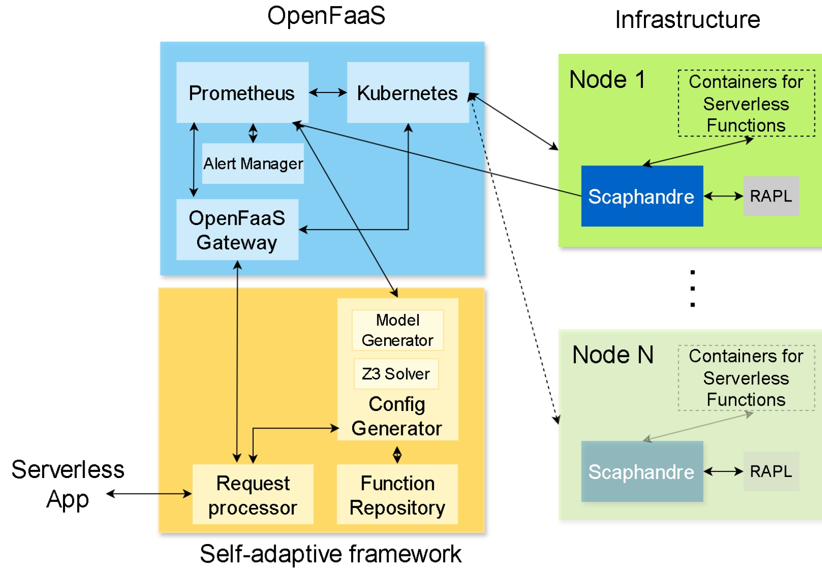

Fig. 5. Architecture of the whole system

The self-adaptive system (bottom left of Fig. 5) analyses the application based on Software Product Lines [7]. It uses information stored in a repository about the application's functions to generate automatically a set of Feature Models [7] that describe the variability of the application. Then, a solver calculates the optimal configuration that allows the system to achieve certain quality of service requirements. In the case of energy, we can combine it with the user experience so that the system chooses the functions that consume the least energy while maintaining a certain level of user experience. When the application requests a function, the system will use the selected configuration to decide which implementation of those available in the repository is actually executed.

The system can also be reconfigured at runtime to work self-adaptively with support for energy measurements, so, we need to add the appropriate rules to achieve it. To avoid overloading the system by continuously measuring the consumption of each function that makes up the serverless application, we have chosen to read this data only after each execution of each function. The data is processed as explained in the previous section, and the values obtained will

be used to compare them with those stored at that moment for that function. Initially, the repository may contain information about the values measured in previous executions or tests of the functions to compare the consumption of the different implementations during the first optimisation process, but this really would not be necessary. since the system updates this information at runtime.

The stored energy values are not updated each time a new measurement is made, but only when these measurements exceed or fall below a certain threshold, calculated as a percentage of that function's currently stored energy value. It is necessary to use thresholds with sufficient margins to determine whether a function has increased or decreased its consumption and whether the variation is due to measurements. A single one could be considered for the entire system, but, as we have verified that the functions can present significant differences in their value range, it is more appropriate to use a different one for each function. To calculate these variable thresholds, we use the same periods as for moving averages, we calculate the average, the maximum and minimum values, and we find the maximum percentage of variation between the average value and said extremes. We consider the greatest of these two results and set the threshold, adding a certain margin. Every time we update the energy values associated with a function, we will also update the considered threshold for that function.

As the period for moving averages, we can choose the one that best suits our needs. The higher it is, the more stable the energy measurements are, and we can use a tighter variation threshold, but it will take longer to react to changes in energy consumption. For example, we can choose 5 minutes for each moving average, so we can adjust thresholds of 15% for the worst case (*f6*) and 5% for the best (*f1*), which is quite acceptable. However, the times used to calculate the second moving average do not have to coincide with those of the first, so variations can also be made in that sense.

6 Case study

We have developed a case study to verify that the system works in a self-adaptive way using the data provided by Scaphandre and the rules we have described to launch the reconfiguration process. As our objective is not to test the operation of the framework, we have chosen a simple case in which only one function varies, so that it is easier to observe the self-adaptation process. It can be extrapolated to a more complex system since the operation of the framework with other systems has been previously tested [10]. The application consists of a system for vehicle control. This controls the passage, and, for security reasons, it takes a photograph of the license plate and another of the driver, which must be stored for some time. The BPMN 2.0 diagram of the case study is shown in Fig. 6. It consists of a step control stage, another for taking photographs and another for storing pictures in a compressed file. To verify the system's response, we will focus on the compression function, for which two alternative implementations with different energy consumption are deployed, as shown in Fig. 7.

