An intelligent privacy-driven offloading algorithm for smart home applications

NIELS ROTMENSEN, University of Twente, The Netherlands

Abstract - Smart home devices are becoming increasingly widespread. Some of these newer smart home applications require more processing power than is available on-device, and require external computational resources. While a common option is using cloud computing power, processing locally on the edge is a better solution in terms of user privacy as data does not leave the local network. Previous hybrid approaches either lack privacy considerations or do not consider the future resource availability when making offloading decisions. In this paper, we propose a hybrid approach that minimizes the privacy risk of offloading tasks to remote servers, considering future resource availability. To reach our goal, we built upon an existing privacy-enhanced offloading algorithm, with the aim of improving its performance by predicting future state, and scheduling tasks accordingly. These improvements give more flexibility in scheduling, where a privacy-sensitive task can wait in queue instead of immediately offloading. The results show that while the system load is high, ~1-1.5% fewer high privacy-sensitive tasks are offloaded compared to a baseline without prediction, while CPU utilization remains nearly identical for both approaches.

Additional Key Words and Phrases: Resource allocation, Smart home, Task offloading

1 INTRODUCTION

Smart home devices are becoming more ubiquitous [3] and smarter due to advances in machine learning, which allows for more involved feature sets. However, these more advanced applications require more computing power. Since most of these devices use embedded hardware, they do not always have enough computational power to process these tasks on-device. Next to that, with more necessary computation, comes a higher power consumption, and since a lot of Internet of Things (IoT) devices are in a lower power environment, such as running on solar power or a battery, this is not desired.

A solution to this problem is offloading the processing towards another device. Here, a separate device will be responsible for processing the data generated by the IoT device. This device can operate within the local network (*Edge computing*) or be located in the cloud. The cloud possesses the unique property of facilitating advanced data processing capabilities, and allows hardware to scale as much as is in demand. While this does enable for more advanced software, it incurs high costs [1] and poses risks to user data privacy. On the other hand, offloading workload to an edge device, which is located in the local network, enhances user privacy by locally processing data. This approach ensures that data remains within the local network, avoiding transmission over the internet, external processing, and storage on third-party servers. Moreover, an edge device is a one-time investment, instead of a cloud device which is rented. However, the limited computational resources of edge devices lead to scalability issues, often causing delayed processing or dropped workloads when demand exceeds capacity.

Both solutions discussed above present distinct advantages and limitations, with their suitability varying depending on the specific context. Therefore, utilizing a hybrid approach where both local and cloud resources are utilized is promising. Thereby, all tasks are run on the local hardware and in the case of insufficient local resources, tasks are offloaded to the cloud.

However, this approach can be further refined through enhanced scheduling techniques. Building upon previous work of Rezaei et al. [6], we propose an adjusted solution of scheduling workloads. While the previous solution schedule workloads only based on the current available resources, we also consider the availability of resources in the future while making decisions. Therefore, we define the following terms for scheduling workload: *local execution*, which refers to a task being executed on the edge device, *remote execution*, which refers to a task being offloaded to the cloud, and *later execution*, which refers to a task that will not be executed this timeslot. The feasibility of scheduling a task for *later execution* depends on specific criteria. A task can be deferred if its deadline permits and if there is a reasonable prediction of available computational resources in the future. If either of these conditions is not met, the task will not be scheduled for *later execution*.

To achieve our proposed improvement, we formulate the following research questions:

- **Research question 1:** What are the benefits of predicting the future state of the edge device on overall system performance?
- **Research question 2:** What improvements to user privacy can be made with the benefits of **RQ1**?

2 RELATED WORK

In this section, we review some of the previous work that studied workload scheduling and computational offloading for smart homes.

In 2016, Vakilinia et al. [7], present a method for running smart home applications in the cloud inside virtual machines (VMs). The paper talks about the minimal amount of resources necessary to satisfy the Quality of Service (QoS) constraints for these applications to run satisfactorily.

In 2020, Liu et al. [4] took a different approach to hosting smart home applications, namely using edge devices. They proposed a resource allocation algorithm that prioritizes workloads with regard to user preference. This was done by applying an economic model to the scenario, where edge devices were seen as providers in a market, and users (and their smart home applications) as consumers. With

TScIT 42, January 31, 2025, Enschede, The Netherlands

that, they were able to write a pricing-based resource allocation algorithm that converges within a reasonable time.

