Acta Scientiarum
http://periodicos.uem.br/ojs
ISSN on-line: 1807-8664
Doi: 10.4025/actascitechnol.v46i1.65769

Extending the Capacity of IoT Devices with Serverless Services for a Face Detection Application: A practical evaluation

João Pedro Bach Dotta, Guilherme Galante* and Marcio Seiji Oyamada

Departamento de Ciência da Computação, Universidade Estadual do Oeste do Paraná, Rua Universitária, 1619, 85819-110, Cascavel, Paraná, Brazil. *Author for correspondence. E-mail: guilherme.galante@unioeste.br

ABSTRACT. IoT systems have limitations and challenges in many ways, such as performance, energy consumption, memory footprint and data storage. One possible solution to minimize these limitations is to integrate IoT devices with the serverless cloud computing model. In this context, this work aims to analyze the integration of these two technologies to determine if it can help overcome the limitations of IoT devices, most notably energy consumption and performance. A face detection application was adapted to run on an IoT device and the AWS Serverless Lambda service. Through experimentation, some metrics like latency, execution time and energy consumption were collected to evaluate the impact of serverless technology on the scope of IoT systems development. Results show that serverless technology is superior to the IoT device used in terms of performance, but it is directly related to the quality of the network connection to which the device is connected. Serverless technology can also be very useful when power supply to an IoT device is an issue, as these devices are usually battery-connected and have limited power source.

Keywords: IoT; serverless; energy consumption; network performance.

Received on November 16, 2022.
Accepted on August 22, 2023.

Introduction

The Internet of Things (IoT) and its associated applications have gained massive popularity recently. An IoT network consists of smart and connected devices called "*things*". The composition of an IoT system can be very heterogeneous, considering the variety of devices that can be used in it. It usually consists of things that monitor an environment using sensors and can interact with the environment using actuators.

The vast majority of these things that integrate IoT systems operate under severely constrained resources, such as limited battery life, computing power, and small memory capacity. Because of these constraints, in most cases, the resources provided by IoT devices are not sufficient to directly host complex applications (Puliafito, Mingozzi, Longo, Puliafito, & Rana, 2019).

In light of these limitations, the development of IoT systems with all their requirements such as security, privacy, availability, scalability, performance, etc. can be very challenging. One possibility to mitigate these limitations is to extend the capacity of devices by using fog and cloud computing, more specifically using the service model called serverless. The integration of serverless and IoT enables resource-constrained IoT devices to move data or complex computations to the cloud and leverage its computing and storage capacity. The use of serverless model is interesting as it abstracts implementation complexities, thereby simplifying the seamless integration between IoT devices and cloud services.

This work addresses the above issues and aims to develop a face detection application that uses serverless computing in the context of IoT, and analyze how these two technologies can be advantageously integrated and in scenarios when such integration is worthwhile. The experiments conducted in this work show that the integration of these two technologies is possible and can overcome the limitations of IoT devices. Results show that serverless technology is superior to the IoT device in terms of performance, but it is directly related to the quality of the network connection to which the device is connected. Serverless technology can also be very useful when power supply to an IoT device is an issue, as these devices are usually battery-connected and use this limited power source.

Related work

This section describes the related work, focusing on initiatives that explore the combination of IoT and serverless technologies.

Kjorveziroski, Filiposka, and Trajkovik (2021) present a systematic review that examines and describes the current state of serverless research as it relates to IoT, and outlines the open issues. They analyze papers published between January 2015 and September 2021. As conclusion, the authors argue that IoT has the potential to become an important use case for serverless computing, however, there are a number of issues that remain unsolved, including scheduling, migration, performance improvement, cold start elimination, and security and isolation.

Cassel et al. (2022) present a comprehensive systematic literature review that includes 60 papers on serverless computing for IoT. The authors provide insights into how functions are offloaded to different devices and how they interact with each other. They also present a taxonomy that summarizes all the features, as well as a discussion of the overall architecture of serverless applications for IoT.

Benedetti, Femminella, Reali, and Steenhaut (2021) explore the suitability of serverless computing for deploying IoT-based services. They analyze some serverless computing technologies (e. g., Kubeless, OpenFaaS and OpenWhisk) that can be used for IoT services, and then present the implementation of a serverless platform which includes typical IoT service elements. Performance is evaluated in terms of resource consumption and latency considering cold and warm start scenarios.

Benomar, Longo, Merlino, and Puliafito (2021) present the Deviceless approach to explore the serverless concept in an IoT scenario. In this approach, an IoT infrastructure consisting of devices deployed at the network edge can be seamlessly integrated as an application execution infrastructure to enable interactions with hosted sensors/actuators. The authors present a preliminary prototype based on the Stack4Things framework and OpenStack that are validated using two experiments. A first use case is related to the instantiation of data pipelines using a distributed IoT infrastructure. In the second use case, an IoT deviceless system is used to mount a virtual file system over serverless.

