Copyright Notice

Institute of Communication Networks and Computer Engineering
University of Stuttgart
Pfaffenwaldring 47, D-70569 Stuttgart, Germany
Phone: ++49-711-685-68026, Fax: ++49-711-685-67983
Email: mail@ikr.uni-stuttgart.de, http://www.ikr.uni-stuttgart.de

2021 Joint European Conference on Networks and Communications & 6G Summit (EuCNC/6G Summit) | 978-1-6654-1526-2/21/$31.00 ©2021 IEEE | DOI: 10.1109/EUCNC/6GSUMMIT51104.2021.9482556

Empirical Investigation of Offloading Decision Making in Industrial Edge Computing Scenarios

Alexander Artemenko*, Ismail Mehrez†, Keerthana Govindaraj*†, Andreas Kirstaedter† and Mykola Kuznietsov‡
*Robert Bosch GmbH, Corporate Sector Research and Advance Engineering, Renningen, Germany
{alexander.artemenko, keerthana.govindaraj}@de.bosch.com
†University of Stuttgart, Institute of Communication Networks and Computer Engineering, Stuttgart, Germany
andreas.kirstaedter@ikr.uni-stuttgart.de
‡Institute of Computer Systems, Odessa National Polytechnic University, Odessa, Ukraine, kuznietsov@opu.ua

***Abstract*—Edge Computing (EC) is a paradigm introduced to support the end devices with the execution of computation intensive tasks while maintaining their intended Quality of Service (QoS). To achieve this, one or more Edge Servers (ES) with powerful capabilities are placed in a close proximity to the edge devices to provide the assistance needed. In order to do that, EC utilizes the concept of application offloading, which is the idea of moving the computation intensive tasks to be computed on a more powerful server.**

The act of offloading is not always a beneficial choice due to the aspect of availability of involved resources and data communication between devices. Therefore, to achieve a successful offloading process, the offloading decision making needs to answer the four questions of when, what, where and how to offload. In this paper, we investigate the "When to offload?" question, which is concerned with whether the offloading process results in a positive gain in performance or not. To strengthen our conclusions, we use empirical observations in a real setup running a set of emulated applications.

***Index Terms*—Edge Computing, Application Offloading, Task Complexity, Offloading Gain**

I. Introduction

In the recent years, the Edge Computing (EC) paradigm has emerged as an extension to the Cloud Computing (CC) and is also sometimes referred to as fog computing [1], [2]. The main goal behind EC is to provide powerful services with high availability and reliability while maintaining a minimum delay for latency critical and control applications for the end devices. In order to achieve this desired goal, the concept of application offloading is utilized and is defined as moving the computation intensive tasks away from the local device to be processed on a more powerful server [3], [4]. Such Edge Servers (ES) are often supported with an enormous amount of processing capabilities such as Central Processing Units (CPU), Graphical Processing Units (GPU) as well as Random Access Memories (RAM) [5] and are placed in a close proximity to the end devices. Not only that, but these servers are supported with a constant power supply in most of the cases and thus energy consumption is not a concern. On the contrary, the end devices are often restricted by a limited amount of processing power and in a lot of cases are mobile and are connected to a mobile power sources [6], therefore introducing a major concern of energy consumption.

Although application offloading is expected to save computation time and energy by moving the complex tasks to other servers, it also introduces a new overhead on the network due to the transfer of data to be processed. This results in increase of time and energy costs [4]. As a result, in case if the computation effort saved is more than the one lost due to communication, then a gain state is achieved which is the goal of offloading. Otherwise, an offloading of this particular task will lead to a degraded performance. Moreover, the decision of offloading is always environment dependent. For instance, a certain low-end device might not be able to handle a specific task, therefore offloading is a reasonable choice, while on the other hand, a high end device can process it successfully and might even lose performance if offloading is done due to the extra unnecessary communication overhead. This introduces the challenge of choosing the appropriate task in a given application and deciding whether it would provide a gain or loss in the performance when offloaded to a certain cloud or edge server available in the network [2].

An optimal solution to the offloading decision problem is the one that achieves the best performance possible at a given cost, but unfortunately it does not have a straight forward solution. It rather needs the answer to four critical sub-questions: When, What, Where and How to offload [7]. The "When to offload?" question has to decide when should the application be offloaded. In other words, it states whether it is better to proceed with the offloading process or settle for the local execution option under the current circumstances. Moreover, many applications could be divided into more than two modules and therefore, a successful offloading mechanism has to decide "What to offload" [8]–[10]. Another aspect is "Where to offload". The answer to this question decides the destination on which the selected tasks should be offloaded to and that achieves the highest gain [11]. Finally, the "How to offload?" question is concerned by the means of offloading including the offloading channel and mechanism [12].

Multiple research works, which we present in Section II, made a considerable effort in solving the offloading decision problem and came up with a closed-form expression that would assist in making such decisions. As a result, the goal of this paper is to investigate the reliability of this expression and to validate that it works when tested in real-life conditions

using practical methods as expected from the theory. Another goal of this work is to do a sensitivity analysis to find out how the decision making is affected by a dynamic environment.

The remainder of this paper is organized as follows. In Section II, we introduce the related work and the state of the art related to the topic of application offloading. We then present our work concerning the offloading decision making in Section III describing the emulation and offloading processes and then evaluate our work in both static and dynamic environments in Section IV. Finally in Section V, we conclude our work and suggest some future steps.

II. Related Work

The offloading decision can be intuitive in cases with extremely complex tasks that require intensive processing and at the same time have a very limited amount of input data to be transmitted. In such scenarios, offloading would result in a gain in performance since a lot of computation effort is saved while a relatively small communication effort is added. On the other hand, offloading a task with a huge amount of transmission load will cause a degraded performance.

Multiple efforts were done in order to formulate a closed-form expression that facilitates making the decision of either to offload a specific task or not under the current state of the surrounding environment to achieve the desired offloading gain [7], [13]–[15]. The main condition that needs to be met for the offloading to be beneficial is that the cost of executing a task locally C_{mobile} should be more than the one when outsourcing it to a server for processing. The cost of performing the task on the server side is represented in terms of the cost of computation C_{server} and the cost to communicate the data between the devices C_{com}. This is represented as [7]:

$$C_{\text{mobile}} > C_{\text{server}} + C_{\text{com}} \tag{1}$$

The definition of the cost is made depending on the desired objective. When the objective is to minimize the overall performance time, Wu et al. [7] defined the condition for a rewarding offloading is;

$$\frac{C}{IPS_m} > \frac{1}{F}.\frac{C}{IPS_m} + \frac{D}{B}, \tag{2}$$

where C is the number of instructions of the task to be offloaded, IPS_m is instructions per second capability of the mobile client, D is the total amount of data transmitted between the nodes and B is the available bandwidth on the network link between them. For simplicity, a certain server s is assumed to be F times more powerful than a certain mobile client m.

Although the offloading expression seems straight forward, the results of the research mentioned were only based on pure simulations and do not take into account the dynamic nature of the surroundings. The theoretical approach assumes an exact knowledge of the resources state. Therefore, the work done lacks practical verification in order to state if the inequality holds when used in a real-life environment.