Work by Dong et al. [2] focused on smart home applications running machine learning workloads. They introduced edge devices into the local network, to allow for local computation. Since machine learning workloads are usually heavy, they used a technique called federated learning to offload parts of the workload in to the cloud. This improves user privacy, since less user data leaves the local network.

Later, Rezaei et al. [6] presented a resource allocation and offloading algorithm for federated learning (FL) applications in smart homes. Their algorithm decides whether to execute FL tasks locally on edge devices, in the cloud, or through a hybrid of both, while minimizing the privacy leakage. However, their algorithm only has knowledge of the current resource availability, and cannot predict the future state of resources to postpone tasks.

3 METHODOLOGY

This section will cover the methods used to conduct the research.

We build upon the previous simulator (RASH) which allocated local resources among IoT tasks and decides where each task should be executed, either locally or remotely [5]. While making offloading decisions, RASH considers available local resources, tasks time budget, tasks computational requirements and the privacy-sensitivity level of the tasks. When local resources are insufficient for timely completion of the tasks, RASH offloads some of the tasks, while prioritizing the lower sensitive tasks for offloading. However, RASH only considers the current available resources for its decisions. We expand the decision-making algorithm in RASH so it also considers the availability of local resources in the future when making decisions.

We define our system model as the following: A system with 1 or more smart home devices, all generating tasks that require computational resources up until the set system load. These tasks are then scheduled by the offloading algorithm, which is RASH as our baseline, or our proposed algorithm.

The simulator makes use of an internal data structure for each task, each holding a lot of metadata which will be used throughout the paper. When a task is generated, it is defined by three characteristics:

- **Time budget**: The amount of time that the task should be finished in. A high time budget allows for a long wait.
- **Required computation**: The amount of computation resources this task needs.
- **Privacy score**: Each task comes with a privacy score, rated 2-9. A higher score means that it is more important to process this locally.

We introduce an additional step on top of RASH's decisions to account for the future availability of local resources in the decision-making process. RASH operates in timeslots, where every timeslot the decision-making algorithm runs. After each decision, we refine it based on our predictions. If a task is initially set to be offloaded to the cloud, the prediction algorithm runs to assess future local processing availability. If sufficient local resources are expected within the task's time budget, the task is marked for *postponement* instead of immediate cloud execution. In the next timeslot, this decision is reevaluated, and it might be possible that cloud execution is the chosen solution, or it is again decided that postponing is the best option. An explanation of how this decision is made is provided below, broken down into multiple code snippets.

First, we predict the number of time slots required for completing each running task. This is calculated in **Algorithm 1** based on the assigned computational resources for that timeslot, denoted by *task.assigned_rsc*, and the remaining progress, denoted by *task.rem_rsc*. With this information, we estimate the amount of timeslots this task needs to complete execution.

Algorithm 1 Predicting end timeslot

```
1: function PREDICT END TIMESLOT(task)
2:     for each task do
3:         cur_rsc ← task.rem_rsc − task.assigned_rsc
4:         timeslots ← task.rem_rsc / cur_rsc
5:     end for
6: end function
```

The next step is to examine the tasks that have not yet started and are scheduled for remote execution. First, we calculate the *threshold* for each of these tasks. A threshold is the latest possible timeslot where it is possible to start execution remotely, without dropping the task due to exceeding the time budget. When this threshold is reached, the task will not enter the postponing cycle and will be executed.

Calculating the threshold is done in **Algorithm 2**. Here we introduce a new variable, *risk*. This variable is predefined before runtime and influences the threshold by determining the proportion of computation resources a task is expected to receive in each timeslot. If, for example, *risk* is set to 0.10, the threshold is calculated on the idea that this task will, on average, use 10% of the available computational resources until it is finished. Setting the *risk* higher will result in a lower threshold (i.e. the task will be kept in the postponing queue for potentially longer), but setting it too high can result in the task being dropped due to a missed deadline, in the case the assigned computational resources were lower than the *risk* used for calculating the threshold.

Calculating this threshold is trivial, where first the predicted amount of timeslots *p_timeslots* is calculated using the *risk* parameter explained above, Then, the latest possible timeslot for postponement is calculated by adding *p_timeslots* to the current timeslot (*cur_time*). Beyond this threshold, postponement is no longer feasible, and the task must be executed to meet its deadline.

