Computational Offloading for Non-Time-Critical Applications

Richard Patsch
University of Applied Sciences
Technikum Wien
Vienna, Austria
richard.patsch@technikum-wien.at

***Abstract*—The increasing demand for computational resources keeps outpacing available User Equipment (UE). To overcome intrinsic hardware limitations of UEs, computational offloading was proposed. The combination of UE and seemingly endless computational capacity in the cloud aims to cope with those limitations. Numerous frameworks leverage Edge Computing (EC) but a significant drawback of this is the required infrastructure. Some use cases however, do not benefit from lower response time and can remain in the cloud, where more potent resources are at one's disposal. Main contributions are to determine computational demands, allocate serverless resources, partition code and integrate computational offloading into a modern software deployment process. By focusing on non-time-critical use cases, drawbacks of EC can be neglected to create a more developer-friendly approach. Originality lies in the resource allocation of serverless resources for such endeavours, appropriate deployment of partitions and integration into CI/CD pipelines. Methodology used will be Design Science Research. Thus, many iterations and proof-of-concept implementations yield knowledge and artefacts.**

***Index Terms*—computational offloading, mobile cloud computing, partitioning, serverless**

I. Introduction

Although the performance of mobile phones and Internet of Things (IoT) devices is rising, emerging use cases such as image recognition, monitoring, and intense communication [1] can overwhelm the executing devices. Besides timing, energy usage must be considered, since some of those devices depend on a battery. One possible solution to alleviate the drain on battery and Central Processing Unit (CPU) is to offload the computations [2]. Computational Offloading (CO) requires infrastructure and effort, as well as sophisticated mechanisms. Depending on what the initial reasoning for that offloading was, several metrics need to be considered. Challenges in EC and the lack of developer-friendly workflows in OC are the reason this concept has not spilled over to the industry yet. The combination of the latest theoretic offloading insights with modern CI/CD lacks is rarely researched on.

II. Contribution

Concrete problems that meet the criteria of this endeavour are: (i) not time-critical but (ii) still benefiting from CO compared to a regular web service. Potential use cases could have a time/precision trade-off such as scientific computing. Key idea of this dissertation is to introduce the concept of CO to a broader audience by automating tedious tasks such as partitioning and resource partitioning to later integrate them into modern CI/CD pipelines. Main steps are (i) determine computational demand, (ii) partition for offloading, (iii) deploy partitions, (iv) handle communication between partitions and (v) automate it in a CI/CD pipeline. In comparison to existing work, the goal is to create a heuristic approach without making it equivalent to an Application Programming Interface (API). Focusing on non-time-critical applications allows to neglect the usage of EC and therefore alleviates the burden of dealing with the involved challenges. Furthermore, leveraging cloud computing increases the target audience, since no additional infrastructure is required and computational resources can be provisioned by every cloud service provider.

III. State of the Art

Main considerations when building an *offloadable* application are (i) granularity, (ii) mobility and (iii) destination. Granularity determines how *big* the partitions are. Mobility defines where the computational resources are situated and how movement of the UE is dealt with. The cloud provides more resources but also adds more latency, which rules out a few use cases. To cope with the introduced latency, it is possible to leverage EC, which is in close proximity to the UE but then hand-overs as used in mobile phone networks are necessary to deal with movement. The destination reveals whether just one or multiple servers are assisting the constrained device. [3]. Offloading frameworks such as [4]–[6] lack public availability of a working implementation.

Cloud Computing frameworks provide ubiquitous and rich functionality, regardless of the resource limitations of mobile devices [7]. Serverless computing can handle heterogeneous devices, with different computational capabilities, and is expected to grow across the IoT-Fog-Edge-Cloud systems as an extension of current cloud-based implementations to IoT applications [8]. Serverless capabilities rely on vendor specific solutions, so generalisations rely on abstractions, as in [9]. Resource allocation becomes more challenging as the number of variables increase, as well as when these variables change more often over time.

This work has been supported by the Doctoral College Resilient Embedded Systems, which is run jointly by the TU Wien's Faculty of Informatics and the UAS Technikum Wien.

The real competitor of CO is a common client-server web API principle. It is simple and for most cases sufficiently fast. Using serverless computing or looking further into provisioning more suitable resources in the cloud is something that are relatively unexplored options.

IV. Originality

Adapting to emerging opportunities, the usage of serverless computing deserves more attention. The main difference in this dissertation to already published work is that it will focus on non-time-critical use cases and closely inspect better ways for resource allocation, particularly toward serverless computing for different categories of applications. The envisioned result has the following workflow: (i) a suitable application that is designed according to the resulting guidelines is added to a repository, (ii) dynamic analysis will be performed to find appropriate resources to provision, (iii) (serverless) resources will be allocated and necessary services are set up, (iv) tests will be run, and (v) deployed serverless functions are ready for invocation. Proposed workflow aims to integrate the whole offloading process into a modern CI/CD pipeline and automate as many steps as possible to finally make the concept of CO more palatable to developers. It will combine the convenience of an API with the resource related opportunities of CO and thus, provide a better software engineering process.

