Proceedings Paper:
Alhindi, A, Djemame, K and Banaie Heravan, F (2023) On the Power Consumption of Serverless Functions: An Evaluation of OpenFaaS. In: 2022 IEEE/ACM 15th International Conference on Utility and Cloud Computing (UCC). 2022 IEEE/ACM 15th International Conference on Utility and Cloud Computing (UCC), 06-09 Dec 2022, Vancouver, Washington, USA. IEEE, pp. 366-371. ISBN: 978-1-6654-6087-3. ISSN: 2373-6860. EISSN: 2473-7178.

https://doi.org/10.1109/UCC56403.2022.00064

On the Power Consumption of Serverless Functions: An Evaluation of OpenFaaS

Abdulaziz Alhindi
School of Computing, University of Leeds, UK
Qassim University, KSA
sc20aaia@leeds.ac.uk

Karim Djemame, Fatemeh Banaie Heravan
School of Computing, University of Leeds, UK
{K.Djemame, F.BanaieHeravan}@leeds.ac.uk

***Abstract*— The rapid growth in cloud-based technologies has introduced the need for very large data centres to meet the increasing demand for cloud services. One of the main challenges in managing these data centres is the sharp increase of power consumption. Research has therefore tackled the issue of power/energy efficiency in cloud data centres. Serverless computing is a cloud computing execution model that gives software developers the option to deploy their code without the need to configure servers, operating systems or runtime libraries, thus allowing them to invest less effort and capital in infrastructure management. This paper investigates whether serverless computing has the ability to support power efficiency. To this aim, a number of experiments are conducted to compare the power consumption of a serverless platform, OpenFaaS, against Docker containers with the consideration of applications and benchmarks. The experimental results show that OpenFaaS is more power efficient than Docker when the processor and memory are under stress.**

Keywords—Serverless, Power Efficiency, OpenFaaS, Docker, Container, Power Consumption.

I. Introduction

The wide adoption of cloud services and cloud-based technologies like containers and microservices in the IT world has encouraged the emergence of the new execution model, serverless computing or FaaS (Function as a Service). This model exploits container-based virtualization in order to provide environments that enable software developers to deploy their applications and is an ideal solution to build and optimize any Internet of Things (IoT) operations with zero infrastructure and maintenance costs and little-to-no operating expense [1]. The high level of abstraction in serverless platforms is a key feature, which means that the applications are decomposed into fine-grained functions and then deployed and executed with zero control from the developers [2]. Furthermore, in contrast to the rest of cloud services, serverless functions are executed only when they are called, which eliminates the cost of allocating resources [1]. Therefore, serverless computing is viewed as a promising solution to increasing the efficient use of cloud resources and consumption of power, as well as reducing the cost of using cloud services, compared to Virtual Machines (VM) and containers.

The fast development of cloud services has led to the rapid growth in building large data centres. The spending on data centres increased by 35% to reach 41.8 billion dollars in the first quarter of 2021, compared to roughly 30 billion dollars in the same quarter of 2020 [3]. One of the main challenges these large data centres need to tackle is the increasing amount of power consumed by the infrastructure. In 2016, data centres in the United States consumed more than 90 billion kilowatt-hour (KWh) of electricity. Likewise, power consumption of data centres around the world amounted to about 416 terawatt-hour (TWh) in 2015, or roughly 3% of all electricity generated on the planet [4,22]. This power consumption is expected to sharply increase to more than 3300 TWh in 2025, compared to 2010 when this figure was close to 189 TWh [5].

The IT equipment, which refers to the equipment used to process, route, store or manage data (compute, storage, network resources) account for 50% of the power consumed in one data centre where servers may consume 40% of the IT equipment's share [6]. Therefore, improving the power consumption of the cloud services running on such infrastructures can lead to a significant reduction in the power consumed by data centres.

The primary purpose of this paper is to evaluate the efficiency of the power consumption of one of the well-known serverless platforms, OpenFaaS [24]. The examination is based on the comparison of OpenFaaS and Docker containers power consumption with the consideration of two applications that stress either the Central Processing Unit (CPU) or the Random Access Memory (RAM). The comparison is conducted on two different architectures : **1)** the two tested platforms, OpenFaaS and Docker, run as stand-alone platforms. **2)** the two tested platforms are deployed on Kubernetes. The main difference between the two architectures is that Kubernetes allows OpenFaaS functions and Docker containers auto-scaling, which lets us examine the power efficiency of OpenFaaS in such setting.

