Leveraging Constrained Devices for Custom Code Execution in the Internet of Things*

Flávia Pisani[1], **Edson Borin**[1]

[1]Institute of Computing – University of Campinas, Campinas – SP – Brazil

{fpisani,edson}@ic.unicamp.br

Abstract. *With the ever-growing scale of the IoT, transmitting a massive volume of sensor data through the network will be too taxing. However, it will be challenging to include resource-constrained IoT devices as processing nodes in the fog computing hierarchy. To allow the execution of custom code sent by users on these devices, which are too limited for many current tools, we developed a platform called LibMiletusCOISA (LMC). Moreover, we created two models where the user can choose a cost metric (e.g., energy consumption) and then use it to decide whether to execute their code on the cloud or on the device that collected the data. We employed these models to characterize different scenarios and simulate future situations where changes in the technology can impact this decision.*

1. Introduction

Current prospects for the Internet of Things (IoT) indicate that, within the next few years, a global network of objects will connect tens of billions of devices through the Internet [Lucero 2016]. As this technology becomes more widespread, we can also expect that a large number of devices with limited resources (e.g., power, memory, and processing) will become part of it. These devices are known as constrained devices.

Most IoT devices will contain sensors to help them interact with the environment around them, and the data they collect will create many opportunities. For instance, this will allow the improvement of strategies for resource usage and management in settings such as urban planning and environmental sustainability, thus advancing agriculture and contributing to an increase in the number of smart cities. It will also promote automation and the use of cyber-physical systems, prompting the widespread adoption of industry 4.0 [Bittencourt et al. 2018]. Moreover, considering that important processes such as data analytics can benefit from working with more data, this will lead to data scientists not only being able to better understand the world we live in, but also making more accurate predictions and creating improved systems based on people's behaviors and tastes.

With petabytes of data being produced by IoT devices every day, the sheer volume of information will make transmitting every single byte to the cloud prohibitively expensive in terms of both time and money [Pisani et al. 2019]. Furthermore, moving data streams from sensor nodes to servers will also have an impact on the energy consumption of devices that may have constricted power budgets (e.g., devices that are battery-operated or depend on limited energy sources such as solar power).

A possible way to meet these expected requirements is not sending all the data to be processed far from the data source, but instead bringing the computation closer to

*This study was financed by FUNCAMP (Samsung grant), CAPES, and CNPq (grant 140653/2017-1).

where the information already is. With this premise, a new paradigm called fog computing emerged, proposing to process data closer to network edge devices, such as switches and routers [Bonomi et al. 2012]. Enabling data to be processed near its origin (e.g., on a local access point, router, or even on the sensor device itself) allows us to address problems related to transmission latency and network congestion. It also opens up the space for new possibilities, such as filtering and discarding unnecessary information, analyzing readings in search of outliers to report, and actual real-time response to local queries.

The USA's National Institute of Standards and Technologies (NIST) calls attention to constrained devices within the fog computing context, naming them mist nodes [Iorga et al. 2018], as they are lightweight fog computing nodes. However, NIST expects the purpose of these devices to be more specialized and dedicated, while we see a valuable opportunity in leveraging them for custom user code execution due to their proximity to the data source. For instance, mist nodes have the potential to perform simple custom operations on sensor streams, such as aggregation and filtering, to reduce network traffic and latency. Given that the mist is part of the fog, when we refer to fog computing in this text, we are considering both fog and mist nodes.

Despite the potential of employing constrained IoT devices as a part of the fog hierarchy, many current fog computing frameworks [Cisco Systems 2019, Apache Software Foundation Incubator 2019, FogHorn Systems 2019, Saint 2015] still require more resources than what these devices provide. Therefore, investigating solutions that involve resource-constrained devices and creating a lean infrastructure that enables their seamless incorporation into the IoT is a gap that both industry and academia must explore to enable the full potential of this technology.

We especially highlight the importance of understanding and characterizing the scenarios where these devices can be used efficiently, as this is the key to developing solutions that use the most profitable approach for each specific problem. For example, there may be cases where the computation takes longer or requires more energy to be completed on the fog device than it would take to send the values to the cloud. In addition, there are situations where the whole computation depends on many different data sources. In these instances, it might be more profitable to send the data to be processed by the more robust cloud servers instead of executing the program locally. Considering this context, we work toward the answer to the research question "*In what cases is it more profitable to perform computation on a constrained IoT device instead of using the cloud?*".

