Optimal Computation Offload Policies For Mobile Edge Computing

UP781439

MEng Computer Science

1. Abstract

Mobile devices tend to be more limited than their non-mobile counterparts, due to factors such as memory and computational speed. These limitations can be mitigated through intelligent computation offloading, which is the concept of sending computationally intensive processes to cloud servers, where the process can be executed much quicker. However the decision as to which processes should be offloaded causes problems because if you offload all processes the user will see a decline in performance and, conversely, if there are not enough processes being offloaded then the performance is below optimal.

2. Introduction

Mobile users today are requiring faster and faster network speeds and better quality of service (QoS). This is driven by the advancements being made in mobile technology, and it also means that these mobile devices are now having to perform more computationally intensive tasks. Although new mobile devices are more and more powerful in terms of central processing unit (CPU), even these may not be able to handle the applications requiring huge processing in a short time (Maach and Becvar, 2017).

Edge computing is a computation architecture, building upon cloud computing, where the resources of a centralised data centre are placed at the edge of the network, in close proximity to mobile devices, sensors, and end users. Terms such as "cloudlets," "micro data centers," "fog, nodes" and "mobile edge cloud" have been used in the literature to refer to these edge-located computing entities (Satyanarayanan, 2017). Mobile edge cloud (MEC) is an implementation of the edge architecture, which combines multiple mobile and stationary devices interconnected through ad hoc and infrastructure-based local area networks are combined to create a small cloud infrastructure at a local physical area such as home (Shah, 2017).

Computational offloading is the term used to refer to the delegation of intense computation of data from Internet of Things (IoT) devices that have sensors installed, to the edge-located servers that can provide cloud computing capabilities. The strategy when deciding which processes need to be offloaded often depends on many parameters such as the network bandwidths and the amounts of data exchanged through the networks (Kumar, Liu, Lu and Bhargava, 2012).

The problem is that you want to offload as many tasks as you can to the server to be processed because it will be performed faster there, however if you perform too many processes in the MEC then you can have too much throughput for the perimeter or edge network to handle and this can result in unnecessary traffic reaching the core network, which the user will experience as latency in the network. Therefore an intelligent computation offloading strategy is required to achieve better and more stable performance of MEC servers.

3. Aims and Objectives

The aim of this poster is to evaluate the literature surrounding the topic of computation offloading in edge computing, specifically the different strategies their strengths and weaknesses. The second aim is to use this research to propose my own strategy for computation offloading to a mobile edge cloud and to then describe and implement an experiment that can simulate my solution. Finally, the results of the experiment will be presented and evaluated and I will propose my own strategy based off the findings that will combine the strengths of the other.

4. Background

Mobile Edge Computing is designed to push resources closer to the radio access networks in 5G. It brings cloud computing capabilities and IT service environment at the edge of the mobile network (Shahzadi, Iqbal, Dagiuklas and Qayyum, 2020). Cloud computing is based around a centralised data centre that provides computing resources - servers, database management, data storage, networking, software applications, and special capabilities such as blockchain and artificial intelligence (AI) - over the internet, as opposed to owning and operating those resources yourself, on premises (IBM, 2020). Cloud computing for mobile computation is extremely useful as it allows mobile devices to perform tasks that would have previously been extremely intensive. However, in the cloud computing model computation happens at data centres, often large distances from the user. This physical distance affects end-to-end latency, economically viable bandwidth, establishment of trust, and survivability (Satyanarayanan, 2017). MEC servers solve this by changing the centralised data processing model of cloud computing, to processing the data at the edge of the network (Edge Computing Resources, 2020).

Computation offloading enables energy-limited mobile devices to expand the range of applications that they can execute (Salmani, Sohraby, Davidson and Yu, 2020). There are two basic computation task offloading models in MEC, i.e., binary and partial computation offloading (Mao et al., 2017). Specifically, binary offloading requires a task to be executed as a whole either locally at the WD (wireless device) or remotely at the MEC server. Partial offloading, allows a task to be partitioned into two or more parts with at least one executed locally and one offloaded for MEC execution. In practice, binary offloading is easier to implement and suitable for simple tasks that are not partitionable, while partial offloading is favorable for some complex tasks composed of multiple parallel segments (Bi and Zhang, 2020).