The contributions of this paper are:

1. the identification of a relevant benchmark for evaluating the power consumption of OpenFaaS as a common serverless platform;
2. a quantitative analysis of the power consumption of OpenFaaS considering a defined benchmark;
3. a set of recommendations for effective adoption of the serverless computing for energy efficiency.

II. Related Work

There have been a few research papers studying the power efficiency of serverless computing, as this field of research is still new and serverless technology is rapidly changing. The work by Xuechao Jia and Laiping Zhao proposes a solution called RAFE [7]. This solution presents a new mechanism for allocating the resources required by serverless functions in order to minimize power consumption. The study shows that this solution can reduce the power consumption of serverless functions by a noticeable per cent, 21.2%. They found that serverless platforms currently do not take power consumption as an

effective factor in making the decision of resource allocation. A new term named *power fungibility* is therefore introduced which means it is possible to reduce the power consumption of serverless functions without affecting the required latency. This study confirms that serverless functions always go through three stages in their life cycle: startup, runtime, and idle. So, if allocating resources in each one of these stages is based on power-aware decisions, power consumption of serverless functions could be reduced.

To achieve their goal, the authors investigated the power consumption of one of the common serverless platforms, OpenFaaS, and they choose Kubernetes as the platform for managing and orchestrating the functions' containers. They concluded that the power consumed in the first stage, *startup*, is mainly by a number of operations for getting the functions ready to run such as loading the containers' images and the required libraries. The second stage, *runtime*, is divided into two parts: the power consumed by the platform, OpenFaaS, and the power consumed by the running functions. In this stage, the results demonstrate that the components of OpenFaaS consume a tiny amount of power compared with the functions. Also, the authors confirmed that the largest amount of the energy consumed by the functions was during the second stage. The final stage, *idle*, when the functions are alive and waiting for a request to execute, was found to consume much less power compared to the other two stages.

This paper focuses on analysing the power consumption during the second stage of OpenFaaS functions.

The measurement of power consumption can be obtained by using several methods. One of these methods is to use hardware devices known as Wattmeters, which is the method used in [8] to measure the power consumption of VMs. However, this method is not applicable in cloud systems due to their shared infrastructure [9]. On the other hand, there are several tools developed for measuring power consumption such as PowerTop [12] and Powerstat [10]. Powerstat is open-source software designed for measuring power consumption by using the Advanced Configuration and Power Interface (ACPI) to read the amount of power consumed from the machine's battery [10]. This tool was used in [11] to measure the power consumption of a laptop running Linux OS. Similarly, PowerTop is an open-source software measuring power consumption by either reading the battery information, or using a mathematical model to estimate the power consumed by each process [12]. This tool is the chosen one in [9] for measuring the power consumption of running processes on Linux OS. In this paper, we chose Powerstat to measure power consumption in our experiments as it completely depends on the readings taken from the machine's battery, which makes the accuracy of the measured power consumption close to the accuracy of Wattmeters. Moreover, this tool is simple to use and its results are more readable compared with PowerTop.

III. OpenFaaS Architecture

OpenFaaS is an open-source serverless framework under an independent project managed and hosted by the company OpenFaaS Ltd. Each function deployed on this platform is created and managed as a immutable container. This platform consists of several basic components such as a gateway, auto-scaling system, and invocation tools, see Fig. 1. OpenFaaS can be deployed on Kubernetes or on a bare-metal environment by using *faasd* which is an alternative platform enabling OpenFaaS to manage functions without Kubernetes.

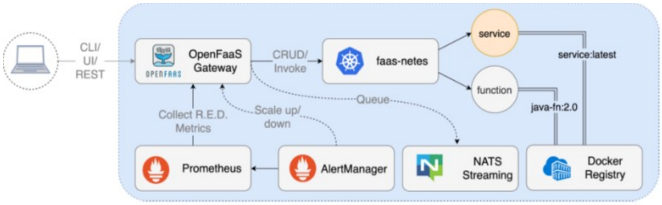


Fig. 1. OpenFaaS Architecture

Faasd uses *Containerd* [27] to containerize functions while Docker is the container runtime used with Kubernetes [24].