III. Offloading Concept

Our goal is to design an emulator that would practically reflect the behavior of a real-life scenario, where application offloading can be performed by a mobile device. Using the emulator, a wide range of applications with different settings for a computation complexity and a payload can be tested in a real practical environment. To indicate a certain task complexity, we used the Dhrystone benchmark [16]. This standard is used as a representation of the integer performance of a general purpose CPU in terms of VAX (Virtual Address eXtension) Dhrystone Million Instructions per Second (DMIPS), or in other words, by how much is this CPU more powerful than the standard 1 MIPS VAX machine.

To emulate the input and output data transferred between the client and server nodes, we prepare a message with a size that ranges between 1 B and 10 MB and send it over to the server side using TCP, where the computation would take place. By doing this, we are able to represent a wide range of application types starting from the ones using sensor data with few bytes up to the ones that transfer high resolution camera frames. Finally, we used the Linux tool iPerf3 [17] and WonderShaper [18] to emulate different network bandwidth availability. These tools have the ability to limit the data rate on a specific network interface. Thus, we can emulate different network conditions.

To observe if under a given combination of system parameters (Task Complexity, Message Size, Bandwidth, CPU) the offloading process would be beneficial, we first run both scripts on the mobile node in order to emulate the scenario when the task is not offloaded and we would get the local performance of the task which in our work was local completion time t_{local}. Then, we move the server script to the server side to emulate the offloaded case and we would get an offloaded performance time t_{offload}. The goal of this step is to check if the cost saved due to a more powerful computing platform is more than the one lost due to communicating the message through the network with a given availability. Finally, we define a gain in performance in the form of a ratio as:

$$Gain = \frac{t_{\text{local}}[s]}{t_{\text{offload}}[s]}, \tag{3}$$

where $t_{\text{local}} > 0$ and $t_{\text{offload}} > 0$. Hence, a gain value that is more than one indicates that the system performed better (faster) when an offloading took place, while a gain less than one means a degraded outcome.

IV. Evaluation Results

For our emulation and results discussed next, we used the devices specified in Table I. A Raspberry Pi model 3B device with a Cortex-A53 CPU acts as a mobile device that sends data D with sizes that range from 1 B to 10 MB to the server. On the other hand, we used a powerful HP Z-Book 15 machine with an Intel(R) Core(TM) i7-4810MQ CPU as the server node. Therefore, the server processing capability is $F \approx 11$ times more powerful than the client's one (23741.92 MIPS over 2162.82 MIPS). This server has the task of performing

an emulated computation complexity C that ranges from 0 to $600\text{x}10^6$ instructions. The measured available bandwidth between both devices was B=94 Mbps as the link between the access point and the client used is IEEE 802.11 (5 GHz) WiFi link and the link to the server is a 1 Gbps Ethernet connection.

TABLE I: A summary of the devices and infrastructure used for the emulation process.

Parameter	Value
Client	
Device	Raspberry Pi 3B
CPU Model	Cortex-A53
CPU MIPS	2162.82 MIPS
Operating System	Raspbian Buster Lite
Data Range	1 B - 10 MB
Transport Protocol	TCP
Infrastructure	
Resultant Bandwidth	94 Mbps
WiFi Link	802.11ac (5 GHz)
Ethernet Link	1 Gbps
Server	
Device	HP-ZBook 15
CPU Model	Intel(R) Core(TM) i7-4810MQ
CPU MIPS	23741.92 MIPS
Operating System	Ubuntu 18.04
Task Complexity	$0 - 600\text{x}10^6$ Instructions

A. Static Environment

In this subsection, we demonstrate our insights and observations when the offloading decision making is done in a static environment, where the measured parameters remain deterministic within the whole process and only minor alternations from the measured mean values take place as part of the environment, in which we observe.

Fig. 1 illustrates our initial findings when we emulated the whole range of complexities and message sizes in our previously described lab setup. The x-axis of the figure represents the range of task complexities used in terms of their number of instructions. The y-axis is the range of different message sizes that are to be transmitted between the devices and is shown in Bytes. Moreover, the color of the grid represents the gain we observed using Equation 3. An observed degraded performance while offloading with a gain value between 0 and 1 is marked in red, while an enhanced performance is marked in colors from yellow till green. The gradient of the colors represents the value of the gain which is demonstrated in the color scale on the right of the figure. For example, a dark red point means an extreme loss in performance while a light red means a slight loss, and the same applies for the green ones.

The blue diagonal linear line represents the theoretical boarder line that defines the offloading decision regions from Expression 2. Therefore, according to this polynomial, for any task with a given complexity in this environment, an offloading decision would lead to a gain (enhanced performance) if a message size to be transferred is below that line, while a loss is to be observed otherwise.

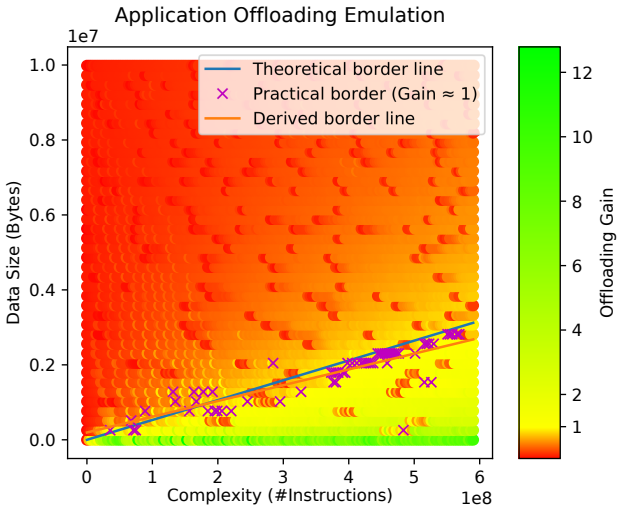

Fig. 1: The output of the offloading emulation process.

In our described emulation, we always offloaded the selected task and compared it's performance versus the case when it is not offloaded. Therefore, the cases which were successful are the ones in green and yellow while the failures are in orange and red. We can observe that the cases with extremely low message sizes resulted in a very high offloading gain reaching over a ten times enhanced performance, which can be explained as a result of the very low overhead introduced by the communication. It can also be seen that the gain value increases as the task gets more complex due to the increase in the computation cost saving. On the other hand, cases with very high message sizes resulted in high loss in communication effort and therefore offloading would not be beneficial. Furthermore, we also observed that offloading is never good for a very simple task with a very low complexity even if the message payload size is low because such tasks could be easily handled by the mobile device and any offloading would just mean an extra overhead.

The interesting region, where the offloading decision does not have an intuitive solution, is the area in the middle of the graph, where the values of the communication cost loss and the computation cost saved are comparable. This area is the one with dark yellow and light red points since the difference in the local and offloaded performance was not significant and therefore resulted in a gain value around one, which can be seen from Equation 3.

Fig. 1 shows that Equation 2 is able to separate the regions with performance loss and gain successfully. For a given task complexity, a payload size below the theoretic blue line practically leads to a gain in most of the cases while a value on the other side of the line leads to a loss in performance.

We could recognize that there are some outliers in the figure with a loss below the line and a gain above it, which would mean that there are cases where the expression is not reliable. By investigating this phenomenon, we suspected that our environmental variables at these cases were fluctuating

around the mean value due to the practical essence of our analysis. Such fluctuations could be a sudden drop or increase in the bandwidth between the devices due to a congestion or attenuation on the WiFi link. Another cause might be that the CPU serving this task on either nodes was not dedicating all its power to this task due to a sudden interrupt from the operating system or it was being shared by another process.