More recently, Esposito, Belli, Palma, and Pierleoni (2023) propose and evaluate a framework that represents a synergistic union of different cloud-computing and IoT-related technologies that can enable accessible and automated home environments. The framework makes use of a messages-exchange protocol based on MQTT and cloud-deployed serverless functions. A smart object, namely a smart kitchen fan extractor system, was used to illustrate the viability of the solution.

A paper with a similar objective is presented by Großmann, Ioannidis, and Le (2019). The authors investigate the merits of integrating serverless computing into IoT device deployments in terms of easier manageability, multi-architecture support and performance. Experiments were conducted using the OpenFaaS platform and a data flow application.

The goal of our work is to analyze how both technologies can be integrated to minimize the limitations of IoT devices and in which cases this integration is worthwhile, leveraging metrics like latency, execution time and energy consumption. Other important contribution is the execution of experiments using a serverless service in the public cloud (AWS Lambda) instead of using on-premise platforms.

Background

In this section, we introduce the key concepts related to development of the research.

IoT

Sensors, controllers, actuators, and any other device that can connect to the Internet are called things. Recent advances in wireless communication, ultra-low power processors, sensors, actuators, and cloud/edge/fog computing have enabled the development of several IoT solutions in areas such as smart agriculture, smart cities, smart factory, logistics and transportation, health, and others (Firouzi, Farahani, Weinberger, DePace, & Aliee, 2020).

Whitmore, Agarwal, and Xu (2015) define the IoT as a paradigm in which things can be equipped with identifying, sensing, networking, and processing capabilities that enable them to communicate with each other and with other devices and services over the Internet to achieve a specific goal. In this way, things can be used to monitor their environment, interact with people and other things, and perform all sorts of tasks with little or no human intervention (Agudelo-Sanabria & Jindal, 2021).

An IoT solution is composed of different layers. The sensing layer is the lowest level and includes the sensors and actuators. Due to restrictive requirements, this layer usually consists of ultra-low power processing elements with low processing capacity. Above the sensing layer, communication is performed by the gateways, which collect and send data to the Internet. The collected data is sent to the fog and cloud layers. Data processing can be done by all these different layers and depends on factors such as latency and available resources. The closer we are to the sensing layer, the less computational resources and data from the environment are available, but the latency can be lower. In another extreme case, in the cloud, we have a lot of data and also computing resources available, but on the other hand we have to consider the communication latency.

Serverless

According to Eismann et al. (2021) serverless computing refers to platforms that hide server usage from users and run code on demand. The cloud provider takes care of scaling and configuration the resources, and users only have to pay for the use of their applications. Serverless computing is spreading rapidly among many cloud providers and is powering an increasing number of mobile and IoT applications (McGrath & Brenner, 2017).

To implement this computational model, Serverless includes Function as a Service (FaaS) and Backend as a Service (BaaS) technologies (Jonas et al., 2019). FaaS allows developers to deploy small source code snippets that run in an isolated environment when triggered. These functions are not constantly active and have a short execution time (few minutes). Instead, FaaS platforms listen for events (such as incoming HTTP requests or data added to a storage service) that instantiate the functions. Thus, the functions execute on demand and automatically scale the underlying virtualized resources to serve elastic workloads with varying concurrency (Scheuner & Leitner, 2020). Functions can be integrated with other services, such as databases, authentication and authorization services, and messaging services. These services are referred to as Backend-as-a-Service (BaaS).

The major advantages of this model are higher scalability and independence of applications as well as lower costs. Since costs are based only on the functionality used, there are no expenses for inactive resources. A major disadvantage is lower control in FaaS, since it is managed by an external service provider.

AWS Lambda

There are many cloud services and infrastructure providers that offer serverless computing services, such as Amazon, IBM, Microsoft, and Google, and also several open-source alternatives such as OpenLambda and OpenWhisk. In this work, we use Amazon Lambda[1] as test platform.

AWS Lambda is a service that allows code to be executed as a function in a serverless environment. AWS takes care of the underlying infrastructure that the function requires to be able to run the code. The customer configures the function, uploads the code to be executed, and determines the triggers that invoke the function. The function executes when it is called, and after execution is complete, it shuts down. AWS Lambda natively supports Java, Go, PowerShell, Node.js, C#, Python, and Ruby code, and provides a Runtime API that allows you to use any additional programming languages.

Lambda functions can be integrated with various services to be triggered by events created by the services. The triggering event can be an external request, a scheduled internal event, a direct request from an AWS service, or a life cycle event on some of the AWS services. For example, in this paper, we use Amazon Simple Storage Service (Amazon S3) to trigger AWS Lambda data processing.