Faasd is developed and maintained by the OpenFaaS project and it is designed to run on a single machine equipped with modest hardware. However, this platform cannot scale functions and the maximum number of each function's replicas is one [24]. The main purpose of creating faasd is similar to the purpose of creating the "Standalone" version of OpenWhisk [24,25].

Although OpenFaaS functions on Kubernetes share the same structure with the other side in the comparison, Docker, the difference comes from the sophisticated way that OpenFaaS follows in orchestrating functions' containers. On the other hand, Docker containers on Kubernetes are scaled and managed by the load balancer component of Kubernetes.

IV. Experimental Design

The goal of the experiment is to measure the power consumption of OpenFaaS and Docker containers during the execution of two chosen applications. The collected power measurements will be used to evaluate the efficiency in consuming power of the two tested platforms.

A. System Architecture

The architecture of the experiment consists of three components and the whole experiment was conducted on a single computer, as shown in Fig. 2.

1) The Tested Platforms: are OpenFaaS and Docker, each one of which is installed inside a different VM. The two benchmarking applications, see section IV.B, are also deployed on each platform. The tested platforms were run inside VMs to provide the flexibility for supporting various scenarios in a similar environment to cloud systems. VirtualBox [13] is the hypervisor used in the experiment.

2) The Power Measurement Tool: Powerstat was run outside the VMs to measure the power consumed by the physical machine hosting the VMs as it needs access to the kernel of the main Operating System (OS).

3) The Workload Tool: Jmeter [14] was run outside the VMs in order to generate workload for the tested platforms in the form of HTTP requests.

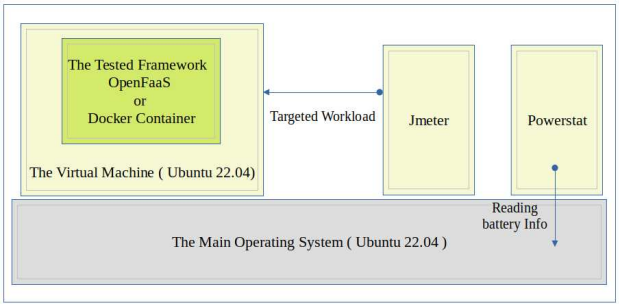


Fig. 2. The experiment's components

B. The Benchmarking Applications

Power consumption is usually caused by using different resources such as CPU, memory, storage and networks. In a computer system, the CPU is considered to be the first consumer of power and is followed by the memory [15]. The CPU and the memory are therefore the most important resources to be evaluated in terms of performance in the context of applications execution [16]. Consequently, they are also important resources to consider when measuring power consumption.

In the experiment, two different benchmarking applications, written in Python, are chosen to stress the CPU and RAM. Both of these applications act as a simple web server to process the workload generated as HTTP requests by Jmeter.

1) Matrices Multiplication: This benchmarking application consists of the multiplication of two square matrices with a size of 5000. The application is used to stress the memory more than any other hardware component [16]. There are two versions of this benchmark that were written to run as an OpenFaaS function and a Docker container, the source code of which can be found in [23] under the names "Matrices-Function-Benchmark.zip" and "Matrices-Docker-Benchmark.py".

2) Prime Numbers Generation: This benchmarking application is designed to generate a list of prime numbers with a size of 1500, which, in turn, stresses the CPU more than other resources. This application is extensively used in the literature to evaluate the performance of the CPU; the code is just a simple loop searching for prime numbers [17]. There are two versions of this benchmark that were written to run as an OpenFaaS function and a Docker container, the source code of which can be found in [23] under the names "PrimeNumbers-Function-Benchmark.zip" and "PrimeNumbers-Docker-Benchmark.py".

C. Scenarios

There are four scenarios in the experiment, each one of which begins with a waiting time of 120 seconds that is required by Powerstat to let the physical machine settle down. Powerstat will then start measuring the power consumed by the physical machine while Jmeter generates workload onto one of the tested platforms, which lasts for 240 seconds. This period of time is defined as *testing time.* During this time, Jmeter generates intensive workload by performing three load tests to ensure more accurate results, with 10 seconds of waiting before starting each one. A Bash script is used to automate the experiments, e.g. running Powerstat and Jmeter tools, parameters setting and collection of results. The source code of this script can be found in [23] under the name of "powerstat-jmeter.bash".