It should be noted that this threshold is only calculated once, the

first time the task is seen by the algorithm. To do this, the threshold is defaulted to −1, to signify that no threshold has been set yet.

Algorithm 2 Calculating task threshold

```
Ensure: task.threshold ≠ −1
    function CALCULATE THRESHOLD(task, risk, cur_time)
2:      p_timeslots ← ceil(task.data_for_processing / risk)
        task.threshold ← cur_time + p_timeslots
4:  end function
```

After running both **Algorithm 1** and **Algorithm 2**, we can now decide on postponing certain tasks. To do this, we loop through all running tasks, and check if it is possible to schedule the to-be-planned task afterwards (line 6). If this is possible, we decide on postponing this task, since we assume that it is going to run in the future. If this is not possible, the task is flagged as impossible to postpone, and will be offloaded on the next iteration. Postponing the task will kick this process off again next iteration, and it might be possible that the predictions made have changed.

Algorithm 3 Deciding if task is able to be scheduled in the future

```
    function ABLE TO SCHEDULE(tasks, task, time_slot)
        for all t ∈ tasks do
3:          if not other_task.decided then
                continue
            end if
6:          end ← task.timeslots_left+other_task.timeslots_left
            if end ≤ task.end_timeslot then
                return true
9:          end if
            return false
        end for
12: end function
```

4 ANALYSIS

In this section, we will discuss how we analyse the performance of the algorithm, and introduce some metrics to measure against. For this, we consider RASH as the baseline to evaluate the performance of our proposed algorithm.

To facilitate developing and testing our methods, we build our proposed algorithms as an extension of RASH, which was explained in more details above. For analysis of the results, we use a python script that parses the logging files, and creates graphs using the `seaborn` library.

We evaluate the performance of our algorithm using the following metrics:

- **CPU usage** This metric shows the proportion of local edge device used during the execution of the simulator.
- **Privacy sensitivity** Every task has a certain privacy score, where higher is more important. When allocating resources, tasks with a higher privacy score are prioritized. This metric will be based on the sum of the total privacy score of all tasks running locally, and all tasks running externally. Our algorithm aims to improve the privacy score for local tasks, by assigning tasks to run in the future, instead of offloading immediately.
- **Task satisfaction** This metric measures the fraction of tasks completed within their deadline.

5 RESULTS

In this section, we will discuss performance of this algorithm, and compare it in different scenarios. Every scenario ran in the simulator for 1000 timeslots, with 50 iterations each. Next to that, we test on system loads of 70%, 90%, 110% and 130%, with the *risk* parameter set to 0.1, 0.2 and 0.3

The *risk* parameters were chosen by observing RASH assign computational resources during runtime. We found that, on average, it assigned a value of ~0.2-0.25. Since *risk* is closely related to the assigned computational resources, we defined *risk* based on this relationship, assigning it the values of 0.1, 0.2 and 0.3.

5.1 CPU usage

The first metric we test is the CPU usage. We ran the simulator for 70%, 90%, 110% and 130% (**Fig. 1**) system load. We compare the CPU usage of our proposed method and RASH in **Fig. 1**. Next, these tests were repeated at different settings for the *risk* parameter. These were set to 0.1, 0.2 and 0.3.

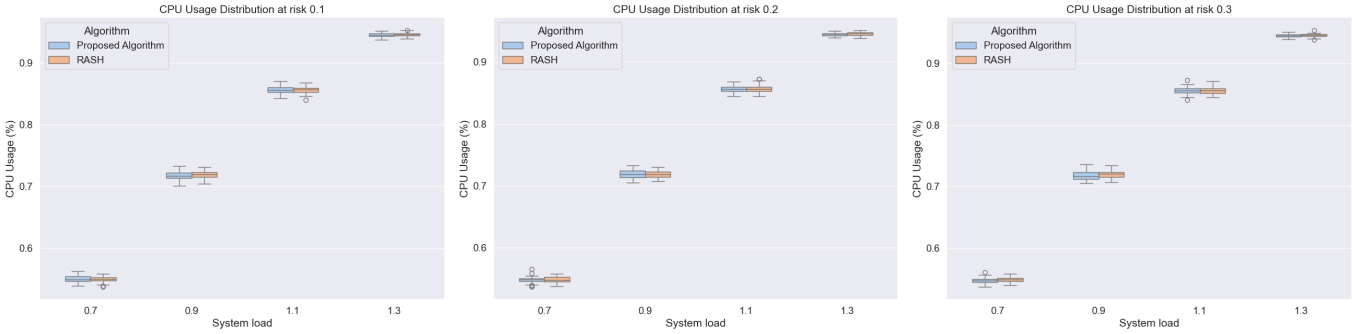
From the results, several observations can be made. First, an increase in system load correlates with higher CPU usage, rising from 55% CPU usage at 70% system load, to 95% CPU usage at 130% system load, across both methods. Next, it can be observed that both methods yield approximately the same CPU usage. This indicates that while our proposed method does not enhance system performance, it also does not negatively impact it due to the extra processing required. From these findings, we can directly address **RQ1**, concluding that while no performance benefits are observed, there are also no degradations in system performance.