To this end, we adopt the definition of constrained device as one where some of the characteristics that are expected to be present on other devices currently connected to the Internet are not achievable, often due to cost constraints and/or physical restrictions [Bormann et al. 2014]. In particular, we target Class 2 devices, as they can support most of the same protocol stacks as servers and laptop computers. An example of a smart constrained device in this class is the Fitbit activity tracker, which has 256 KiB of RAM, 64 KiB of Flash, and a microprocessor with a frequency of up to 80 MHz.

In the first part of our analysis, we developed a platform called LibMiletusCOISA (LMC), which allows users to execute their code on constrained devices, and compared it to the Apache Edgent framework [Apache Software Foundation Incubator 2019]. LMC performed well when executed on the same device as Edgent and made it possible to

execute the code on a constrained device which does not have enough memory to support the execution of existing fog computing tools. With that, we established that indeed there are cases where it is faster to perform the computation on a constrained IoT device (that is, more profitable in terms of time), therefore giving us more evidence to support the investigation of our research question.

In the second part of our analysis, we created two models where the user chooses a certain cost metric (e.g., execution time or energy consumption) and employs it to decide where they should execute their code. One of them is a generic mathematical model that uses a linear equation to determine the costs and the other is a visual model that allows the user to conclude quickly what is the most profitable approach in a specific scenario.

We used datasheets and the infrastructure built in the first part of our analysis to obtain real-world values to use in our test cases, and then used these values as the input for simulations that employed our models to identify the situations where executing the code on the device that collected the data would be the chosen approach. We also simulated future scenarios where changes in communication and processing technologies can affect whether the fog or the cloud is the most profitable solution. With that, we formalized an approach to identifying the cases where it is more profitable to perform computation on a constrained IoT device instead of using the cloud, which is what we intended with the investigation guided by our research question.

2. Related Work

Despite the novelty of the fog computing paradigm, several tools that employ it for general-purpose computation, or even specific applications such as data analytics, have appeared since its inception. For instance, we have Apache Edgent [Apache Software Foundation Incubator 2019], Cisco's IOx [Cisco Systems 2019], FogHorn [FogHorn Systems 2019], and Parstream [Saint 2015].

In order to compare our proposed framework to these tools, we chose four main characteristics that are relevant to our investigation of code execution on heterogeneous constrained devices. Table 1 has the results of this comparison. As Edgent is an open-source framework that can be executed on end devices, we chose it as a baseline for the experiments mentioned in Section 3.

Table 1. Comparison between LMC and Edgent, IOx, FogHorn, and Parstream.

	LMC	Edgent	IOx	FogHorn	Parstream
Runs on end devices?	✓	✓	✗	✗	✗
Runs on Class 2 constrained devices?	✓	✗	✗	✗	✗
Supports many programming languages?	✓	✗	✓	✓	✗
Open source?	✓	✓	✗	✗	✗

In Section 4, we discuss our two models that assist the analysis of the cost trade-off between fog and cloud computing. One of them is a general mathematical model and the other is a visual model inspired by the roofline model introduced by Williams, Waterman, and Patterson [Williams et al. 2009].

We note that the main goal of our mathematical model is different from many other computation offloading approaches [Jayaraman et al. 2014, Deng et al. 2016,

Xu and Ren 2016, Liu et al. 2018, Neto et al. 2018], given that we aim to enable users to select the most profitable platform according to a metric of their choosing, while the other studies focus on optimizing specific metrics (mostly time and energy).

Furthermore, the proposed mathematical model is intended to be simple so its implementation can be executed on constrained devices, which is not the case for several models. Although some approaches such as NWSLite [Gurun et al. 2004] also discuss the cost model needing to be mindful of resource usage, it is still more complex than our solution and too large to be used in Class 2 constrained devices. We consider this to be an important distinction between our approach and the others, as constrained devices will have a growing importance in the IoT scenario due to their low power consumption and reduced cost, size, and weight.

3. A Framework for Execution on Constrained Devices

As the studied tools were not compatible with Class 2 devices, we developed LibMiletusCOISA (LMC)[1], a framework that enables cross-platform code execution on constrained IoT devices by combining a compact virtual machine (Constrained OpenISA (COISA) [Auler et al. 2017]) with a lean event handling mechanism (LibMiletus [MotorolaMobilityLLC 2019]). Figure 1 shows an overview of the main components of the LMC framework and how they are connected.