The four scenarios, shown in Fig. 3, are described next:

1) Docker containers: The goal of this scenario is to measure the power consumption of a Docker container running one of the two benchmarking applications, so each one of these applications is run inside a different container. This scenario was conducted on a different VM.

2) OpenFaaS with faasd: In this scenario, the power consumption of OpenFaaS running as a stand-alone platform is measured while it is under a given load. OpenFaaS with *faasd* was installed in a different VM with two OpenFaaS functions created to run the two benchmarking applications, each function running one application.

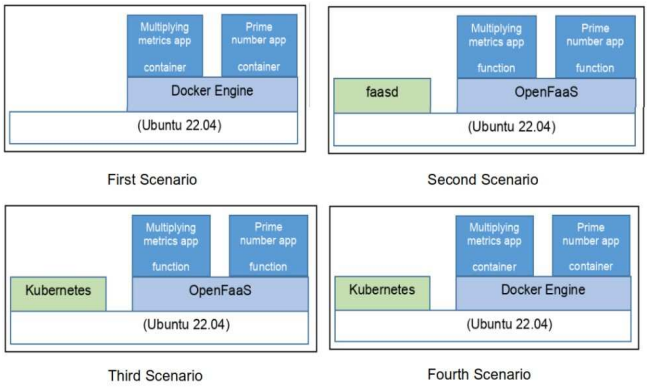


Fig. 3. The four scenarios

3) OpenFaaS on Kubernetes: This scenario aims to measure the power consumed by OpenFaaS deployed on Kubernetes instead. OpenFaaS was installed on Kubernetes in a different VM with two OpenFaaS functions created for the two benchmarking applications.

4) Docker containers on Kubernetes: In this scenario, two Docker containers were deployed on Kubernetes with the auto-scaling feature in a different VM. Enabling this feature aims to make this scenario comparable with the previous one because OpenFaaS with *faasd* does not scale the deployed functions but OpenFaaS on Kubernetes does. Two Kubernetes components are essential for this feature and were installed on the VM of this scenario. The first one is *Kubernetes Metrics Server* that provides the resources usage of containers, such as the CPU usage, to the controller *HorizontalpodAutoscaler* [18]. The second one is the load balancer which is a Kubernetes service, named *LoadBalancer*, whose task is to balance the coming workload between the pods of the targeted application [19]. The LoadBalancer implementation *MetalLB* was used in the experiment as it offers network load balancing on Kubernetes clusters running on bare-metal environments [20]. The two YAML files for deploying the two benchmarking applications in this scenario can be found in [23] under the names of "new-Matrices-Benchmark-Kubernetes.yaml" and "new-PrimeNumbers-Benchmark-Kubernetes.yaml".

D. Workload

Jmeter generates workload using a user-defined plan that contains the following parameters: *number of threads (users)* that is set to 5, r*amp-Up period* that is set to 0, and *loop count* which is set to 40. The values of these parameters were chosen based on the limitations of the physical machine's hardware, as increasing these values will cause the tested platforms to run out of the memory. Moreover, each HTTP request contains an HTTP parameter defining the size of either the matrices or the list of the prime numbers. In terms of the workload sent to the matrices multiplication application, the value of the HTTP parameter is 5000, while it is 1500 in case of the prime numbers generation application.

E. Evaluation Metrics

The following metrics are considered for measuring power consumption:

1) Power Consumption (P): refers to the average power consumption of the physical machine during *testing time*. This metric is calculated as:

$$P = \frac{\sum_{1}^{T} Watt_e}{T}$$

T refers to the length of *testing time* which is 240 seconds, while $Watt_e$ refers to the Watt value taken in every second sampling during *testing time*.

2) Idle Power (I): refers to the average power consumption of the physical machine while the VM of the tested platform is idle.

$$I = \frac{\sum_{1}^{T} Watt_i}{T}$$

T is equivalent to the length of *testing time,* 240 seconds, because this metric is for showing the VM's idle power before starting the load tests. $Watt_i$ refers to the Watt value taken in every second sampling while the VM is idle.

3) Estimated Power Consumption (EP): refers to the average power consumption of the tested platform.

$$EP = P - I$$

This metric could show the average of the actual power consumed by the tested platform.