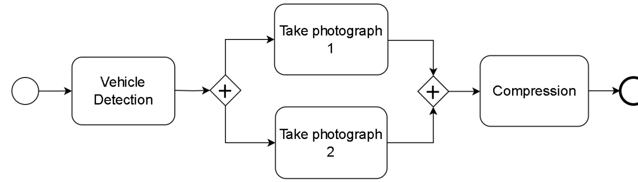

Fig. 6. BPMN 2.0 diagram of the sample application

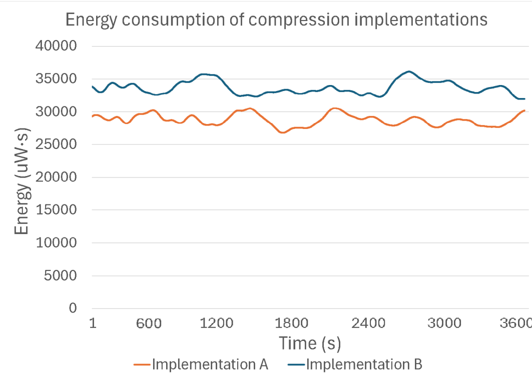

Fig. 7. Energy data obtained for compression functions

The tools used in our experiments are Scaphandre v1.0.0 and OpenFaaS v0.27. Our software is programmed using Python 3, and the hardware used is a PC Intel i5-7400, 3.00 GHz, 24 GiB of RAM. As an interval for the moving averages, we have chosen 5 minutes and adjusted the thresholds automatically for the same period, as explained in the previous section. This assures us that not only is the variability due to the measurements taken into account but also due to the data. As the photographs are of the same size and to similar objects, there is not expected to be much variation due to this. If there were, the system could also be used, but it would only make sense for long sampling periods so that the consumption data could be homogenised and thus be used to compare with that of another function.

To ensure that the measurements are stable, it will be necessary to wait 5 minutes for the first moving average to be correct and 5 more for the double to be correct. That is, we establish a setup time of 10 minutes, until which we will not start any reconfiguration process. When running the test application without starting information about the energy consumption of the functions, the chosen compression function can be any of the two available. We choose the worst case, which is that the application starts running the one with the highest consumption. We observe how, after the setup time has elapsed, the self-adaptation process is launched and the system chooses the alternative implementation, reducing the energy consumption of the application, as shown in Fig. 8 - Case A.

For this case, in which 10 concurrent calls are made continuously for 1 hour, the energy consumption due to functions compared to the estimated consumption that would have been generated if the reconfiguration was not performed, is 5%. It must be taken into account that the difference in consumption of the two implementations considered is not very high, as can be seen in Fig. 7 and now we are considering the whole application. On the other hand, in an application with more functions that present alternatives, the reduction may be greater.

We also reproduce a situation in which the energy consumption of the function increases. The system launches the reconfiguration process after exceeding the threshold previously calculated by the system, which in this case is 9.1% for that function. At that point, it starts running the alternative implementation, and the application's energy consumption is reduced, as can be observed in Fig. 8 - Case B. In this situation the reduction of consumption is about 2%. This reduction is lower because reconfiguration is done in minute 24 and the alternative implementation is the one that previously presented more consumption. However, it is important to keep in mind that these results are absolutely dependent on the specific application and the infrastructure on which it is deployed.

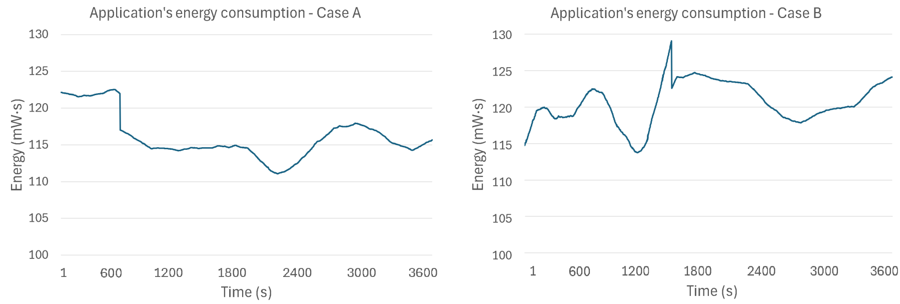

Fig. 8. Execution results for cases A and B

7 Conclusions

We have integrated energy measurement software into a FaaS environment to measure the energy consumed by serverless functions. Due to the high level of variation of the data obtained, some solutions have been proposed to be able to properly compare the energy consumption of various functions at runtime, presenting the improvements achieved. It has also been proven that the variations in the measured values differ depending on the type of function executed. Due to this, the use of variable thresholds has been proposed and a method to calculate them dynamically at runtime for each function has been presented. We have verified that the system presented works appropriately by implementing a case study of a self-adaptive system.