With the recent advancements made in machine learning techniques, many computation offloading strategies are adopting a machine learning approach. For example, (Huang, Bi and Zhang, 2019) developed an algorithm that they call, deep reinforcement learning-based online offloading (DROO) framework to maximize the weighted sum of the computation rates of all the wireless devices. the deep neural network (DNN) takes wireless channel gain h_t as the input, and outputs X_t, a relaxed computation offloading action, which is then quantized into K binary offloading actions, among which one best action x_t^* is selected.

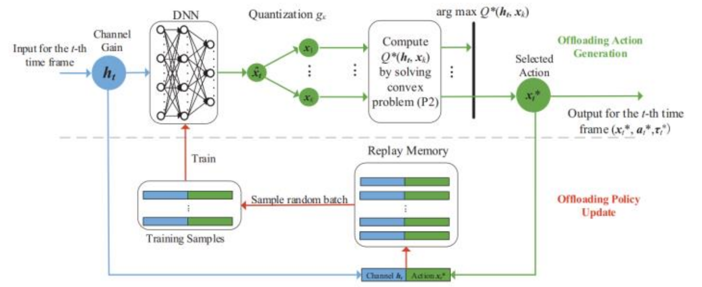

Fig 1. DROO algorithm (Huang, Bi and Zhang, 2019)

The strengths of a method such as this is that 90% of apps use cloud services (Henze et al., 2017) so there can be an immense amount of data generated on computation offloading decisions being made to a MEC server. However, the drawback to this are that it can be very time-consuming to train such a model to get reasonable performance (Cheng, Wang, Zhou and Zhang, 2019). Another drawback, is that while this is a semi-supervised learning model that can be trained on a mixture of labelled and unlabelled data, it is still very time consuming to label the results which has to be done by hand.

Another approach used to decide whether a computation should be executed on the mobile device or at the MEC server is presented in (Liu, Mao, Zhang and Letaief, 2016), where they propose a stochastic computation task scheduling policy, based on Markov chain theory. This is a binary method that assumes there are four possible options or decisions to choose from,

$$(v_C[t], v_L[t]) = \begin{cases} (0,1) & w.p.\ g_\tau^1, \\ (1,0) & w.p.\ g_\tau^2, \\ (1,1) & w.p.\ g_\tau^3, \\ (0,0) & w.p.\ (1 - \sum_{k=1}^{3} g_\tau^k). \end{cases}$$

where (0, 0) indicates that both transmission unit and local processor are idle, while (1,0) indicates that the local processor is idle, but the transmission unit is in use, (0, 1) indicates the opposite of (1.0) and (1, 1) indicates both are in use.

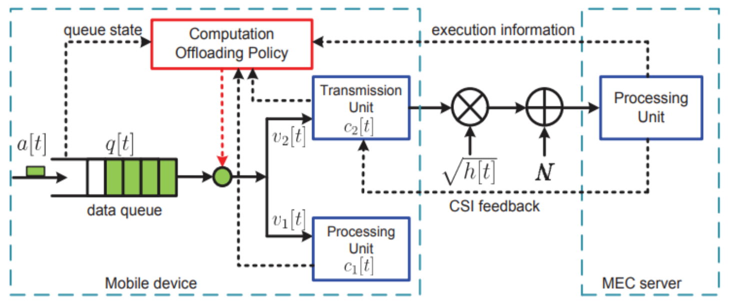

Fig 2. MEC computation offloading model (Liu, Mao, Zhang and Letaief, 2016)

This model was compared to three baseline computation offloading policies:

- All computation done locally, on the device,
- All computation offloaded to the MEC server,
- Greedy offloading policy, where the mobile device schedules the computation tasks waiting in the buffer to the local CPU or the MEC server for task executions whenever the local CPU or TU is idle.