Lambda functions scale up and down in a very flexible manner. When a request arrives and all the currently existing environment instances are still processing the previous requests, Lambda creates a new instance for the new request. After processing a request, the execution environment waits for incoming requests for a while and if none arrives, it shuts down.

Material and methods

As mentioned earlier, the integration of IoT and serverless devices is promising, but it can also bring challenges. In this context, the objective of this study is to analyze how these technologies can be integrated into a face detection application. With this in mind, we have developed a use case to test this integration and analyze the results in terms of performance, energy and cost.

[1] https://aws.amazon.com/lambda/

Face detection application

The face detection application was developed in Python programming language and uses the OpenCV library for the processing of images with the Haar-Cascade method[2]. This implementation already has a training base for face recognition, so this step is unnecessary and was not executed. The application receives an image and uses the already trained database to detect how many faces and how many eyes are present in that image. Processing can be done on the IoT device using its own hardware or using the serverless service AWS Lambda. The source code of both versions is available in GitHub[3].

Serverless technology uses containers for function execution, but controlling installed packages, as in a local environment, it is not possible. This is challenging when there is a need of the OpenCV library for the application in this study. To solve this, Lambda functions employ the Layers concept, enabling the addition of extra code via ZIP files. This lets us bundle the OpenCV library to match the AWS container structure, effectively solving the issue.

Concerning the application execution, when running on local hardware, the application loads a user-specified number of images (Figure 1 (a), step 1), processes them (Figure 1 (a), step 2), stores the results in output files, and then sends the results to a database in AWS S3 service (Figure 1 (a), step 3). The device used to run the application and simulate an IoT device was a Raspberry PI 4 Model B with a quad-core Cortex-A72 (ARM v8) 64-bit SoC @ 1.5 GHz, 2 GB LPDDR4-3200 SDRAM, and Raspberry Pi OS 64bit operating system. Although the application is sequential, the OpenCV library uses all 4 threads of Raspberry PI, so all of the computing power of the device can be used.

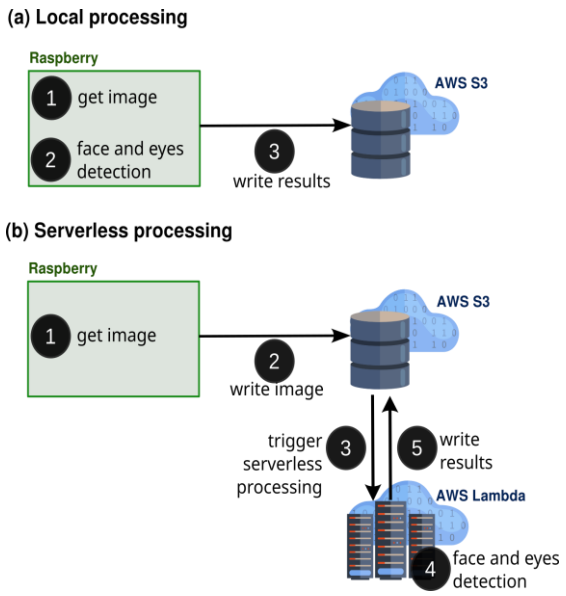

Figure *1.* Local (a) and serverless (b) execution steps.

The serverless implementation shows a different behavior, as illustrated in Figure 1(b). Here, the Raspberry is used to load (Figure 1(b), step 1) and send the images to the S3 database over the network (Figure 1(b), step 2). Thus, instead of loading images locally for processing, the Lambda function uses images stored in an AWS S3 database. When an image is sent to a specific folder in the database, the function is triggered in AWS Lambda service (Figure 1(b), step 3). After processing (Figure 1(b), step 4), the function stores the results in AWS S3 (Figure 1(b), step 5).

Experimental evaluation

For testing with the face detection application, we used 1024×1024 resolution images from the faces database available in the Flickr-Faces-HQ Dataset (FFHQ)[4]. The application was run with 1, 100, 500 and 999 images, both locally on the device and on the serverless server. The application was run 10 times in each experiment. The objective of the experiments is to assess situations in which the utilization of serverless

[2] https://docs.opencv.org/4.x/db/d28/tutorial_cascade_classifier.html
[3] https://github.com/pistydotta/face_recognition_python
[4] https://github.com/Nvlabs/ffhq-dataset

technology proves advantageous within the context of IoT. We evaluate three aspects: performance, energy consumption and financial cost.

The performance of the different solutions is evaluated based on the image processing time and the total execution, which comprises the image processing itself, and the time spent with communication and writing of the results. In device, images are processed locally and one result file is generated to each image. Thus, the processing time is calculated by obtaining a timestamp before and after image processing, using the Python *time()* method[5], and calculating the difference between timestamps. Next, results can be uploaded to S3 database in two manners: one file at each time (after the processing) or in batch. We evaluated the results transference time using wireless and wired networks. The computation of transfer time is analogous to that of processing time.