4) Execution time (E): means the period of time a tested platform spent on completing all three load tests without considering the idle time. This metric is obtained by monitoring the CPU to measure the time the CPU spends in processing the incoming requests during the load tests.

5) Energy (N): refers to the estimated energy consumed by a tested platform.

$$N = \sum_{1}^{T} Watt_e - Watt_i$$

T refers to the length of *testing time,* 240 seconds.

F. Implementation

The experiment was conducted on a laptop with the following hardware specifications: the CPU is AMD Ryzen 7 5800h with 8 cores and base clock 3.2GHz, the memory is 16 GB, the main OS is Ubuntu 22.04 LTS, and the storage is 512 GB SSD.

At the beginning of the implementation of each scenario, the VM's idle power is recorded. The results of the implementation of each scenario were then collected and saved as three CSV files containing: **1)** the idle power of the scenario's VM, **2)** the power consumption while the workload generated by Jmeter was sent to the matrices multiplication application, and **3)** the power consumption while the Jmeter workload was sent to the prime numbers generation application. The results of the experiments are analysed and discussed next.

V. Results and Discussion

A. Idle Power

In terms of idle power, the physical machine consumed on average 14.5 Watts before running any VM. The idle power of the VMs in the four scenarios are: 17.8, 20.2, 20.5, and 22.8 Watts respectively, as shown in Fig. 4.

B. Measurement Results

The final results, shown in Fig. 5, demonstrate that OpenFaaS with *faasd* is 12% more efficient in consuming power than Docker containers when running the benchmarking application that puts intensive loads on the RAM. However, running the same benchmarking application, Docker containers are 17% more efficient when both of the tested platforms are deployed on Kubernetes with the auto-scaling feature.

The OpenFaaS functions were scaled up to 5 replicas and then down to 1 replica twice during the *testing time,* while the Docker containers were scaled up to 5 replicas and never scaled down until the end of the *testing time.* As Jmeter performed three load tests, Fig. 5 shows the average of *execution time* for each load test.

On the other hand, OpenFaaS is significantly more efficient in consuming power when the benchmarking application puts high loads on the CPU. As stand-alone platform, OpenFaaS with *faasd* is 58% more efficient in consuming power than Docker when running this application. Moreover, when both of the tested platforms were deployed on Kubernetes, OpenFaaS was 27% more efficient even though the OpenFaaS functions were scaled up to 5 replicas and then scaled down to 1 replica during the *testing time* while the Docker containers were scaled up to 5 replicas and kept this number until the end of the experiment.

C. Findings

Fig. 6a shows that stand-alone OpenFaaS consumed less power and far less time than the Docker container to complete all the three load tests of the multiplying matrices application that stresses the RAM. On the other hand, Fig. 6b shows that Docker containers on Kubernetes consumed less power and less time to complete the three load tests of this benchmarking application, but there are other factors that need attention. The OpenFaaS functions were scaled up and down twice during completion of the load tests, while the container's replicas were kept alive until the end of the *execution time*. Moreover, Fig. 6b shows that Docker containers spent a longer time on the first load test compared with the next two tests because increasing the replicas of the Docker container from 1 to 5 took place in this period of time. In contrast, the periods of time spent by OpenFaaS on Kubernetes to complete the three load tests seem close to each other in their lengths. Therefore, it can be concluded that OpenFaaS was more efficient in using resources and faster in scaling than Docker containers, which could increase the efficiency in consuming power in cloud systems where a big number of user applications run on one server.

In terms of the other benchmarking application that stresses the CPU, Fig. 7a demonstrates that OpenFaaS was remarkably more efficient in consuming power, 58%, and 63% faster during completing the three load tests. In addition, it is clear from Fig. 7b that OpenFaaS on Kubernetes was more efficient in consuming power and faster in completing the three load tests, with percentages reaching 27 and 37 respectively. Also, this figure shows that the first load test of Docker containers needed notably a longer time than the next ones because, during this test, the container's replicas were scaled up from 1 to 5 and then these five replicas were kept alive during the rest of *execution time*. In contrast, scaling the OpenFaaS function up to 5 and down to 1 replica did not leave a clear trend for OpenFaaS in Fig

7b. This suggests that that OpenFaaS functions are clearly more lightweight and more power efficient in scaling.