They developed a one-dimensional search algorithm to find the optimal computation scheduling policy. It was found that their proposed model achieved the minimum average delay out of the four tested. The strengths of this model are that it is shown to offload more processes to the MEC as the rate of arrival of new processes increases. Processing speeds in the MEC server are much faster than on the mobile device. Therefore, as the arrival rate of new processes increases it is beneficial to offload a higher proportion of those to the MEC to reduce the average delay experienced by the user, figure 3 displays this.

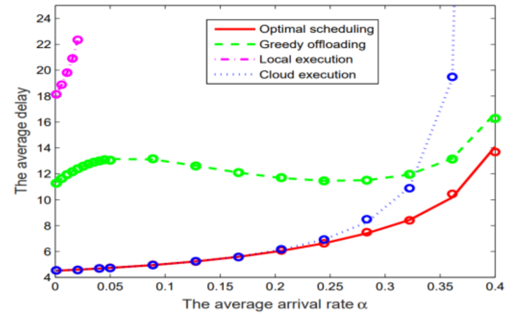

Fig 3. results of search algorithm (Liu, Mao, Zhang and Letaief, 2016)

However, the MEC system used in this experiment is quite simplistic and not applicable to most real life scenarios, as there are a lot more environmental variables involved when using wireless connections that this doesn't take into account such as physical objects that could alter strength of the wireless channels.

5. Experimental Design

Based on the research I have gathered I am proposing a binary, threshold based offloading policy that will decide which tasks get offloaded based on their computational intensity. The Experiments that were analysed during the research process did not take into account the computational intensity of the task that is being considered, which I believe should be one of the main deciding factors to increase performance. Therefore, I propose a policy that revolves around making offloading decisions based on the computation intensity, with computations above a given threshold being offloaded.

Each new computation that is created, t, will be given an integer value from 1 to 10 that will represent the computational intensity of the task, c_t, with 1 being a low intensity computation and 10 being a very high intensity computation. I will be using the c++ function "rand()" to generate random values of c_t.

Using this policy I will test three different thresholds: x_t, $x_t \in \{3, 5, 7\}$, against each other and then compare the results. The decision policy will also act upon the truth of four different assumptions and assemble an offloading decision based on that. These four assumptions are the availability of resources; the local CPU R_L, and MEC server R_m, where both values will have Boolean type. The availability at the time of the incoming task, a_t, of the resources will take one of four forms: $a_t = (R_L, R_m) \mid\mid (\bar{R}_L, R_m) \mid\mid (R_L, \bar{R}_m) \mid\mid (\bar{R}_L, \bar{R}_m)$. Once the availability of resources has been determined the offloading decision can be created, which will take this form: $\mathrm{D_t} = \{a_t, x_t, c_\mathrm{t}\}$. This array will inform the mobile device whether the task needs to be offloaded. For example, if $\mathrm{D_t} = \{(0,1), 4, 3\}$, then t will be executed locally because (0,1) indicates the local CPU is idle but the MEC server is busy, and $x_t > c_t$, so the task must not be offloaded as the intensity does not surpass the threshold. Conversely, if $\mathrm{D_t} = \{(1,1), 2, 3\}$, then this task will be offloaded because $x_t < c_t$, but the task must first wait for the resource to become available.

I will be using OMNeT++ , which is an extensible, modular, component based C++ simulation library and framework, primarily for building network simulators (OMNeT++, 2019). This will allow me to build and customise the network with the required functionality to accurately

Optimal Computation Offload Policies For Mobile Edge Computing

UP781439

MEng Computer Science

model a computation offloading scenario. The network contains a mobile device node which is connected through a router to the MEC server. MEC aims to unite the telecommunication and IT cloud services to provide the cloud-computing capabilities within radio access networks in the close vicinity of mobile users (Patel et al., 2014). Therefore the mobile device is only 2 hops from the server to replicate the intended implementation of MEC. Also there is 30ms delay in the simulation connection channel between the MEC server and device[0] which is inline with the ultra low latency provided by real 5G networks, such as that offered by (Verizon, 2020), the enabling technology of MEC.