In serverless execution, the processing time is not directly available because it depends on the trigger time function. Hence, an individual output file is generated for each image, containing timestamps indicating the beginning and the end of the processing phase. This resultant file is then analyzed to extract the earliest and latest timestamps, enabling the computation of the processing duration. The transfer of results was not considered, as the communication occurs internally in AWS servers and is not measurable. We evaluated the cloud execution using wireless and wired networks.

To measure the energy consumed by the Raspberry PI 4, we used an INA219 module connected to an Arduino Nano, as illustrated in Figure 2. The INA219 module is a current and voltage sensor used to monitor current, voltage and power. The Arduino Nano was connected to a computer via a USB cable and the data was collected by the Arduino IDE software using the Serial Monitor. Note that the Raspberry Pi 4 used in this study was in its default configuration, without any power-saving optimizations.

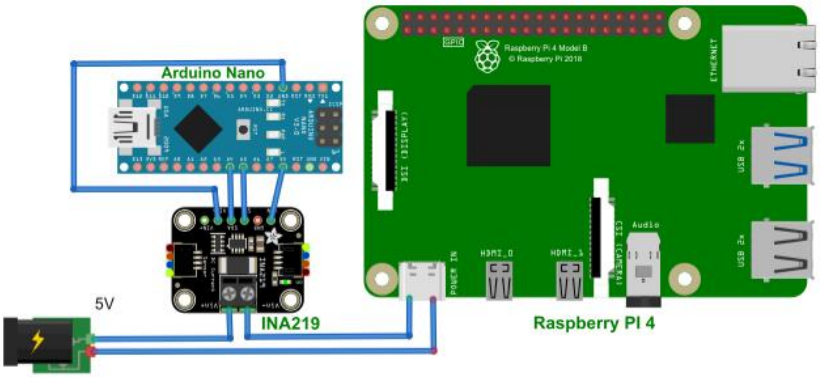

Figure 2. Energy consumption measurement module.

The function executed in Arduino collects the current consumed by the device in *m*A using the continuous mode of INA219 module library. Assuming a constant voltage of 5V to power the Raspberry PI 4 and dividing the current by 1,000 to get the measurement in Amperes, we multiply the current by the voltage to get the average consumption of the device in Watts. The Raspberry PI 4 used in this study was in its default configuration with no power consumption optimizations. Energy consumption is acquired through data output in the Arduino console, taking into account the measurements captured during image processing and data transfer periods, indicated by the timestamps produced by the application.

At last, we evaluate the financial cost of using serverless services. The financial cost in AWS Lambda is calculated based on the amount of computing resources and execution time consumed by the functions. To accurately calculate the costs for using AWS Lambda, it is recommended to use the AWS Pricing Calculator[6] or refer to the official documentation for detailed information about current pricing.

Results and discussion

In this section, we present the results of experimental evaluation. The results are presented in terms of execution and communication times, energy consumption and financial costs.

[5] https://docs.python.org/3/library/time.html
[6] https://calculator.aws/#/addService/Lambda

Execution and communication times

This section presents the results of executing the application directly on device and using serverless. Figures 3, 4, 5 and 6 present the total execution time and processing time for experiments with 1, 100, 500, and 999 images. The processing time for one image is shown in Figure 3. We can observe that local processing for the execution of a unique image performed better on the local device, since it is not possible to use the scalability of the serverless platform. Likewise, the startup time of a platform container is relevant in this case.

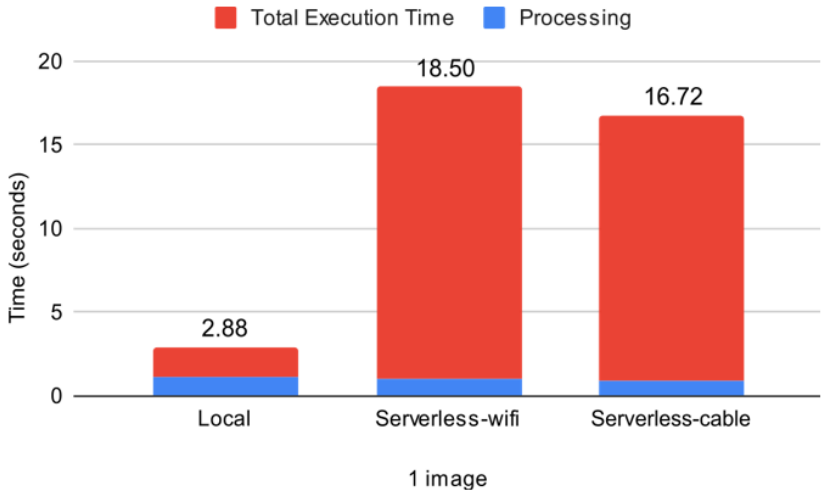

Figure 3. Execution time: 1 image.