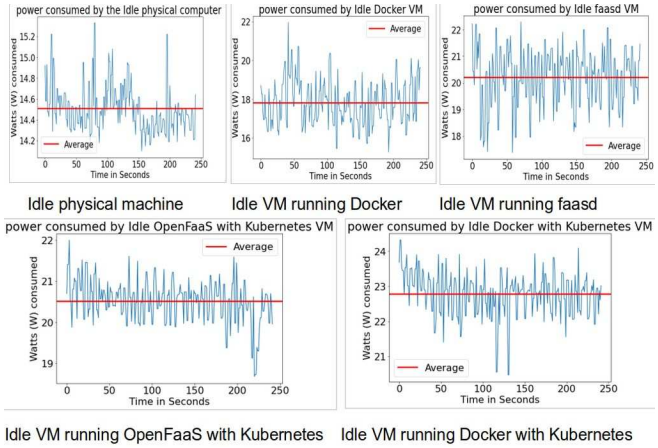
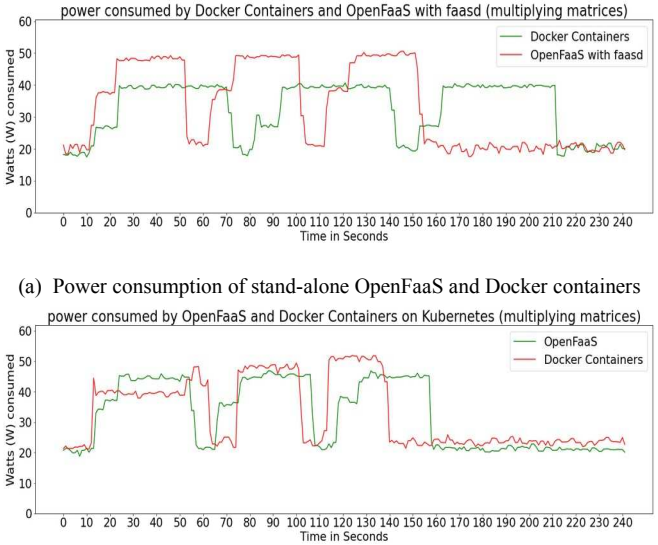


Fig. 4. Idle power for the physical and virtual machines

Benchmark	Metrics	Stand Alone		Deployed on Kubernetes	
		Docker Containers	OpenFaaS with faasd	OpenFaaS	Docker Containers
Multiplying Matrices Benchmark	Estimated Power Consumption (EP)	14.8 Watts	13 Watts	11.8 Watts	9.8 Watts
	Standard Deviation of EP	8.65	13.15	11.02	11.2
	Execution time (E)	178 seconds	121 seconds	125 seconds	107 seconds
	Energy (N)	3552 Joule	3110.4 Joule	2820 Joule	2356.8 Joule
	Avg. E each load test	60 seconds	40 seconds	41.7 seconds	35.7 seconds
	Was scaled?	No	No	Yes	Yes
	EP comparison	faasd consumed 12% less		Docker consumed 17% less	
Prime numbers benchmark	Estimated Power Consumption (EP)	16.1 Watts	6.7 Watts	6.5 Watts	9 Watts
	Standard Deviation of EP	7.63	10.75	8.64	10.04
	Execution time (E)	198 seconds	73 seconds	74 seconds	117 seconds
	Energy (N)	3859.2 Joule	1608 Joule	1548 Joule	2162.4 Joule
	Avg. E each load test	66 seconds	24.3 seconds	24.7 seconds	39 seconds
	Was scaled?	No	No	Yes	Yes
	EP comparison	faasd consumed 58% less		FaaS consumed 27% less	

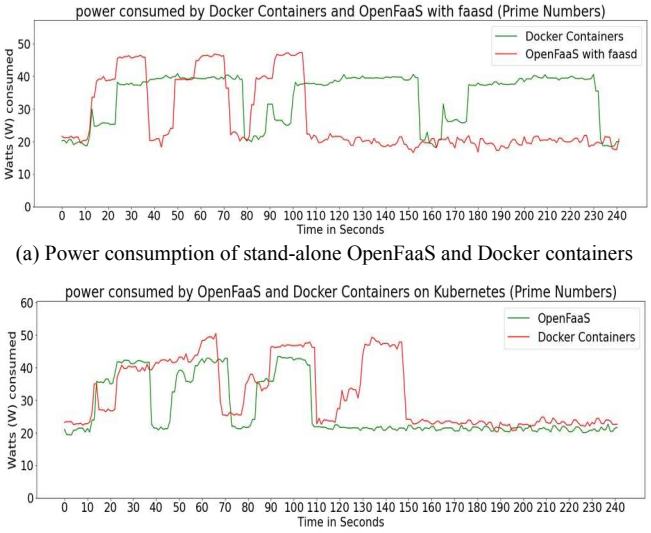
Fig. 5. The results of both tested platforms with the relates statistics