The effect of the infrastructure over the measures is challenging, so, as a line of future work, we plan to add information about the infrastructure on which the functions are executed to the analysis, to make better self-adaptation decisions.

Acknowledgments. Work supported by the *TASOVA PLUS* research network (RED-2022-134337-T), and the projects *IRIS* PID2021-122812OB-I00 (FEDER funds), *DAEMON* H2020-101017109.

References

1. Alhindi, A., Djemame, K., Heravan, F.B.: On the power consumption of serverless functions: An evaluation of openfaas. In: IEEE/ACM 15th International Conference on Utility and Cloud Computing (UCC). pp. 366–371. IEEE (2022)
2. Byrne, A., Pang, Y., Zou, A., Nadgowda, S., Coskun, A.K.: Microfaas: Energy-efficient serverless on bare-metal single-board computers. In: Design, Automation Test in Europe Conference Exhibition (DATE). pp. 754–759 (2022)
3. Ewim, D.R.E., Ninduwezuor-Ehiobu, N., Orikpete, O.F., Egbokhaebho, B.A., Fawole, A.A., Onunka, C.: Impact of data centers on climate change: A review of energy efficient strategies. The Journal of Engineering and Exact Sciences (2023)
4. Intel: RAPL (2012), https://www.intel.com/content/www/us/en/developer/articles/technical/intel-sdm.html
5. Jia, X., Zhao, L.: Raef: Energy-efficient resource allocation through energy fungibility in serverless. In: IEEE 27th International Conference on Parallel and Distributed Systems (ICPADS). pp. 434–441. IEEE (2021)
6. Le, D.N., Pal, S., Pattnaik, P.K.: OpenFaaS, chap. 17, pp. 287–303. John Wiley Sons, Ltd (2022). https://doi.org/10.1002/9781119682318.ch17
7. Lee, K., Kang, K.C., Lee, J.: Concepts and guidelines of feature modeling for product line software engineering. In: Gacek, C. (ed.) Software Reuse: Methods, Techniques, and Tools. pp. 62–77. Springer Berlin Heidelberg (2002)
8. Manotas, I., Bird, C., Zhang, R., Shepherd, D., Jaspan, C., Sadowski, C., Pollock, L., Clause, J.: An empirical study of practitioners' perspectives on green software engineering. In: Proceedings of the 38th International Conference on Software Engineering. p. 237–248. ICSE '16, ACM, New York, NY, USA (2016)
9. Moreno-Vozmediano, R., Huedo, E., Montero, R.S., Llorente, I.M.: Latency and resource consumption analysis for serverless edge analytics. Journal of Cloud Computing **12**(1), 108 (Jul 2023). https://doi.org/10.1186/s13677-023-00485-9
10. Serrano-Gutierrez, P., Ayala, I., Fuentes, L.: Fuspaq: A function selection platform to adjust qos in a faas application. In: Service-Oriented Computing – ICSOC 2022 Workshops. pp. 249–260. Springer Nature Switzerland, Cham (2023)
11. Siddik, M.A.B., Shehabi, A., Marston, L.: The environmental footprint of data centers in the united states. Environmental Research Letters **16**(6), 064017 (2021). https://doi.org/10.1088/1748-9326/abfba1
12. Tsenos, M., Peri, A., Kalogeraki, V.: Energy efficient scheduling for serverless systems. In: 2023 IEEE International Conference on Autonomic Computing and Self-Organizing Systems (ACSOS). pp. 27–36 (2023)
13. Tzenetopoulos, A., Marantos, C., Gavrielides, G., Xydis, S., Soudris, D.: Fade: Faas-inspired application decomposition and energy-aware function placement on the edge. In: Proceedings of the 24th International Workshop on Software and Compilers for Embedded Systems. p. 7–10. SCOPES '21, Association for Computing Machinery, New York, NY, USA (2021). https://doi.org/10.1145/3493229.3493306