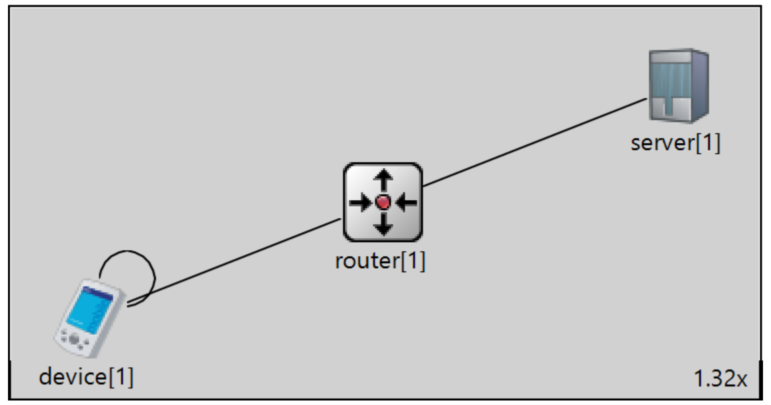

Fig 4. Network simulation layout

Fig. 4 shows the layout of the experiment that will be tested. If a computation is decided to be executed locally then it is sent around the loop connection visible on the "device[1]" node in fig 4., with a certain amount of delay to represent the time it would take to process the computation The same happens when an offloaded task reaches the server, the server waits a prescribed amount of time before responding to the device. The amount of delay is calculated using the benchmark speeds of the fastest current smart phone CPU, the Qualcomm Snapdragon 855 (Qualcomm, 2019), this CPU has a processing speed of 2.9Ghz and has 4 cores. There is not one fastest CPU for those aimed at being used in servers, because servers often have more than 1 CPU and these CPU's can often have as many as 12 cores, meaning the server can have 24 or more CPU cores and can concurrently deal with 48 threads, for example the Intel Xeon E5 2697 (Intel, 2019). So, with a simple calculation we can convert these numbers into an accurate ratio and derive delay values based on the intensity value.

$$\frac{(2.9\times10^9)*4}{2*((3.5\times10^9)*8)}\times 100 = 41.429\%,$$

$$100-41.429 = 58.571\%$$

This means that the computations that are executed locally must be delayed by their intensity value multiplied by 1.58571 and the ones which are offloaded to MEC are delayed only by their intensity value. This will not represent accurate values of the delay but it will represent an accurate ratio between the delay of different computation policies. For example, a computation executed internall with an intensity value of 2 will be delayed by, $2\times 1.58751 = 3.17502$s.

6. Results

The experiments that were conducted are as follows:

- Test 1, where $x_t = 3$
- Test 2, where $x_t = 5$
- Test 3, where $x_t = 7$

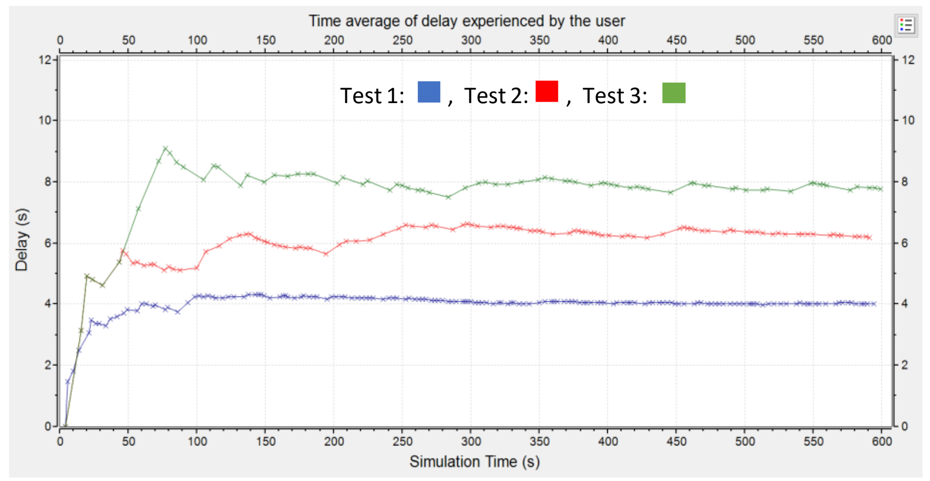

Fig 5. Average Delay Experienced by the user

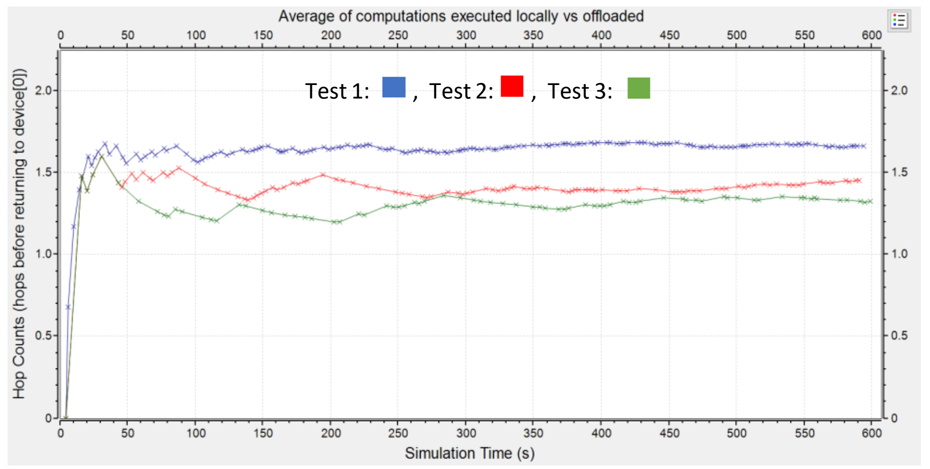

Fig 6. Ratio of computations executed locally vs offloaded

Fig 5. displays the average delay experienced by the user at every moment in time that a message arrives back to the mobile device either from the MEC server or local CPU. The average delay for test 1 across the entire simulation was 4.05s, for test 2 the final average delay was 6.09s and for test 3 the final average delay was 7.91s. This is inline with the related works that also found that by having a computation offloading which offloads a higher percentage of tasks to the MEC you can achieve less delay.

Fig 6. shows that the ratio between computation executed locally and at the MEC. As the average of each test gets closer to 2, there are a higher number of tasks being offloaded to the MEC.

7. Discussion

The delay of every computation that was completed was measured by first recording the simulation time at the moment that the computation is sent either to local CPU or the MEC, T_i, and then recording the simulation time at the moment the computation returns to the device, T_f. Therefore, the entire delay experienced by the user, including latency in the network, was calculated by, $Delay = T_i - T_f$. This was then converted this into the time average of the delay experienced over the whole simulation by plotting the change in the averages over time, which can be seen in Fig 5. This graph shows that as the threshold is lowered and you allow more, but less computationally intensive, tasks to go to the MEC the user will receive a better QoS with less delay. This is as was predicted by the research into related works that was conducted. You can also see from Fig 5. what a drastic impact on the differences in delay changing the threshold has, when the threshold = 7 the average delay is almost 100% higher than when the threshold = 6.

Fig 6. shows us the ratio between computations executed at the MEC or locally by the device. The closer to 2 on the y-axis that the lines get the more computations that are executed at the MEC server and the closer to 1 the more computations that are executed locally. This is because computations returning from the MEC server travel 2 hops, through the router, and tasks executed locally only take 1 hop to return to the device. As is evident in Fig 6. almost twice as many computations were offloaded in Test 1 than in Test 3 and this is reflected in Fig 5. by Test 1 having almost half as much experienced delay. This further backs up the theory that is crucial to offload as many computations as possible to utilize the extra power or the MEC Server.

There were , however, several issues with this experiment that have been identified. The network that was developed was simplistic and not truly representative of the architecture of MEC. For example, there are supposed to be multiple mobile devices in close proximity that create a virtual cloud at the edge of the network. However, in this experiment there was only one device that was in communication with the MEC server which made it hard to produce accurate results. Other problems caused by only having 1 mobile device is that it makes it difficult to simulate queueing times and network traffic because the server never has to deal with more than 1 incoming message at a time.

8. Conclusion

In conclusion I conducted a research experiment into finding an optimal computation offloading policy by attempting to simulate an offloading scenario with a MEC server and mobile device. The results of this experiment cannot determine an optimal policy for computation offloading. However, they can be used to show that offloading even smaller than average, i.e. intensity level < 5, computations for execution at an edge-located computing entity will result in being able to deliver smaller delay and lower network latency to the user.