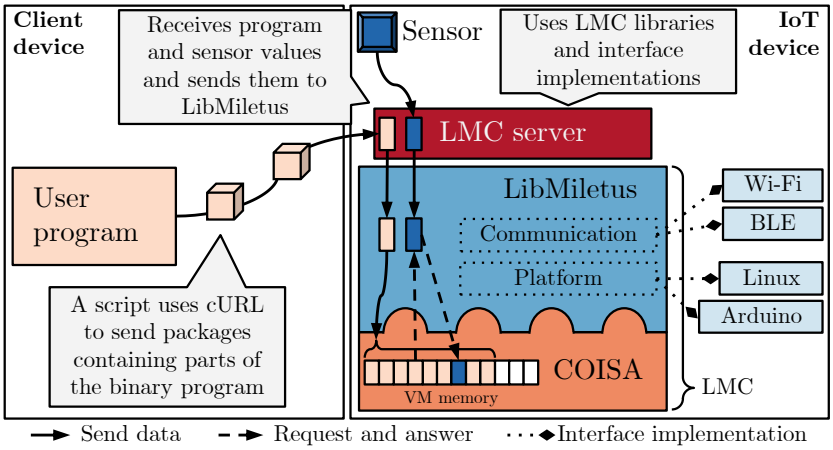

Figure 1. Overview of the main components of the LMC framework.

On the left side, we have the client device, which hosts a program written by the user with the help of libraries from the LMC framework. The program is compiled to create a binary file and sent to the IoT device through the network. On the right side, we have the IoT device where the LMC server is running. The server program's main function is divided into two parts: setup and event handling.

The first step of the setup consists in creating a new `MiletusDevice` object and defining the properties that identify the device and represent its characteristics (e.g., what type of sensor the device contains) and the methods that allow it to answer requests and execute user programs. Once the object is created, the setup defines its Platform interface, which is responsible for handling the output (e.g., a GNU/Linux Platform interface

[1]LMC is an open-source project available at https://github.com/lmcad-unicamp/LibMiletusCOISA.

outputs with `printf`, while an Arduino Platform interface uses `Serial.print`), as well as its Communication interface, which handles the connection using a certain wireless network technology (e.g., Wi-Fi or Bluetooth Low Energy (BLE)).

The event handling part is placed inside of an endless loop. It starts by calling the `handleEvents` method from the `MiletusDevice` object, which checks if there are any pending requests. In case there is custom code installed on the device, it is executed until an `exit` syscall is performed. If an event such as a sensor value update happens, the method sends it to the COISA event queue so it can be properly treated by the Virtual Machine (VM). When the `handleEvents` method returns, the current sensor values are requested and the `MiletusDevice` object is updated with this information (as we used simulated values in our experiments, the implementation of this function simply reads the next value instead of polling the sensors).

In order for COISA to be compatible with other intercommunication frameworks, we kept it completely independent from LibMiletus. Still, although we did not change COISA functionalities, we had to extend the implementation of its instruction set to support single and double precision floating-point operations, which are part of OpenISA, but still needed to be added to the VM. We also included counters to COISA that allowed us to get the statistics required for our experiments.

We modified LibMiletus to allow access to certain COISA functions and structures, such as COISA's VM memory. This way, when the user program invokes a syscall provided by LMC for reading values, the server event handling process can respond to this request with the current value of a certain sensor by storing it in a place allocated by the user program in the VM memory. We also added a new `_coisa` trait and a command that installs the user code on the device that is running the server. This installation process occurs by copying the code to the VM's memory. If a new event happens and there is a code to be executed, we made LibMiletus use COISA for this job.

We used LMC to deploy and execute one-time and continuous queries on constrained IoT devices and compared its performance with the Apache Edgent framework using three test cases on two IoT platforms, the simple NodeMCU 1.0 and the more robust DragonBoard 410c. Our experimental results indicated that LMC is: *i)* very compact and compatible with Class 2 constrained devices; *ii)* overall faster than the Edgent framework if we disable dynamic translation mechanisms; *iii)* faster than Edgent at lightweight quick queries when they are both being interpreted, in some cases even if LMC is running on the small NodeMCU platform while Edgent is running on the DragonBoard 410c.

4. Modeling Cloud and Fog Execution Costs

In order to be able to systematically identify the cases where it is more profitable to leverage constrained IoT devices for custom code execution instead of employing more powerful devices, it was necessary to characterize the scenarios where this holds true. Thus, we introduced two cost models for fog and cloud computing.

The first is a mathematical model that can be used to estimate the cost of processing a data stream of size z on the device that collected it (fog computing cost, or $C_{_F}$) and the cost of sending the data to be processed on the cloud (cloud computing cost, or $C_{_C}$). These costs consider steps such as reading a sensor value (r), performing a custom code

operation (t), sending the data to the cloud (s), and being idle until there is a new read (i). The cost of each of these steps is calculated through profiling and from the point of view of the device, allowing it to choose where to process the data stream while collecting its elements. As filters have a very important role in the future of the IoT and can be implemented as lightweight programs capable of running on constrained devices, this kind of procedure is the main focus of our study. Therefore, another parameter of our model is the probability that a number will pass the filter (f). The left side of Figure 2 illustrates how each of these parameters are related to C_F and C_C, as well as the corresponding equations.