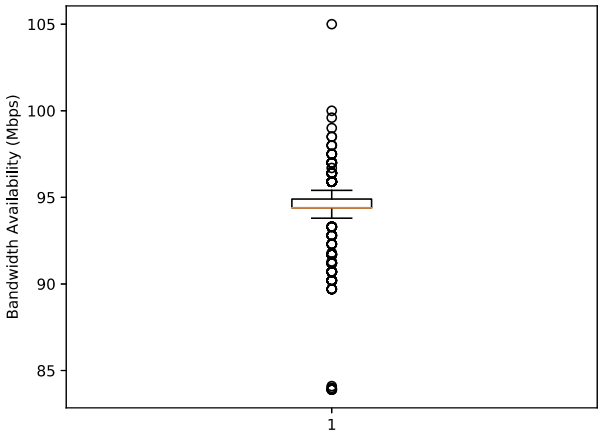

Fig. 2: Network status monitor.

To verify our suspicion and eliminate the effect of any sudden disruption in our environment, we monitored the available bandwidth as a test parameter during runtime and the result can be seen in Fig. 2. We observed a fluctuation in the availability around the mean value. Moreover, we ran each combination of payload and complexity five times and took the average in performance between all five trials, which can be seen in Fig. 3.

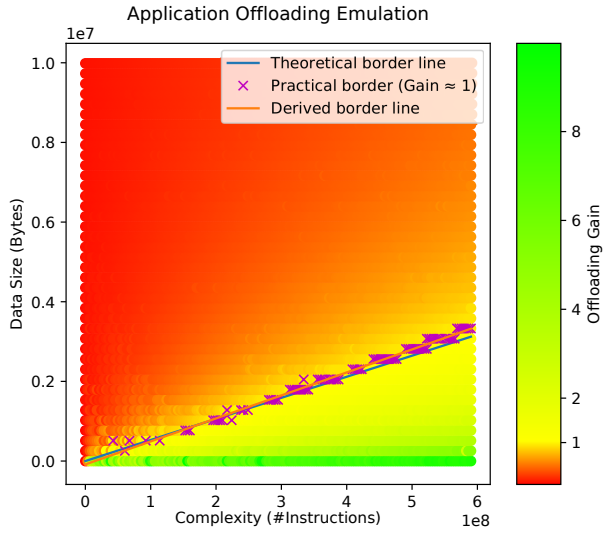

Fig. 3: Offloading emulation result using the average value of five iterations.

We observed from Fig. 3 that the offloading gain was changing in a constant manner and that the color grid became much smoother compared to the one in Fig. 1, where we used the result of only a single iteration. This observation made us realize that the outliers observed previously were actually a result of an unexpected change in the environment that happened at a particular moment.

It is now clear that the expression can be reliable whenever the cost of computing the task locally is significantly larger than the transfer cost. But we wanted to further investigate if the expression is reliable in the scenarios, in which the difference is small and it is not intuitive to make the offloading decision. To do that, we identified all the cases with offloading gain being close to one ($0.98 < \text{gain} < 1.02$) and marked them with the magenta crosses seen in Fig. 3. These cases represent the cases with a very low gain difference between local and offloaded performance being very close to the theoretical boarder line defined by Equation 2. We fitted a polynomial labeled derived borderline to those points, which we will use as our practical baseline. By comparing our derived boarder line to the theoretical one, we can observe a slight deviation between both lines. Since the theoretical expression is a linear polynomial, the gradient of the line is highly dependent on the accuracy of the parameters of our environment given, which are the CPU capabilities of our nodes and the network bandwidth available. As a result, we conclude that at the cases, in which the local and offloading costs are similar, the offloading decision making highly depends on the accuracy of the measured parameters.

B. Dynamic Environment

In this section, we present our results and observations when emulating different offloading examples as before but in more dynamic environments, where a measured parameter used in the decision making would not be the most recent value during the actual offloading. Such environments can be one with a mobile client device roaming around a factory floor causing attenuations or multiple connection handovers between access points due to the mobility aspect. Another case could be a multi-user scenario, in which the server keeps getting numerous requests from other devices and thus cannot dedicate all its processing power to the specified task. Any of the mentioned examples can result in a sudden drop of our parameters, required for the decision making. In this work, we focused on emulating an environment with sudden drops of the network bandwidth that becomes non-deterministic and analyzed its effect on the offloading decision making.

To analyze the described effect of the dynamic network availability on the decision making, we ran our whole emulation process multiple times and we changed the available network bandwidth in each of them. Fig. 4 illustrates our results for emulating different levels of sudden bandwidth drops that result in unpredictable changes of the data rate availability. While the decision making algorithm assumes the full bandwidth (94 Mbps) to be available, we on purpose overload the network and thus limit the bandwidth (BW)

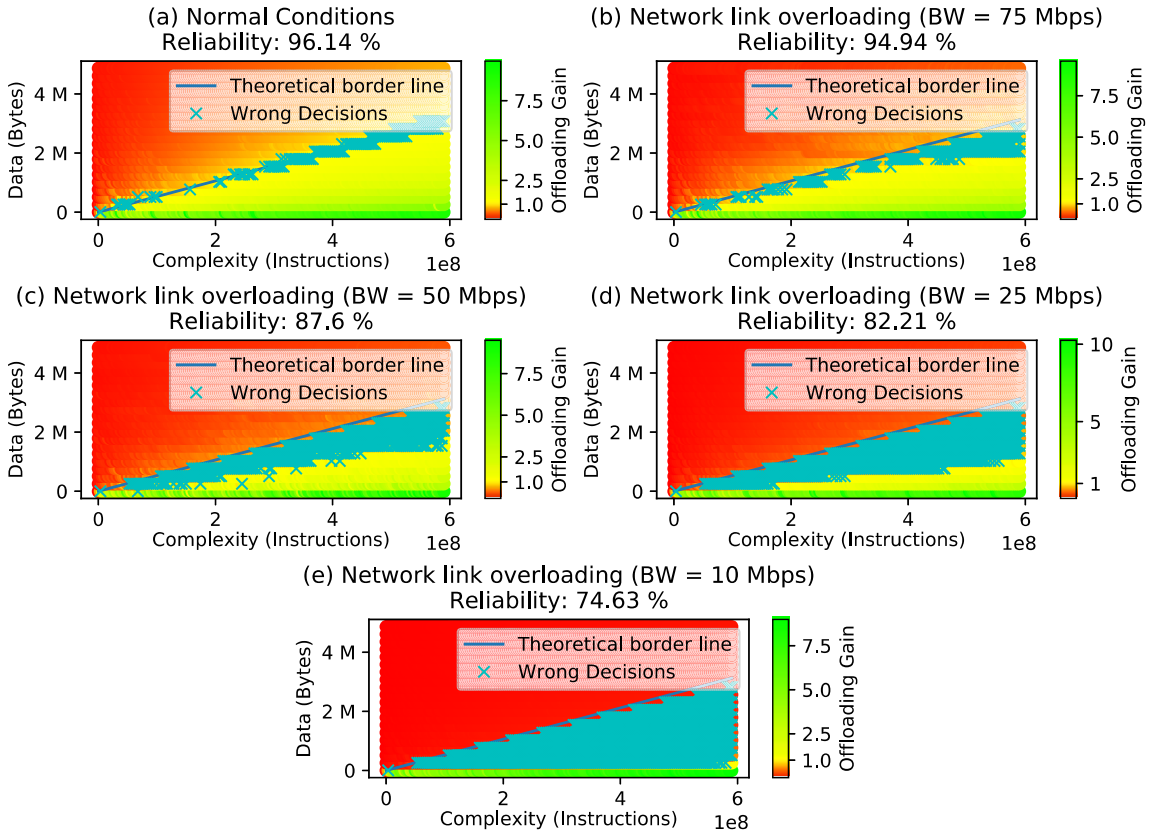

Fig. 4: Emulation results for different levels of a dynamic environment.

to 75, 50, 25 and 10 Mbps correspondingly. Moreover, the blue polynomial in each of the results is the decision maker result from our theoretic expression in Equation 2 using the full expected bandwidth value. By this methodology, we can have a better understanding about what would happen if we anticipated to have a certain bandwidth value (100% value), but in the real case, it dropped to the availability level being emulated.