The results for processing 100 images are shown in Figure 4. We can observe that the performance of local processing is comparable to the performance of serverless processing when using the wireless network. It can also be observed that sending results in single files significantly affects the overall execution time of local processing compared to sending results in batch. Serverless processing over a wired network present a smaller execution time. This is due to stability and transmission speed, as images are sent to AWS S3 faster.

Figure *4.* Execution time: 100 images.

In Figures 5 and 6 we present the results for 500 and 999 images, respectively. We can observe similar behavior in these 2 experiments, with serverless processing in the wired network giving the best results in both cases. It is more than 3 times faster than serverless processing on the wireless network and almost 6 times faster than local processing with batch communication.

In both experiments, cloud processing outperforms local processing due to the scalability of the serverless platform. In this study, we found that the biggest limitation of serverless processing is the speed at which the images are sent to the AWS S3. The faster the images are sent to the cloud; the more container instances are used and the more parallelism is achieved.

Figure 5. Execution time: 500 images.

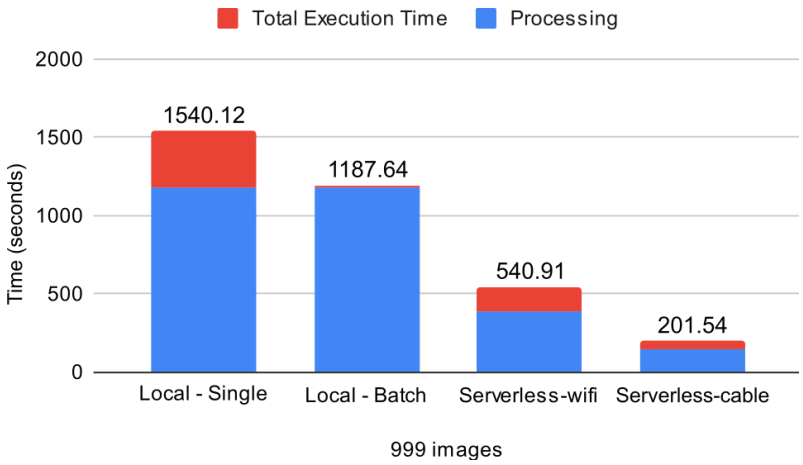

Figure 6. Execution time: 999 images.

In the serverless platform, the time to send the images has a significant impact on the execution time of the application because, as mentioned earlier, the major limitation of this approach in this application is the time it takes for the device to send the images to the AWS S3 database. Figure 7 shows the time spent to send images to AWS S3 over wireless and a wired network. Here, we can observe that the time for sending the images summed to the time of serverless processing is smaller than the total execution time (Figures 3-6). This is because not all images need to be sent to start processing. The AWS Lambda trigger works in such a way that every time an image is received in AWS S3, the function is triggered for processing it. Thus, processing is started with the first received image, which significantly reduces the total execution time, since there is an overlap of communication and processing.

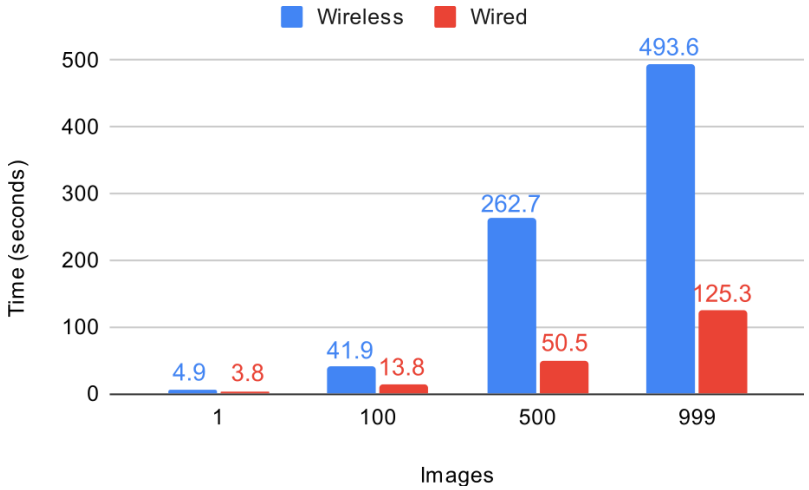

Figure 7. Time spent to send images to AWS S3.