V. Research Questions

#RQ1 How to determine the bottleneck on the local machine for the task at hand, provision adequate resources for assistance and deal with costs?
Answering this question involves static- or dynamic code analysis as seen in [10]–[12]. This has to be the first step since one of the first decisions is which resources should be provisioned. Even when using serverless computing, executing containers have to be chosen judiciously. To cope with the pricing aspect, [13] tried to answer this question with a mathematical approach. However, determining the cost of offloading is not trivial when taking into account more metrics than just the price tag of the provisioned resource itself.

#RQ2 How is code ideally prepared/partitioned for distribution? Which problems require which partitions and are there problems that can't be distributed?
Partitioning is not a new discipline and had also to be considered for a regular distributed system. Particularly interesting here, is if there are additional requirements when dealing with a cloud environment or serverless computing.

#RQ3 How can these partitions be deployed on different platforms such as EC2-instances or lambda functions.
Deployment to EC2-instances is easier than deploying lambda functions since the lambda environment is more limited and also has a time limit of 15 minutes. After this time frame, the function terminates, regardless of the progress made.

#RQ4 How should data exchange and communication between instances be handled? Via the local instance or within the cloud?
Especially parallelised functions that run on more than one server need to communicate. Possibly, via the invoking device or by using the cloud infrastructure. By setting up the required security policies to allow the instances to communicate with each other one could get rid of the additional communication between the local instance and the cloud.

#RQ5 Can such a code deployment be done automatically, s.t. this option can be migrated into a CI/CD pipeline?
Build a framework that automates and *answers* #RQ1 - #RQ4 as much as possible to a degree where the developer does not need to worry too much about the *magic* that happens when offloading. Of course there need to be manually set parameters since offloading descisions' veracity is individual.

References

[1] A. Mehonic and A. Kenyon, "Brain-inspired computing: We need a master plan," 04 2021.

[2] K. Kumar and Y.-H. Lu, "Cloud computing for mobile users: Can offloading computation save energy?" *Computer*, vol. 43, pp. 51–56, 2010.

[3] J. Wang, J. Pan, F. Esposito, P. Calyam, Z. Yang, and P. Mohapatra, "Edge cloud offloading algorithms: Issues, methods, and perspectives," vol. 52, no. 1, 2019.

[4] E. Cuervo, A. Balasubramanian, D.-k. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, "Maui: Making smartphones last longer with code offload." New York, NY, USA: Association for Computing Machinery, 2010.

[5] S. Kosta, A. Aucinas, P. Hui, R. Mortier, and X. Zhang, "Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading," in *2012 Proceedings IEEE INFOCOM*, 2012, pp. 945–953.

[6] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, "Clonecloud: Elastic execution between mobile device and cloud." New York, NY, USA: Association for Computing Machinery, 2011.

[7] M. R. Rahimi, J. Ren, C. H. Liu, A. V. Vasilakos, and N. Venkatasubramanian, "Mobile cloud computing: A survey, state of art and future directions," *Mobile Networks and Applications*, vol. 19, no. 2, pp. 133–143, Apr 2014.

[8] "The internet of things, fog and cloud continuum: Integration and challenges," *Internet of Things*, vol. 3-4, pp. 134–155, 2018.

[9] M. Villari, M. Fazio, S. Dustdar, O. Rana, and R. Ranjan, "Osmotic computing: A new paradigm for edge/cloud integration," *IEEE Cloud Computing*, vol. 3, no. 6, pp. 76–83, 2016.

[10] P. Hellström, "Tools for static code analysis: A survey," p. 119, 2009.

[11] P. Korhonen and M. Syrjänen, "Resource allocation based on efficiency analysis," *Management Science*, vol. 50, no. 8, pp. 1134–1144, 2004.

[12] M. Verma, G. R. Gangadharan, N. Narendra, R. Vadlamani, V. Inamdar, L. Ramachandran, R. Calheiros, and R. Buyya, "Dynamic resource demand prediction and allocation in multi-tenant service clouds," *Concurrency and Computation: Practice and Experience*, vol. 28, 12 2016.

[13] A. u. R. Khan, M. Othman, N. Khan, J. Shuja, and S. Mustafa, "Computation offloading cost estimation in mobile cloud application models," *Wireless Personal Communications*, vol. 97, p. 4897–4920, 12 2017.