We also defined a metric called Offloading Reliability, which is the percentage of correct decisions that the offloading decision maker would have made when used on all the range of application types we emulate. Although this value is relative to the range of complexities and payloads defined, we just use it as an assistance guide to help evaluate the correctness of our decision making not as a direct reference. To identify a correct decision, we contrasted the result from our theoretic decision maker in Equation 2 to the actual gain we practically observed. So if the expression anticipated a gain in performance and our emulated run showed a gain value more than one, then it is considered a correct decision. Similarly, an experienced gain value less than one and an anticipated loss in performance is also a correct decision because in that case offloading will not take place and therefore the system performance will not be degraded. On the other hand, if the results were different, this would be considered a false decision because when relying on the expression in such cases, we might offload and lose performance or even in some cases choose to perform the task locally when a positive gain from offloading could be achieved. We illustrated these wrong decisions by the cyan crosses on the figures.

Subplot (a) in Fig. 4 represents a scenario without bandwidth drops and hence a maximum reliability with a very little number of wrong decisions observed. Similarly, as the observation from Section IV-A, we got a number of wrong decisions around the boarder line defined by the theoretic expression that represent 3.9% of all our cases and therefore the expression is reliable in 96.1% of our test cases.

In subplots (b), (c), (d) and (e), we show the results of our emulation when we limited the available bandwidth to 75, 50, 25 and 10 Mbps respectively. We can clearly observe that the region of wrong decisions increases with the decrease of the network availability. Moreover, the wrong decisions are clearly concentrated below the theoretical boarder line since the bandwidth available is always less than expected and the offloading cost got higher thus affecting the cases with a decision to offload.

The main contribution of this subsection is that we were able to illustrate the range of applications (marked in cyan in Fig. 4) that would experience a false offloading decision when a certain level of network dynamics is present. Therefore, application and system architects can use diagrams like Fig. 4 as a heuristic guide to decide when to offload using the range

of complexities that the task under inspection is expected to perform and range of data sizes used for communication while considering the level of network dynamics it is expected to encounter during runtime. Thus, we propose using a gain error margin for offloading, meaning that the offloading decision is to be made when the expected gain value is greater than a value, which is more than one depending on the level of the environmental dynamics.

V. Conclusion and Outlook

This paper presented the effect of practical environments on the application offloading decision making that aims to support the end devices with the computation intensive tasks. We utilized an emulated approach in order to be able to investigate a wide range of application types and environmental conditions. In a static environment, the theoretical offloading expression was able to estimate a correct decision whenever the environmental parameters are deterministic and accurately measured. As a result, it was not successful when the computation gain and communication losses were almost equal and the slightest change in the values mattered. We also investigated the effect of more dynamic environment and were able to indicated the application types that would experience a wrong offloading decision in each scenario.

In the future, we aim to extend the work presented here by considering more parameters like the effect of RAM and GPU units on offloading. We would also want to look into the process of selecting a gain error margin for offloading. Moreover, we aim to investigate the use of artificial intelligence to overcome the effect of dynamic environment.

ACKNOWLEDGMENT

This work has been supported by the Federal Ministry for Economic Affairs and Energy of the Federal Republic of Germany (Foerderkennzeichen 01MT20001G, pfp GEMIMEG-II). The authors alone are responsible for the content of the paper.

References

[1] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*, 2012, pp. 13–16.

[2] S. Yi, C. Li, and Q. Li, "A survey of fog computing: concepts, applications and issues," in *Proceedings of the 2015 workshop on mobile big data*. ACM, 2015, pp. 37–42.

[3] N. I. M. Enzai and M. Tang, "A taxonomy of computation offloading in mobile cloud computing," in *2014 2nd IEEE international conference on mobile cloud computing, services, and engineering*. IEEE, 2014, pp. 19–28.

[4] O. Munoz, A. Pascual-Iserte, and J. Vidal, "Optimization of radio and computational resources for energy efficiency in latency-constrained application offloading," *IEEE Transactions on Vehicular Technology*, vol. 64, no. 10, pp. 4738–4755, 2014.

[5] P. Mach and Z. Becvar, "Mobile edge computing: A survey on architecture and computation offloading," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 3, pp. 1628–1656, 2017.

[6] N. Kiran, C. Pan, S. Wang, and C. Yin, "Joint resource allocation and computation offloading in mobile edge computing for sdn based wireless networks," *Journal of Communications and Networks*, 2019.

[7] H. Wu, "Multi-objective decision-making for mobile cloud offloading: A survey," *IEEE Access*, vol. 6, pp. 3962–3976, 2018.

[8] H. Wu, W. Knottenbelt, and K. Wolter, "An efficient application partitioning algorithm in mobile environments," *IEEE Transactions on Parallel and Distributed Systems*, 2019.

[9] Y. Zhang, H. Liu, L. Jiao, and X. Fu, "To offload or not to offload: An efficient code partition algorithm for mobile cloud computing," in *2012 IEEE 1st International Conference on Cloud Networking (CLOUDNET)*. IEEE, 2012, pp. 80–86.

[10] J. Wang, J. Pan, F. Esposito, P. Calyam, Z. Yang, and P. Mohapatra, "Edge cloud offloading algorithms: Issues, methods, and perspectives," *ACM Computing Surveys (CSUR)*, vol. 52, no. 1, pp. 1–23, 2019.

[11] M. Whaiduzzaman, A. Gani, N. B. Anuar, M. Shiraz, M. N. Haque, and I. T. Haque, "Cloud service selection using multicriteria decision analysis," *The Scientific World Journal*, vol. 2014, 2014.

[12] M. R. Mallick, "A comparative study of wireless protocols with li-fi technology: A survey," in *Proceedings of 43rd IRF International Conference*, 2016, pp. 8–12.

[13] K. Kumar and Y.-H. Lu, "Cloud computing for mobile users: Can offloading computation save energy?" *Computer*, no. 4, pp. 51–56, 2010.

[14] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 4, pp. 2322–2358, 2017.

[15] S. Melendez and M. P. McGarry, "Computation offloading decisions for reducing completion time," in *2017 14th IEEE Annual Consumer Communications & Networking Conference (CCNC)*. IEEE, 2017, pp. 160–164.

[16] R. P. Weicker, "Dhrystone: a synthetic systems programming benchmark," *Communications of the ACM*, vol. 27, no. 10, pp. 1013–1030, 1984.

[17] L. B. N. Laboratory, "iperf3," https://iperf.fr/iperf-download.php, accessed: 2020-03-16.