The second is a visual model that is a C_C vs. C_F graph (right side of Figure 2). In this graph, the horizontal axis represents C_F as a function of the number of stream values being processed on the fog. Therefore, a case where all data points are processed on the fog is represented by a point crossing this axis, called C_{F_0}. The same holds for the vertical axis, C_C, and C_{C_0}. Using this visual model, the user can better understand the test cases they are working with and improve the implementation of their processing strategy. The user can decide where to execute their code by analyzing the slope of a line that connects C_{F_0} to C_{C_0}, with fog computing being more profitable in the cases where the slope is less than or equal to -1, and cloud computing having the lower cost otherwise.

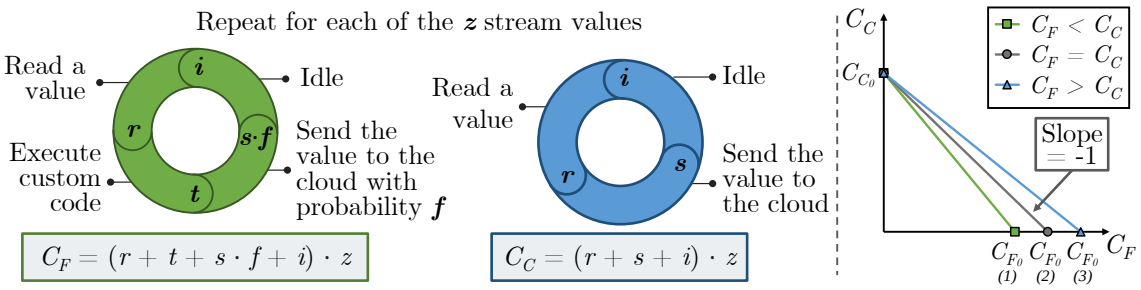

Figure 2. Overview of the mathematical (left) and visual (right) models.

We analyzed different strategies to estimate the value of f and observed that looking at a contiguous set of elements at the beginning of the stream is a straightforward approach that yields good estimates. The two other approaches that we tested, which involve continuously monitoring the stream and dynamically adjusting the estimate, presented very little performance gains, not justifying their use.

We applied our models to two instances of two different filters, used execution time and energy consumption as the cost types for our analysis, and executed the tests on five different datasets (four datasets with real-world climatological data and one dataset with artificial data). By comparing the slope of the linear equation obtained with the real and estimated values of f, we noticed that our estimation process worked well, as it presented an error of less than 5% in most of our test cases and allowed us to decide correctly on the more profitable strategy to process the values in all but one case.

We also simulated a different range of values for our test cases and found out how different parameters would affect our decision. We looked at how much it would be necessary to decrease s or increase t for cloud computing to become the more profitable approach in cases where the fog was the currently chosen solution. When using execution time as the cost type, the values of the parameters had to change from $1.9\times$ to $9.8\times$ to affect our decision, and in the case of energy consumption as the cost type, they had to

change from $7.9\times$ up to $39.8\times$. We noticed that the size of these alterations depends on factors such as the value of f and how close s and t are to each other. We point out that this type of investigation is very useful to visualize possible changes in technology. Again, our estimation process proved to be effective, as the simulations using the real and predicted values presented the same decisions in all cases.

5. Main Contributions

The main contributions of this work are as follows:

- We developed an open-source platform called LibMiletusCOISA (LMC), which allows users to execute their code on constrained devices;
- We used our infrastructure to obtain real-world values to use in our test cases;
- We created two models where the user chooses a certain cost metric (e.g., number of instructions, execution time, and energy consumption) and employs it to decide where they should execute their code:
  - A generic mathematical model that uses a linear equation to determine the costs;
  - A visual model that allows the user to quickly conclude what is the most profitable approach in a specific scenario.
- We created a procedure to use our mathematical model to estimate the probability of a value passing a filter based on the cost penalty that the user is willing to pay for this calculation;
- We simulated future scenarios where changes in communication and processing technologies can affect if the fog or the cloud is the most profitable approach.