We also evaluate the cost of transmitting the results in cases where the images are processed directly on the device. The results are presented in Figure 8. Batch transmission of results is similar regardless of the number of images processed (about 2.8 seconds), so the size of the individual file results does not significantly affect the transmission time. For results in separate files, the time increases according to the number of images processed, i. e., even if the files are small, the quantity has a significant effect on the transmission time. Therefore, when processing in local device, the transmission of data must be in batch.

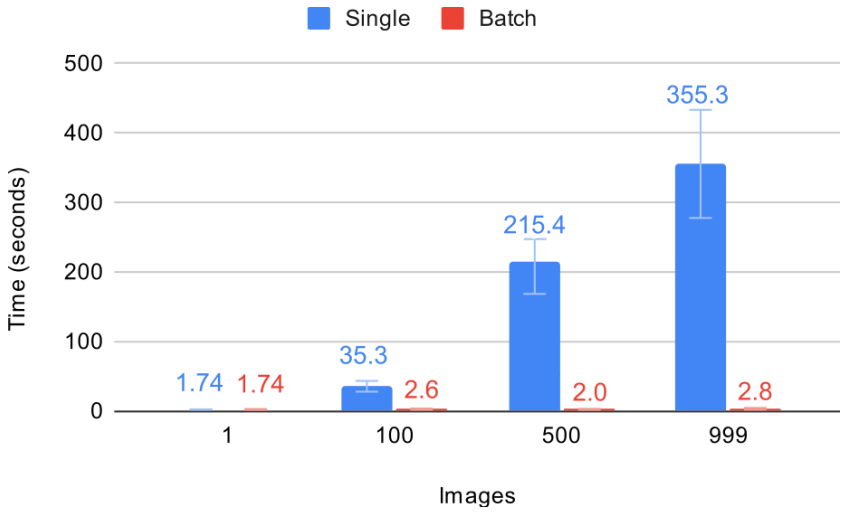

Figure 8. Time spent to send results to AWS S3.

From the experiments, it was also possible to determine the aggregate total execution time that the serverless platform achieved to perform the processing, as shown in Figure 9. The aggregate total execution time of the platform is a sum of the image processing times, which represents the time required to process the images using a single container, i. e., sequentially. This metric highlights the importance of scalability of FaaS platforms. While the processing aggregated time of 500 images is 546 seconds, the elapsed processing time for 500 images on the platform was less than 250 seconds for the wireless network, and less than 100 seconds for the wired network. In the latter case, it can be stated that at least 5 containers were active at the same time while executing the functions.

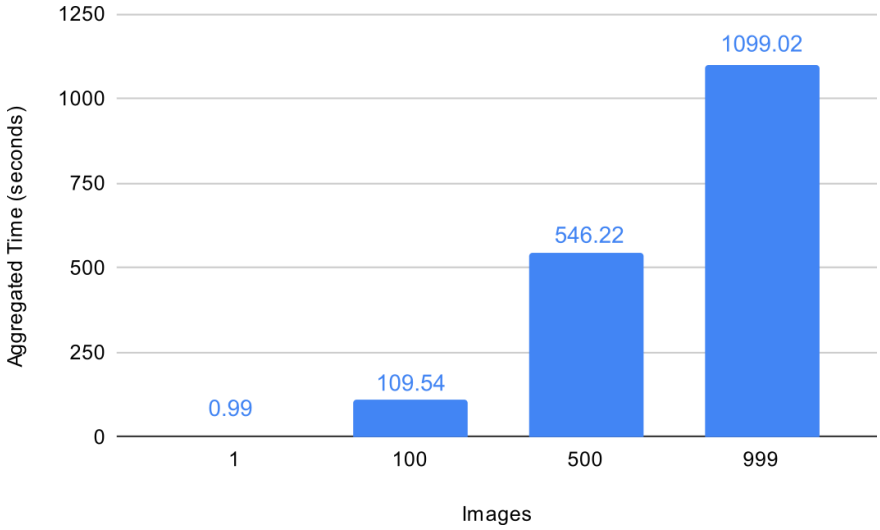

Figure 9. Serverless aggregated execution time.

Energy

The consumption data was collected during image processing and also when sending images to the S3 database over the wired and wireless networks. On average, the device costs 5,234 W while processing images costs 3,835 W in wireless network communication, and costs 3,565 W using a wired network. These average values were multiplied by time spent for processing 500 images, and by the time spent to send 500 images via

wireless and wired networks, resulting in the energy value that the device consumes, in joules, in each case. These values are shown in Figure 10.