[18] S. S. Bert Hubert, Jacco Geul, "Wondershaper," https://www.tecmint.com/wondershaper-limit-network-bandwidth-in-linux/, accessed: 2020-03-16.

2021 Joint European Conference on Networks and Communications & 6G Summit (EuCNC/6G Summit) | 978-1-6654-1526-2/21/$31.00 ©2021 IEEE | DOI: 10.1109/EUCNC/6GSUMMIT51104.2021.9482556

Empirical Investigation of Offloading Decision Making in Industrial Edge Computing Scenarios

Alexander Artemenko*, Ismail Mehrez†, Keerthana Govindaraj*†, Andreas Kirstaedter† and Mykola Kuznietsov‡
*Robert Bosch GmbH, Corporate Sector Research and Advance Engineering, Renningen, Germany
{alexander.artemenko, keerthana.govindaraj}@de.bosch.com
†University of Stuttgart, Institute of Communication Networks and Computer Engineering, Stuttgart, Germany
andreas.kirstaedter@ikr.uni-stuttgart.de
‡Institute of Computer Systems, Odessa National Polytechnic University, Odessa, Ukraine, kuznietsov@opu.ua

***Abstract*—Edge Computing (EC) is a paradigm introduced to support the end devices with the execution of computation intensive tasks while maintaining their intended Quality of Service (QoS). To achieve this, one or more Edge Servers (ES) with powerful capabilities are placed in a close proximity to the edge devices to provide the assistance needed. In order to do that, EC utilizes the concept of application offloading, which is the idea of moving the computation intensive tasks to be computed on a more powerful server.**

The act of offloading is not always a beneficial choice due to the aspect of availability of involved resources and data communication between devices. Therefore, to achieve a successful offloading process, the offloading decision making needs to answer the four questions of when, what, where and how to offload. In this paper, we investigate the "When to offload?" question, which is concerned with whether the offloading process results in a positive gain in performance or not. To strengthen our conclusions, we use empirical observations in a real setup running a set of emulated applications.

***Index Terms*—Edge Computing, Application Offloading, Task Complexity, Offloading Gain**

I. Introduction

In the recent years, the Edge Computing (EC) paradigm has emerged as an extension to the Cloud Computing (CC) and is also sometimes referred to as fog computing [1], [2]. The main goal behind EC is to provide powerful services with high availability and reliability while maintaining a minimum delay for latency critical and control applications for the end devices. In order to achieve this desired goal, the concept of application offloading is utilized and is defined as moving the computation intensive tasks away from the local device to be processed on a more powerful server [3], [4]. Such Edge Servers (ES) are often supported with an enormous amount of processing capabilities such as Central Processing Units (CPU), Graphical Processing Units (GPU) as well as Random Access Memories (RAM) [5] and are placed in a close proximity to the end devices. Not only that, but these servers are supported with a constant power supply in most of the cases and thus energy consumption is not a concern. On the contrary, the end devices are often restricted by a limited amount of processing power and in a lot of cases are mobile and are connected to a mobile power sources [6], therefore introducing a major concern of energy consumption.

Although application offloading is expected to save computation time and energy by moving the complex tasks to other servers, it also introduces a new overhead on the network due to the transfer of data to be processed. This results in increase of time and energy costs [4]. As a result, in case if the computation effort saved is more than the one lost due to communication, then a gain state is achieved which is the goal of offloading. Otherwise, an offloading of this particular task will lead to a degraded performance. Moreover, the decision of offloading is always environment dependent. For instance, a certain low-end device might not be able to handle a specific task, therefore offloading is a reasonable choice, while on the other hand, a high end device can process it successfully and might even lose performance if offloading is done due to the extra unnecessary communication overhead. This introduces the challenge of choosing the appropriate task in a given application and deciding whether it would provide a gain or loss in the performance when offloaded to a certain cloud or edge server available in the network [2].

An optimal solution to the offloading decision problem is the one that achieves the best performance possible at a given cost, but unfortunately it does not have a straight forward solution. It rather needs the answer to four critical sub-questions: When, What, Where and How to offload [7]. The "When to offload?" question has to decide when should the application be offloaded. In other words, it states whether it is better to proceed with the offloading process or settle for the local execution option under the current circumstances. Moreover, many applications could be divided into more than two modules and therefore, a successful offloading mechanism has to decide "What to offload" [8]–[10]. Another aspect is "Where to offload". The answer to this question decides the destination on which the selected tasks should be offloaded to and that achieves the highest gain [11]. Finally, the "How to offload?" question is concerned by the means of offloading including the offloading channel and mechanism [12].

Multiple research works, which we present in Section II, made a considerable effort in solving the offloading decision problem and came up with a closed-form expression that would assist in making such decisions. As a result, the goal of this paper is to investigate the reliability of this expression and to validate that it works when tested in real-life conditions

using practical methods as expected from the theory. Another goal of this work is to do a sensitivity analysis to find out how the decision making is affected by a dynamic environment.

The remainder of this paper is organized as follows. In Section II, we introduce the related work and the state of the art related to the topic of application offloading. We then present our work concerning the offloading decision making in Section III describing the emulation and offloading processes and then evaluate our work in both static and dynamic environments in Section IV. Finally in Section V, we conclude our work and suggest some future steps.

II. Related Work

The offloading decision can be intuitive in cases with extremely complex tasks that require intensive processing and at the same time have a very limited amount of input data to be transmitted. In such scenarios, offloading would result in a gain in performance since a lot of computation effort is saved while a relatively small communication effort is added. On the other hand, offloading a task with a huge amount of transmission load will cause a degraded performance.

Multiple efforts were done in order to formulate a closed-form expression that facilitates making the decision of either to offload a specific task or not under the current state of the surrounding environment to achieve the desired offloading gain [7], [13]–[15]. The main condition that needs to be met for the offloading to be beneficial is that the cost of executing a task locally C_{mobile} should be more than the one when outsourcing it to a server for processing. The cost of performing the task on the server side is represented in terms of the cost of computation C_{server} and the cost to communicate the data between the devices C_{com}. This is represented as [7]:

$$C_{\text{mobile}} > C_{\text{server}} + C_{\text{com}} \tag{1}$$

The definition of the cost is made depending on the desired objective. When the objective is to minimize the overall performance time, Wu et al. [7] defined the condition for a rewarding offloading is;

$$\frac{C}{IPS_m} > \frac{1}{F}.\frac{C}{IPS_m} + \frac{D}{B}, \tag{2}$$

where C is the number of instructions of the task to be offloaded, IPS_m is instructions per second capability of the mobile client, D is the total amount of data transmitted between the nodes and B is the available bandwidth on the network link between them. For simplicity, a certain server s is assumed to be F times more powerful than a certain mobile client m.

Although the offloading expression seems straight forward, the results of the research mentioned were only based on pure simulations and do not take into account the dynamic nature of the surroundings. The theoretical approach assumes an exact knowledge of the resources state. Therefore, the work done lacks practical verification in order to state if the inequality holds when used in a real-life environment.

III. Offloading Concept

Our goal is to design an emulator that would practically reflect the behavior of a real-life scenario, where application offloading can be performed by a mobile device. Using the emulator, a wide range of applications with different settings for a computation complexity and a payload can be tested in a real practical environment. To indicate a certain task complexity, we used the Dhrystone benchmark [16]. This standard is used as a representation of the integer performance of a general purpose CPU in terms of VAX (Virtual Address eXtension) Dhrystone Million Instructions per Second (DMIPS), or in other words, by how much is this CPU more powerful than the standard 1 MIPS VAX machine.