(a) Power consumption of stand-alone OpenFaaS and Docker containers

(b) Power consumption of OpenFaaS and Docker containers on Kubernetes

Fig. 6. Power consumption of the multiplying matrices application



(a) Power consumption of stand-alone OpenFaaS and Docker containers

(b) Power consumption of OpenFaaS and Docker containers on Kubernetes

Fig. 7. Power consumption of the generating prime numbers application

D. Recommendations

The results of this study confirm that OpenFaaS can be more efficient in power consumption when it is run as a stand-alone platform, compared to Docker containers. This superiority of OpenFaaS can be seen clearly when the main task of the OpenFaaS functions does not cause heavy loads on the memory. On the other hand, on Kubernetes, Docker containers showed better figures in power consumption when the executed code induces massive loads on the memory, compared with OpenFaaS. However, on Kubernetes, OpenFaaS could be a much better option if the main task of the functions uses the CPU significantly more than the memory.

When an application deployed on Kubernetes is expected to receive heavy workloads and needs to be scaled up and down repeatedly, OpenFaaS functions are expected to consume less power than Docker containers. Likewise, OpenFaaS could be a far better option in environments where better resource utilization leads to a decrease in power consumption as the results confirm that OpenFaaS functions are more effective in using resources than Docker containers.

VI. Conclusion and Future Work

To sum up, this paper evaluates the power efficiency of one of the serverless platforms, OpenFaaS, by comparing its power consumption with that of Docker containers. To achieve this goal, two benchmarks are developed to generate intensive loads on the CPU and RAM. The power consumed by these two platforms while executing the two benchmarking applications was measured using Powerstat. The final results of this experiment showed that OpenFaaS was more efficient in power consumption when it was run as stand-alone. However, when OpenFaaS and Docker containers were deployed on Kubernetes, OpenFaaS was more efficient in consuming power when intensive workloads target the CPU. On the other hand, Docker containers consumed less power than what OpenFaaS did when the memory was targeted by heavy workloads.

Future work aims at realizing the concept of modular Software Defined Networking (SDN) based on serverless functions with the goal to implement a novel platform to reduce the energy consumption of applications deployment and operation on the Internet [21,26].

VII. Acknowledgments

The authors would like to thank the European Next Generation Internet Program for Open INTErnet Renovation (NGI-Pointer 2) for supporting this work under contract 871528 (EDGENESS Project).

References

[1] P. Castro, V. Ishakian, V. Muthusamy, and A. Slominski, "The rise of serverless computing," *Communications of the ACM*, vol. 62, no. 12, pp. 44–54, 2019.

[2] I. Baldini, P. Castro, K. Chang, P. Cheng, S. Fink, V. Ishakian, N. Mitchell, V. Muthusamy, R. Rabbah, A. Slominski, and P. Suter, "Serverless Computing: Current Trends and open problems," Research Advances in Cloud Computing, pp. 1–20, 2017.

[3] "Global Cloud Services Market Q1 2021," *Canalys Newsroom*. [Online]. Available: https://www.canalys.com/newsroom/global-cloud-market-Q121. [Accessed: 16-Sep-2022].

[4] D. R. Danilak, "Council post: Why energy is a big and rapidly growing problem for data centres," *Forbes*, 15-Dec-2017. [Online]. Available: https://www.forbes.com/sites/forbestechcouncil/2017/12/15/why-energy-is-a-big-and-rapidly-growing-problem-for-data-centres6-Sep-2022].

[5] A. Anders. "Total Consumer Power Consumption Forecast," *Presentation at the Nordic Digital Business Summit, Helsinki, Finland*, 2017.