6. Publications

1. F. Pisani, F. M. C. de Oliveira, E. de S. Gama, R. Immich, L. F. Bittencourt, and E. Borin. "Fog Computing on Constrained Devices: Paving the Way for the Future IoT". In: *Advances in Edge Computing: Massive Parallel Processing and Applications*, pp. 22–60. doi:10.3233/APC200003.
2. F. Pisani, V. M. do Rosario, and E. Borin. "Fog vs. Cloud Computing: Should I Stay or Should I Go?" In: *Future Internet* 11.2 (2019). Article 34. doi:10.3390/fi11020034.
3. F. Pisani and E. Borin. "Fog vs. cloud computing: should I stay or should I go?" In: *Proc. INTESA@ESWEEK*. Oct. 2018, pp. 27–32. doi:10.1145/3285017.3285026.
4. F. Pisani, J. R. Brunetta, V. M. do Rosario, and E. Borin. "Beyond the Fog: Bringing Cross-Platform Code Execution to Constrained IoT Devices". In: *Proc. SBAC-PAD*. Oct. 2017, pp. 17–24. doi:10.1109/SBAC-PAD.2017.10.

References

Apache Software Foundation Incubator (2019). Apache Edgent Overview. https://edgent.apache.org/docs/overview. Accessed: May 01, 2019.

Auler, R., Millani, C. E., Brisighello, A., Linhares, A., and Borin, E. (2017). Handling IoT platform heterogeneity with COISA, a compact OpenISA virtual platform. *Concurr. Comp.-Pract. E.*, 29(22):e3932.

Bittencourt, L., Immich, R., Sakellariou, R., Fonseca, N., Madeira, E., Curado, M., Villas, L., da Silva, L., Lee, C., and Rana, O. (2018). The Internet of Things, Fog and Cloud continuum: Integration and challenges. *Internet of Things*, 3–4:134–155.

Bonomi, F., Milito, R., Zhu, J., and Addepalli, S. (2012). Fog Computing and Its Role in the Internet of Things. In *Proc. MCC '12*, pages 13–16.

Bormann, C., Ersue, M., and Keranen, A. (2014). Terminology for Constrained-Node Networks. Technical report, Internet Engineering Task Force.

Cisco Systems (2019). Introduction to IOx. https://developer.cisco.com/docs/iox/. Accessed: May 01, 2019.

Deng, R., Lu, R., Lai, C., Luan, T. H., and Liang, H. (2016). Optimal Workload Allocation in Fog-Cloud Computing Toward Balanced Delay and Power Consumption. *IEEE Internet Things*, 3(6):1171–1181.

FogHorn Systems (2019). Real-Time Edge Intelligence for Industrial IoT. www.foghorn.io/. Accessed: May 01, 2019.

Gurun, S., Krintz, C., and Wolski, R. (2004). NWSLite: A Light-Weight Prediction Utility for Mobile Devices. In *Proc. MobiSys '04*, pages 2–11.

Iorga, M., Feldman, L., Barton, R., Martin, M. J., Goren, N., and Mahmoudi, C. (2018). Fog Computing Conceptual Model - Recommendations of the National Institute of Standards and Technology. Technical report, National Institute of Standards and Technology.

Jayaraman, P. P., Gomes, J. B., Nguyen, H. L., Abdallah, Z. S., Krishnaswamy, S., and Zaslavsky, A. (2014). CARDAP: A Scalable Energy-Efficient Context Aware Distributed Mobile Data Analytics Platform for the Fog. In *Proc. ADBIS '14*, pages 192–206.

Liu, L., Chang, Z., Guo, X., Mao, S., and Ristaniemi, T. (2018). Multiobjective Optimization for Computation Offloading in Fog Computing. *IEEE Internet Things*, 5(1):283–294.

Lucero, S. (2016). IoT platforms: enabling the Internet of Things. Technical report, IHS Technology.

MotorolaMobilityLLC (2019). LibMiletus - IoT prototyping made easy! Accessed: May 02, 2019.

Neto, J. L. D., Yu, S.-Y., Macedo, D. F., Nogueira, J. M. S., Langar, R., and Secci, S. (2018). ULOOF: A User Level Online Offloading Framework for Mobile Edge Computing. *IEEE T. Mobile Comput.*, 17(11):2660–2674.

Pisani, F., Martins do Rosario, V., and Borin, E. (2019). Fog vs. Cloud Computing: Should I Stay or Should I Go? *Future Internet*, 11(2).

Saint, A. (2015). Where next for the Internet of Things? 10(1):72–75.

Williams, S., Waterman, A., and Patterson, D. (2009). Roofline: An Insightful Visual Performance Model for Multicore Architectures. *Commun. ACM*, 52(4):65–76.

Xu, J. and Ren, S. (2016). Online Learning for Offloading and Autoscaling in Renewable-Powered Mobile Edge Computing. In *Proc. GLOBECOM '16*.