To emulate the input and output data transferred between the client and server nodes, we prepare a message with a size that ranges between 1 B and 10 MB and send it over to the server side using TCP, where the computation would take place. By doing this, we are able to represent a wide range of application types starting from the ones using sensor data with few bytes up to the ones that transfer high resolution camera frames. Finally, we used the Linux tool iPerf3 [17] and WonderShaper [18] to emulate different network bandwidth availability. These tools have the ability to limit the data rate on a specific network interface. Thus, we can emulate different network conditions.

To observe if under a given combination of system parameters (Task Complexity, Message Size, Bandwidth, CPU) the offloading process would be beneficial, we first run both scripts on the mobile node in order to emulate the scenario when the task is not offloaded and we would get the local performance of the task which in our work was local completion time t_{local}. Then, we move the server script to the server side to emulate the offloaded case and we would get an offloaded performance time t_{offload}. The goal of this step is to check if the cost saved due to a more powerful computing platform is more than the one lost due to communicating the message through the network with a given availability. Finally, we define a gain in performance in the form of a ratio as:

$$Gain = \frac{t_{\text{local}}[s]}{t_{\text{offload}}[s]}, \tag{3}$$

where $t_{\text{local}} > 0$ and $t_{\text{offload}} > 0$. Hence, a gain value that is more than one indicates that the system performed better (faster) when an offloading took place, while a gain less than one means a degraded outcome.

IV. Evaluation Results

For our emulation and results discussed next, we used the devices specified in Table I. A Raspberry Pi model 3B device with a Cortex-A53 CPU acts as a mobile device that sends data D with sizes that range from 1 B to 10 MB to the server. On the other hand, we used a powerful HP Z-Book 15 machine with an Intel(R) Core(TM) i7-4810MQ CPU as the server node. Therefore, the server processing capability is $F \approx 11$ times more powerful than the client's one (23741.92 MIPS over 2162.82 MIPS). This server has the task of performing

an emulated computation complexity C that ranges from 0 to $600\text{x}10^6$ instructions. The measured available bandwidth between both devices was B=94 Mbps as the link between the access point and the client used is IEEE 802.11 (5 GHz) WiFi link and the link to the server is a 1 Gbps Ethernet connection.

TABLE I: A summary of the devices and infrastructure used for the emulation process.

Parameter	Value
Client	
Device	Raspberry Pi 3B
CPU Model	Cortex-A53
CPU MIPS	2162.82 MIPS
Operating System	Raspbian Buster Lite
Data Range	1 B - 10 MB
Transport Protocol	TCP
Infrastructure	
Resultant Bandwidth	94 Mbps
WiFi Link	802.11ac (5 GHz)
Ethernet Link	1 Gbps
Server	
Device	HP-ZBook 15
CPU Model	Intel(R) Core(TM) i7-4810MQ
CPU MIPS	23741.92 MIPS
Operating System	Ubuntu 18.04
Task Complexity	$0 - 600\text{x}10^6$ Instructions

A. Static Environment

In this subsection, we demonstrate our insights and observations when the offloading decision making is done in a static environment, where the measured parameters remain deterministic within the whole process and only minor alternations from the measured mean values take place as part of the environment, in which we observe.

Fig. 1 illustrates our initial findings when we emulated the whole range of complexities and message sizes in our previously described lab setup. The x-axis of the figure represents the range of task complexities used in terms of their number of instructions. The y-axis is the range of different message sizes that are to be transmitted between the devices and is shown in Bytes. Moreover, the color of the grid represents the gain we observed using Equation 3. An observed degraded performance while offloading with a gain value between 0 and 1 is marked in red, while an enhanced performance is marked in colors from yellow till green. The gradient of the colors represents the value of the gain which is demonstrated in the color scale on the right of the figure. For example, a dark red point means an extreme loss in performance while a light red means a slight loss, and the same applies for the green ones.

The blue diagonal linear line represents the theoretical boarder line that defines the offloading decision regions from Expression 2. Therefore, according to this polynomial, for any task with a given complexity in this environment, an offloading decision would lead to a gain (enhanced performance) if a message size to be transferred is below that line, while a loss is to be observed otherwise.

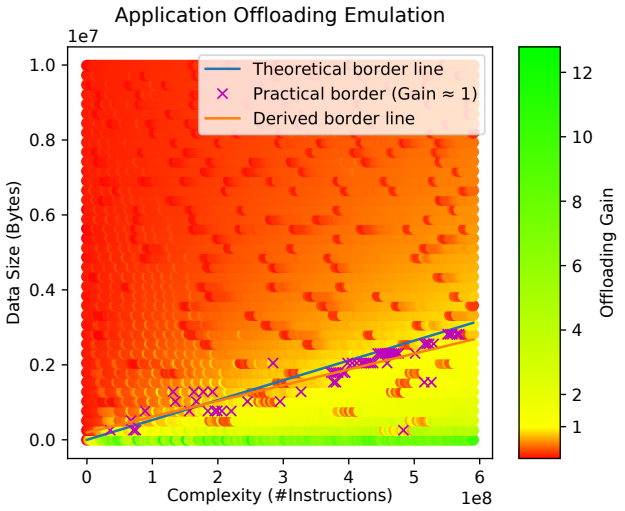

Fig. 1: The output of the offloading emulation process.

In our described emulation, we always offloaded the selected task and compared it's performance versus the case when it is not offloaded. Therefore, the cases which were successful are the ones in green and yellow while the failures are in orange and red. We can observe that the cases with extremely low message sizes resulted in a very high offloading gain reaching over a ten times enhanced performance, which can be explained as a result of the very low overhead introduced by the communication. It can also be seen that the gain value increases as the task gets more complex due to the increase in the computation cost saving. On the other hand, cases with very high message sizes resulted in high loss in communication effort and therefore offloading would not be beneficial. Furthermore, we also observed that offloading is never good for a very simple task with a very low complexity even if the message payload size is low because such tasks could be easily handled by the mobile device and any offloading would just mean an extra overhead.

The interesting region, where the offloading decision does not have an intuitive solution, is the area in the middle of the graph, where the values of the communication cost loss and the computation cost saved are comparable. This area is the one with dark yellow and light red points since the difference in the local and offloaded performance was not significant and therefore resulted in a gain value around one, which can be seen from Equation 3.

Fig. 1 shows that Equation 2 is able to separate the regions with performance loss and gain successfully. For a given task complexity, a payload size below the theoretic blue line practically leads to a gain in most of the cases while a value on the other side of the line leads to a loss in performance.

We could recognize that there are some outliers in the figure with a loss below the line and a gain above it, which would mean that there are cases where the expression is not reliable. By investigating this phenomenon, we suspected that our environmental variables at these cases were fluctuating

around the mean value due to the practical essence of our analysis. Such fluctuations could be a sudden drop or increase in the bandwidth between the devices due to a congestion or attenuation on the WiFi link. Another cause might be that the CPU serving this task on either nodes was not dedicating all its power to this task due to a sudden interrupt from the operating system or it was being shared by another process.