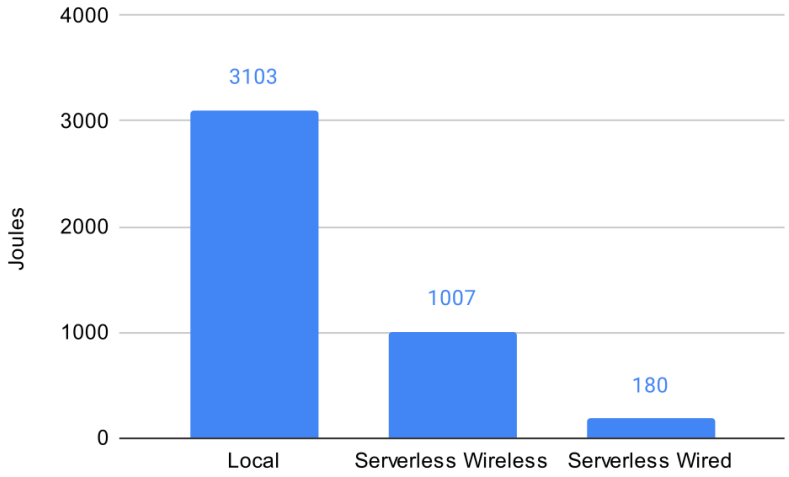

Figure 10. Energy cost: 500 images.

We can observe the high energy cost when the device performs local processing (3,103 joules), as this is the strategy that consumes the maximum device resources and also takes longer for its accomplishment. In serverless processing, the device only sends the images to be processed. Thus, while the device takes about 600 seconds to process locally, it takes only 50 seconds to send the images over the wired network. Therefore, the energy cost for sending the images is smaller than the processing cost (1,007 joules using wireless versus 180 joules in wired network).

Therefore, if the IoT device is connected to a battery and energy consumption is a major concern, serverless technology is a viable alternative, either over a wireless network or a wired network. However, when only a single image needs to be processed, transmission over wireless network costs almost 3 times as much as local processing, as illustrated in Figure 11. In this particular case, local processing is more advantageous in terms of energy consumption (about 6 joules) than using serverless (13 joules in wired network and about 19 in wireless).

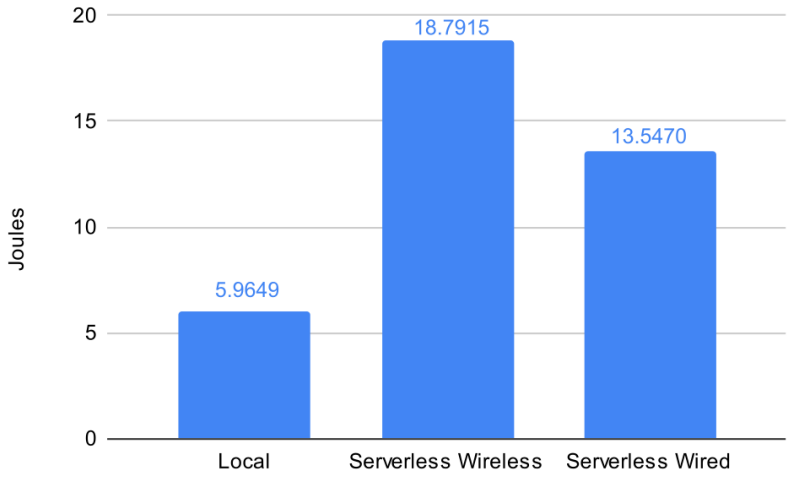

Figure 11. Energy cost: 1 image.

Financial cost

Despite the potential benefits of cloud processing, one counterargument could be budget constraints. It is possible to calculate the cost of running the application by considering the number of images and the time needed for each image. For example, to process 999 images, the total time on the serverless platform is 1,099 seconds. Since the AWS Lambda service was configured to use 3,008 MB of memory, the total cost is US$ 0.05.

When processing one single image, the average processing time on the platform was 0.993 seconds, resulting in a cost of US$ 0.00005 per image. Note, however, that the cost may depend on the platform used and the configuration adopted[7].

Discussion

The experiments conducted in this work have shown that the integration of these two technologies is possible and can overcome the limitations of IoT devices. Not surprisingly, the serverless technology was superior to the IoT device used in terms of performance. However, in this work, the performance of the technology was directly related to the quality of the network connection to which the device was connected. This can be a problem if the network connection available at the location of the IoT device is limited, or it can be an advantage if the network connection is stable and present low latency when handling file uploads.

Serverless technology can also be very useful when power supply to an IoT device is an issue, as these devices are usually battery-connected and have this limited power source. The experiments have shown how expensive it can be for the device to process an application locally, making the serverless paradigm an effective alternative for reducing a device's energy consumption.

This integration can also be situational, as serverless technology does not require local resources and can be activated at any time, thus, the device can choose to use the technology in specific cases where processing speed is a concern. Other situation is when the energy available on the device is low, meaning IoT devices can only use the serverless only as an alternative for when local processing is not the ideal choice.