The results cannot provide an optimal solution to computation offloading due to several factors including the need of a more accurate MEC architecture being implemented in the simulation. Another limitation of this experiment was the traffic generation in the simulation which was only producing one task at a time.

9. Future Work

For future work it would be interesting to have multiple mobile devices sending signals all at the same time and also to observe the effect and results when the distribution of computational intensity is not random but is either all very intense or all very simple to compute.

10. References

2020. Omnet++. OpenSim Ltd.

Bi, S. and Zhang, Y., 2020. *Computation Rate Maximization For Wireless Powered Mobile-Edge Computing With Binary Computation Offloading*. [online] Available at: <https://arxiv.org/pdf/1708.08810.pdf> [Accessed 15 March 2020].

Cheng, Y., Wang, D., Zhou, P. and Zhang, T., 2019. *A Survey Of Model Compression And Acceleration For Deep Neural Networks*. IEEE

ETSI, 2018. *MEC In 5G Networks*. ETSI White Paper No. 28. [online] Available at: <https://www.etsi.org/images/files/ETSIWhitePapers/etsi_wp28_mec_in_5G_FINAL.pdf> [Accessed 15 March 2020].

Ibm.com. 2020. *Cloud Computing*. [online] Available at: <https://www.ibm.com/cloud/learn/cloud-computing> [Accessed 11 March 2020].

IEEE Innovation at Work. 2020. *Edge Computing Resources - IEEE Innovation At Work*. [online] Available at: <https://innovationatwork.ieee.org/edge_computing/> [Accessed 11 March 2020].

Henze, M., Pennekamp, J., Hellmanns, D., Mühmer, E. and Ziegeldorf, J., 2017. *Cloudanalyzer: Uncovering The Cloud Usage Of Mobile Apps*.

Huang, L., Bi, S. and Zhang, Y., 2019. *Deep Reinforcement Learning For Online Computation Offloading In Wireless Powered Mobile-Edge Computing Networks*.

Kumar, K., Liu, J., Lu, Y. and Bhargava, B., 2012. *A Survey Of Computation Offloading For Mobile Systems*. [online] Springer Science+Business Media. Available at: <https://www.cs.purdue.edu/homes/bb/mobile-cloud-survey.pdf> [Accessed 9 March 2020].

Liu, J., Mao, Y., Zhang, J. and Letaief, K., 2016. *Delay-Optimal Computation Task Scheduling For Mobile-Edge Computing Systems*. The Hong Kong University of Science and Technology.

Maach, P. and Becvar, Z., 2017. *Mobile Edge Computing: A Survey On Architecture And Computation Offloading*.

Mao, Y., You, C., Zhang, J., Huang, K. and Letaief, K., 2017. *A survey on mobile edge computing: the communication perspective. IEEE Commun.*

Omnet++., 2019. *OMNeT++*.

Patel, M., Naughton, B., Chan, C., Sprecher, N., Abeta, S., Neal, A., et al., 2014. *Mobile-edge computing introductory technical white paper*.

Salmani, M., Sohraby, F., Davidson, T. and Yu, W., 2020. *Multiple Access Binary Computation Offloading Via Reinforcement Learning*. [online] IEEE. Available at: <https://www.comm.utoronto.ca/~weiyu/2019_Mahsa_CWIT.pdf> [Accessed 15 March 2020].

Satyanarayanan, M., 2017. *Edge Computing: Vision And Challenges*. [online] Carnegie Mellon University. Available at: <https://www.usenix.org/conference/hotcloud17/program/presentation> [Accessed 11 March 2020].

Satyanarayanan, M., 2017. *The Emergence Of Edge Computing*. Carnegie Mellon University.

Shah, S., 2017. *Mobile Edge Cloud: Opportunities And Challenges*. Hankuk University of Foreign Studies.

Shahzadi, S., Iqbal, M., Dagiuklas, T. and Qayyum, Z., 2020. *Multi-Access Edge Computing: Open Issues, Challenges And Future Perspective*.