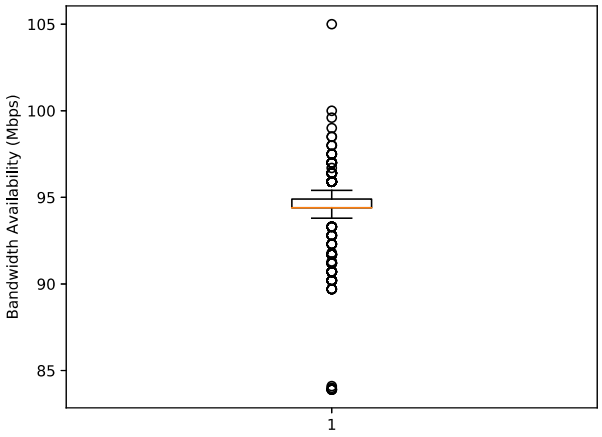

Fig. 2: Network status monitor.

To verify our suspicion and eliminate the effect of any sudden disruption in our environment, we monitored the available bandwidth as a test parameter during runtime and the result can be seen in Fig. 2. We observed a fluctuation in the availability around the mean value. Moreover, we ran each combination of payload and complexity five times and took the average in performance between all five trials, which can be seen in Fig. 3.

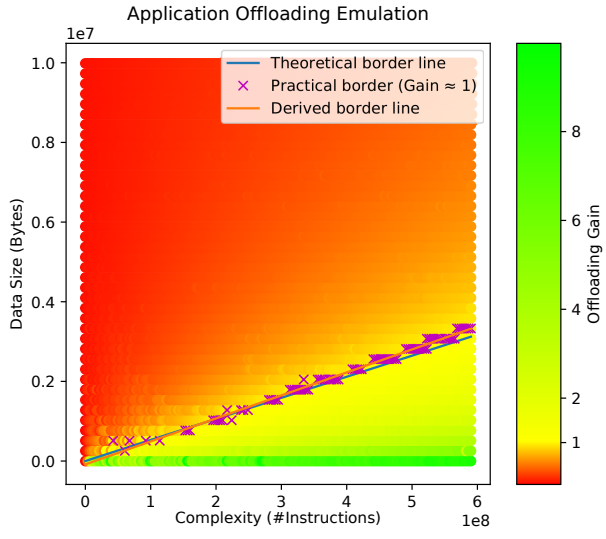

Fig. 3: Offloading emulation result using the average value of five iterations.

We observed from Fig. 3 that the offloading gain was changing in a constant manner and that the color grid became much smoother compared to the one in Fig. 1, where we used the result of only a single iteration. This observation made us realize that the outliers observed previously were actually a result of an unexpected change in the environment that happened at a particular moment.

It is now clear that the expression can be reliable whenever the cost of computing the task locally is significantly larger than the transfer cost. But we wanted to further investigate if the expression is reliable in the scenarios, in which the difference is small and it is not intuitive to make the offloading decision. To do that, we identified all the cases with offloading gain being close to one ($0.98 < \text{gain} < 1.02$) and marked them with the magenta crosses seen in Fig. 3. These cases represent the cases with a very low gain difference between local and offloaded performance being very close to the theoretical boarder line defined by Equation 2. We fitted a polynomial labeled derived borderline to those points, which we will use as our practical baseline. By comparing our derived boarder line to the theoretical one, we can observe a slight deviation between both lines. Since the theoretical expression is a linear polynomial, the gradient of the line is highly dependent on the accuracy of the parameters of our environment given, which are the CPU capabilities of our nodes and the network bandwidth available. As a result, we conclude that at the cases, in which the local and offloading costs are similar, the offloading decision making highly depends on the accuracy of the measured parameters.

B. Dynamic Environment

In this section, we present our results and observations when emulating different offloading examples as before but in more dynamic environments, where a measured parameter used in the decision making would not be the most recent value during the actual offloading. Such environments can be one with a mobile client device roaming around a factory floor causing attenuations or multiple connection handovers between access points due to the mobility aspect. Another case could be a multi-user scenario, in which the server keeps getting numerous requests from other devices and thus cannot dedicate all its processing power to the specified task. Any of the mentioned examples can result in a sudden drop of our parameters, required for the decision making. In this work, we focused on emulating an environment with sudden drops of the network bandwidth that becomes non-deterministic and analyzed its effect on the offloading decision making.

To analyze the described effect of the dynamic network availability on the decision making, we ran our whole emulation process multiple times and we changed the available network bandwidth in each of them. Fig. 4 illustrates our results for emulating different levels of sudden bandwidth drops that result in unpredictable changes of the data rate availability. While the decision making algorithm assumes the full bandwidth (94 Mbps) to be available, we on purpose overload the network and thus limit the bandwidth (BW)

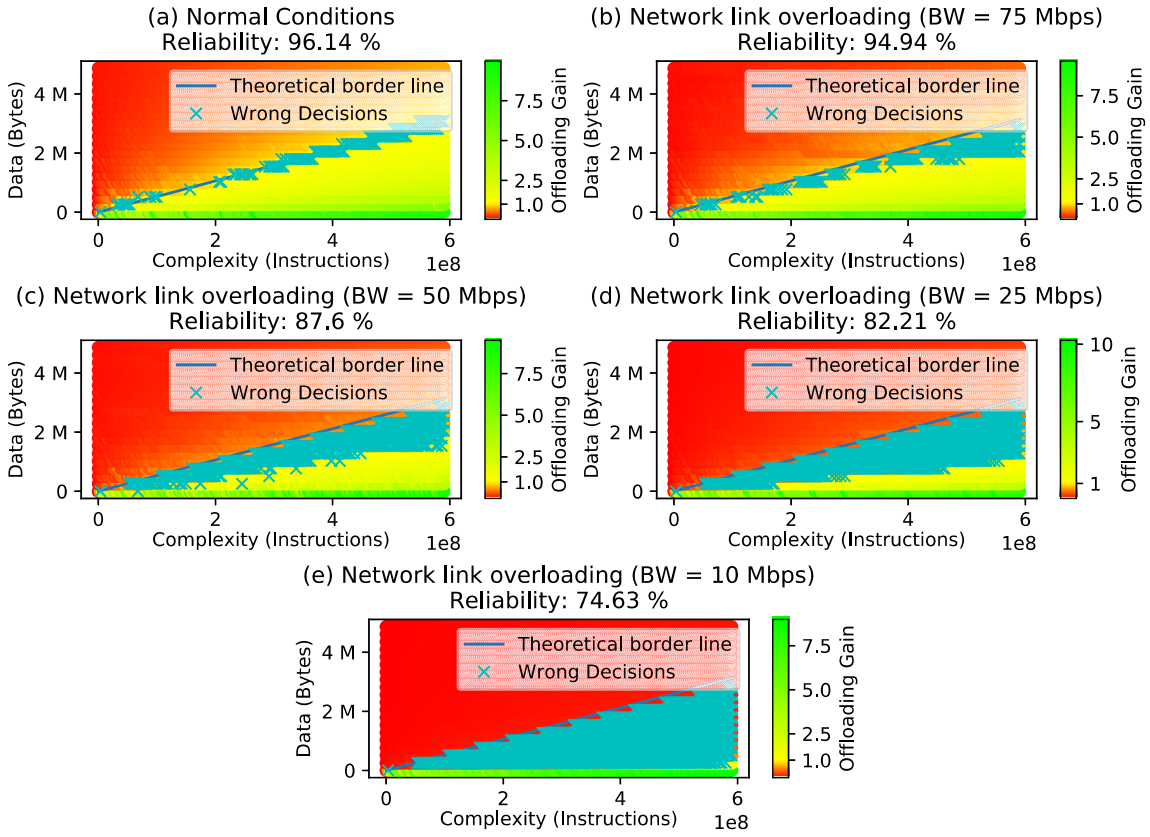

Fig. 4: Emulation results for different levels of a dynamic environment.

to 75, 50, 25 and 10 Mbps correspondingly. Moreover, the blue polynomial in each of the results is the decision maker result from our theoretic expression in Equation 2 using the full expected bandwidth value. By this methodology, we can have a better understanding about what would happen if we anticipated to have a certain bandwidth value (100% value), but in the real case, it dropped to the availability level being emulated.