Conclusion

The goal of this work was to analyze how to integrate IoT and serverless technologies and in which scenarios such integration is worthwhile. The results show that serverless technology outperforms the IoT device in terms of performance, considering its intrinsic scalability. However, this performance is closely tied to the quality of the network connection to which the device is connected. In addition, serverless technology proves beneficial in scenarios where power supply to an IoT device is an issue.

References

Agudelo-Sanabria, S. D., & Jindal, A. (2021). The Ifs and Buts of the Development Approaches for IoT Applications. *arXiv*. DOI: https://doi.org/10.48550/ARXIV.2101.09796

Benedetti, P., Femminella, M., Reali, G., & Steenhaut, K. (2021). Experimental analysis of the application of serverless computing to iot platforms. *Sensors*, *21*(3), 1-20. DOI: https://doi.org/10.3390/s21030928

Benomar, Z., Longo, F., Merlino, G., & Puliafito, A. (2021). Deviceless: A serverless approach for the internet of things. In *Proceedings of 2021 ITU kaleidoscope: Connecting physical and virtual worlds* (p. 1-8). DOI: https://doi.org/10.23919/ITUK53220.2021.9662096

Cassel, G. A. S., Rodrigues, V. F., Rosa Righi, R., Bez, M. R., Nepomuceno, A. C., & Costa, C. A. (2022). Serverless computing for internet of things: A systematic literature review. *Future Generation Computer Systems*, *128*, 299-316. DOI: https://doi.org/10.1016/j.future.2021.10.020

Eismann, S., Scheuner, J., van Eyk, E., Schwinger, M., Grohmann, J., Herbst, N., ... Iosup, A. (2021). Serverless applications: Why, when, and how? *IEEE Software*, *38*, 32-39. DOI: https://doi.org/10.1109/MS.2020.3023302

Esposito, M., Belli, A., Palma, L., & Pierleoni, P. (2023). Design and Implementation of a Framework for Smart Home Automation Based on Cellular IoT, MQTT, and Serverless Functions. *Sensors*, *23*(9), 1-16. DOI: http://dx.doi.org/10.3390/s23094459

Firouzi, F., Farahani, B., Weinberger, M., DePace, G., Aliee, F. S. (2020). IoT Fundamentals: Definitions, Architectures, Challenges, and Promises. In F. Firouzi, K. Chakrabarty, S. Nassif (Eds.), *Intelligent Internet of Things* (p. 3-50). Cham, SW: Springer. DOI: https://doi.org/10.1007/978-3-030-30367-9_1

Großmann, M., Ioannidis, C., & Le, D. T. (2019). Applicability of serverless computing in fog computing environments for iot scenarios. In *Proceedings of the 12th IEEE/ACM international conference on utility and cloud computing companion* (p. 29-34). DOI: https://doi.org/10.1145/3368235.3368834

[7] https://aws.amazon.com/lambda/pricing/

Jonas, E., Schleier-Smith, J., Sreekanti, V., Tsai, C., Khandelwal, A., Pu, Q., ... Patterson, D. A. (2019). Cloud Programming Simplified: A Berkeley View on Serverless Computing. *arXiv*. DOI: https://doi.org/10.48550/ARXIV.1902.03383

Kjorveziroski, V., Filiposka, S., & Trajkovik, V. (2021). IoT serverless computing at the edge: A systematic mapping review. *Computers*, *10*(10), 1-21. DOI: https://doi.org/10.3390/computers10100130

McGrath, G., & Brenner, P. R. (2017). Serverless computing: Design, implementation, and performance. In *Proceedings of 2017 IEEE 37th international conference on distributed computing systems workshops (ICDCSW)* (p. 405-410). DOI: https://doi.org/10.1109/ICDCSW.2017.36

Puliafito, C., Mingozzi, E., Longo, F., Puliafito, A., & Rana, O. (2019). Fog computing for the internet of things: A survey. *ACM Transactions on Internet Technology*, *19*(2), 1-41. DOI: https://doi.org/10.1145/3301443

Samie, F., Bauer, L., & Henkel, J. (2016). IoT technologies for embedded computing: A survey. In *Proceedings of the eleventh IEEE/ACM/IFIP international conference on hardware/software codesign and system synthesis* (p. 1-10). DOI: https://doi.org/10.1145/2968456.2974004

Scheuner, J., & Leitner, P. (2020). Function-as-a-service performance evaluation: A multivocal literature review. *Journal of Systems and Software*, *170*, 1-23. DOI: https://doi.org/10.1016/j.jss.2020.110708

Whitmore, A., Agarwal, A., & Xu, L. (2015). The internet of things–a survey of topics and trends. *Information Systems Frontiers*, *17*(2), 261-274. DOI: https://doi.org/10.1007/s10796-014-9489-2