[6] H. Cheng, B. Liu, W. Lin, Z. Ma, K. Li, and C.-H. Hsu, "A survey of energy-saving technologies in cloud data centres," *The Journal of Supercomputing*, vol. 77, no. 11, pp. 13385–13420, 2021.

[7] X. Jia and L. Zhao, "RAEF: Energy-efficient resource allocation through energy fungibility in serverless," *2021 IEEE 27th International Conference on Parallel and Distributed Systems (ICPADS)*, 2021.

[8] Q. Chen, P. Grosso, K. van Veldt, C. de Laat, R. Hofman, and H. Bal, "Profiling energy consumption of VMS for green cloud computing," *2011 IEEE Ninth International Conference on Dependable, Autonomic and Secure Computing*, 2011.

[9] I. M. Murwantara and B. Bordbar, "A simplified method of measurement of energy consumption in cloud and virtualized environment," *2014 IEEE Fourth International Conference on Big Data and Cloud Computing*, 2014.

[10] ColinIanKing, "Powerstat," *GitHub*. [Online]. Available: https://github.com/ColinIanKing/powerstat. [Accessed: 16-Sep-2022].

[11] M. A. Abou-Of, A. H. Taha, and A. A. Sedky, "Trade-off between low power and energy efficiency in benchmarking," *2016 7th International Conference on Information and Communication Systems (ICICS)*, 2016.

[12] fenrus75, "Linux PowerTOP Tool," *GitHub*. [Online]. Available: https://github.com/fenrus75/powertop. [Accessed: 16-Sep-2022].

[13] "Virtualbox," *Oracle VM VirtualBox*. [Online]. Available: https://www.virtualbox.org/. [Accessed: 16-Sep-2022].

[14] "Apache JMeter." [Online]. Available: https://jmeter.apache.org/. [Accessed: 16-Sep-2022].

[15] X. Zhang, Z. Shen, B. Xia, Z. Liu, and Y. Li, "Estimating power consumption of containers and virtual machines in data centres," 2020 IEEE International Conference on Cluster Computing (CLUSTER), 2020.

[16] K. Djemame, M. Parker, and D. Datsev, "Open-source serverless architectures: An evaluation of apache OpenWhisk," *2020 IEEE/ACM 13th International Conference on Utility and Cloud Computing (UCC)*, 2020.

[17] A. Pereira Ferreira and R. Sinnott, "A performance evaluation of containers running on managed Kubernetes Services," *2019 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, 2019.

[18] "Horizontal pod autoscaling," *Kubernetes*, 10-Jun-2022. [Online]. Available: https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/. [Accessed: 16-Sep-2022].

[19] "Kubernetes load balancer," *Kubernetes*, 23-Mar-2021. [Online]. Available: https://kubernetes.io/docs/tasks/access-application-cluster/create-external-load-balancer/. [Accessed: 16-Sep-2022].

[20] "Metallb," *MetalLB, bare metal load-balancer for Kubernetes*. [Online]. Available: https://metallb.org/. [Accessed: 16-Sep-2022].

[21] K. Djemame, "Energy efficiency in edge environments: A serverless computing approach," *Economics of Grids, Clouds, Systems, and Services*, pp. 181–184, 2021.

[22] "How liking something on Facebook can damage the planet," *The Independent*, 23-Jan-2016. [Online]. Available: https://www.independent.co.uk/climate-change/news/global-warming-data-centres-to-consume-three-times-as-much-energy-in-next-decade-experts-warn-a6830086.html. [Accessed: 20-Oct-2022].

[23] Aalhindi, "AALHINDI/MSPROJECT2022-a.alhindi," *GitHub*. [Online]. Available: https://github.com/aalhindi/MsProject2022-a.alhindi. [Accessed: 21-Oct-2022].

[24] O. F. S. Authors, "Introduction," *OpenFaaS*. [Online]. Available: https://docs.openfaas.com/. [Accessed: 26-Oct-2022].

[25] *Documentation*. [Online]. Available: https://openwhisk.apache.org/documentation.html. [Accessed: 26-Oct-2022].

[26] F. Banaie Heravan and K. Djemame, "A Serverless Computing Platform for Software Defined Networks," *Economics of Grids, Clouds, Systems, and Services (GECON'2022)*, 2022.

[27] "Containerd," *containerd*. [Online]. Available: https://containerd.io/. [Accessed: 31-Oct-2022].