We also defined a metric called Offloading Reliability, which is the percentage of correct decisions that the offloading decision maker would have made when used on all the range of application types we emulate. Although this value is relative to the range of complexities and payloads defined, we just use it as an assistance guide to help evaluate the correctness of our decision making not as a direct reference. To identify a correct decision, we contrasted the result from our theoretic decision maker in Equation 2 to the actual gain we practically observed. So if the expression anticipated a gain in performance and our emulated run showed a gain value more than one, then it is considered a correct decision. Similarly, an experienced gain value less than one and an anticipated loss in performance is also a correct decision because in that case offloading will not take place and therefore the system performance will not be degraded. On the other hand, if the results were different, this would be considered a false decision because when relying on the expression in such cases, we might offload and lose performance or even in some cases choose to perform the task locally when a positive gain from offloading could be achieved. We illustrated these wrong decisions by the cyan crosses on the figures.

Subplot (a) in Fig. 4 represents a scenario without bandwidth drops and hence a maximum reliability with a very little number of wrong decisions observed. Similarly, as the observation from Section IV-A, we got a number of wrong decisions around the boarder line defined by the theoretic expression that represent 3.9% of all our cases and therefore the expression is reliable in 96.1% of our test cases.

In subplots (b), (c), (d) and (e), we show the results of our emulation when we limited the available bandwidth to 75, 50, 25 and 10 Mbps respectively. We can clearly observe that the region of wrong decisions increases with the decrease of the network availability. Moreover, the wrong decisions are clearly concentrated below the theoretical boarder line since the bandwidth available is always less than expected and the offloading cost got higher thus affecting the cases with a decision to offload.

The main contribution of this subsection is that we were able to illustrate the range of applications (marked in cyan in Fig. 4) that would experience a false offloading decision when a certain level of network dynamics is present. Therefore, application and system architects can use diagrams like Fig. 4 as a heuristic guide to decide when to offload using the range

of complexities that the task under inspection is expected to perform and range of data sizes used for communication while considering the level of network dynamics it is expected to encounter during runtime. Thus, we propose using a gain error margin for offloading, meaning that the offloading decision is to be made when the expected gain value is greater than a value, which is more than one depending on the level of the environmental dynamics.

V. Conclusion and Outlook

This paper presented the effect of practical environments on the application offloading decision making that aims to support the end devices with the computation intensive tasks. We utilized an emulated approach in order to be able to investigate a wide range of application types and environmental conditions. In a static environment, the theoretical offloading expression was able to estimate a correct decision whenever the environmental parameters are deterministic and accurately measured. As a result, it was not successful when the computation gain and communication losses were almost equal and the slightest change in the values mattered. We also investigated the effect of more dynamic environment and were able to indicated the application types that would experience a wrong offloading decision in each scenario.

In the future, we aim to extend the work presented here by considering more parameters like the effect of RAM and GPU units on offloading. We would also want to look into the process of selecting a gain error margin for offloading. Moreover, we aim to investigate the use of artificial intelligence to overcome the effect of dynamic environment.

ACKNOWLEDGMENT

This work has been supported by the Federal Ministry for Economic Affairs and Energy of the Federal Republic of Germany (Foerderkennzeichen 01MT20001G, pfp GEMIMEG-II). The authors alone are responsible for the content of the paper.

References

[1] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*, 2012, pp. 13–16.

[2] S. Yi, C. Li, and Q. Li, "A survey of fog computing: concepts, applications and issues," in *Proceedings of the 2015 workshop on mobile big data*. ACM, 2015, pp. 37–42.

[3] N. I. M. Enzai and M. Tang, "A taxonomy of computation offloading in mobile cloud computing," in *2014 2nd IEEE international conference on mobile cloud computing, services, and engineering*. IEEE, 2014, pp. 19–28.

[4] O. Munoz, A. Pascual-Iserte, and J. Vidal, "Optimization of radio and computational resources for energy efficiency in latency-constrained application offloading," *IEEE Transactions on Vehicular Technology*, vol. 64, no. 10, pp. 4738–4755, 2014.

[5] P. Mach and Z. Becvar, "Mobile edge computing: A survey on architecture and computation offloading," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 3, pp. 1628–1656, 2017.

[6] N. Kiran, C. Pan, S. Wang, and C. Yin, "Joint resource allocation and computation offloading in mobile edge computing for sdn based wireless networks," *Journal of Communications and Networks*, 2019.

[7] H. Wu, "Multi-objective decision-making for mobile cloud offloading: A survey," *IEEE Access*, vol. 6, pp. 3962–3976, 2018.

[8] H. Wu, W. Knottenbelt, and K. Wolter, "An efficient application partitioning algorithm in mobile environments," *IEEE Transactions on Parallel and Distributed Systems*, 2019.

[9] Y. Zhang, H. Liu, L. Jiao, and X. Fu, "To offload or not to offload: An efficient code partition algorithm for mobile cloud computing," in *2012 IEEE 1st International Conference on Cloud Networking (CLOUDNET)*. IEEE, 2012, pp. 80–86.

[10] J. Wang, J. Pan, F. Esposito, P. Calyam, Z. Yang, and P. Mohapatra, "Edge cloud offloading algorithms: Issues, methods, and perspectives," *ACM Computing Surveys (CSUR)*, vol. 52, no. 1, pp. 1–23, 2019.

[11] M. Whaiduzzaman, A. Gani, N. B. Anuar, M. Shiraz, M. N. Haque, and I. T. Haque, "Cloud service selection using multicriteria decision analysis," *The Scientific World Journal*, vol. 2014, 2014.

[12] M. R. Mallick, "A comparative study of wireless protocols with li-fi technology: A survey," in *Proceedings of 43rd IRF International Conference*, 2016, pp. 8–12.

[13] K. Kumar and Y.-H. Lu, "Cloud computing for mobile users: Can offloading computation save energy?" *Computer*, no. 4, pp. 51–56, 2010.

[14] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 4, pp. 2322–2358, 2017.

[15] S. Melendez and M. P. McGarry, "Computation offloading decisions for reducing completion time," in *2017 14th IEEE Annual Consumer Communications & Networking Conference (CCNC)*. IEEE, 2017, pp. 160–164.

[16] R. P. Weicker, "Dhrystone: a synthetic systems programming benchmark," *Communications of the ACM*, vol. 27, no. 10, pp. 1013–1030, 1984.

[17] L. B. N. Laboratory, "iperf3," https://iperf.fr/iperf-download.php, accessed: 2020-03-16.

[18] S. S. Bert Hubert, Jacco Geul, "Wondershaper," https://www.tecmint.com/wondershaper-limit-network-bandwidth-in-linux/, accessed: 2020-03-16.