5.2 Privacy score

The next test, is the privacy score of each iteration. To recall, every task has a certain privacy score attached, ranging from 2-9. A higher score means it has priority over the local hardware. For this section, we computed the percentage of tasks executed locally and on the cloud for each privacy score, categorized by method. Privacy-sensitive tasks should be prioritized for local execution, while less privacy-sensitive tasks should run externally, given there is no room locally.

The results can be found in **Fig. 2** for risk 0.1, **Fig. 3** for risk 0.2, and **Fig. 4** for risk 0.3.

For loads under 100%, almost all tasks ran locally. This is expected behaviour, as in these cases it is rare to have the system overloaded enough that offloading should occur.



(a) Risk: 0.1 (b) Risk: 0.2 (c) Risk: 0.3

Fig. 1. CPU usage with and without scheduling at 70%, 90%, 110% and 130% load

At 110% and 130% system load, we see that tasks start being offloaded, as the local resources are now not enough to fully satisfy the resource demand. First, it can be observed that, for both methods, tasks with a lower privacy-score are more likely to be executed in the cloud. Of course, not all tasks with a high privacy score can be executed locally, as that depends on the current computational resources available. Next, we can observe the differences between the baseline and our proposed method. When *risk* is set to 0.3, we can note that overall, tasks with a higher privacy score run around 1-2% more locally with our proposed method. When *risk* is set to a lower value, similar observations can be made. Curiously enough, this pattern is inverted at a system load of 130% with *risk* set to 0.2. This leads us to address **RQ2**, where it can be noted that, while the improvements are minimal, there is some degree of enhancement. These improvements could likely be further optimized through fine-tuning the algorithm and the *risk* parameter.

5.3 Task satisfaction

Across all tests, task satisfaction nears 100%. Overdue tasks are exceedingly uncommon, and across all iterations of every task, only 20 tasks were dropped. To bring that into perspective, every test consists of 50 iterations * 4 system loads * two methods * ~250 tasks, which would make all overdue tasks insignificant.

6 CONCLUSION

In this paper, we try to improve performance of a IoT workload offloading algorithm by predicting the future state of the local edge device, and scheduling accordingly. The main aim of this improvement is to run more privacy-sensitive workloads locally. We try to implement this by first running a regular offloading algorithm and optimizing its results further. This result will consist of tasks scheduled to run locally or externally. Next, a schedule is compiled of all running tasks, of which the decision is made for all externally-marked tasks to wait until resources are available, or to run a task externally.

Our result should that our proposed method improved local execution of privacy-sensitive tasks by ~1-1.5%, but this difference is not statistically significant. While the proposed method shows potential for improved privacy-aware scheduling, the observed gains were marginal, indicating a need for further refinement. Future work could focus on refining the algorithm and parameters used, exploring alternative approaches to future prediction, or incorporating additional parameters to enhance its performance and applicability.

REFERENCES

[1] Lubna Luxmi Dhirani, Thomas Newe, Elfed Lewis, and Shahzad Nizamani. 2017. Cloud computing and Internet of Things fusion: Cost issues. In *2017 Eleventh International Conference on Sensing Technology (ICST)*. 1–6. https://doi.org/10.1109/ICSensT.2017.8304426

[2] Yueyu Dong, Fei Dai, and Mingming Qin. 2022. A Privacy-Preserving Deep Learning Scheme for Edge-enhanced Smart Homes. In *2022 IEEE Intl Conf on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Technology Congress (DASC/PiCom/CBDCom/CyberSciTech)*. 1–6. https://doi.org/10.1109/DASC/PiCom/CBDCom/Cy55231.2022.9928002

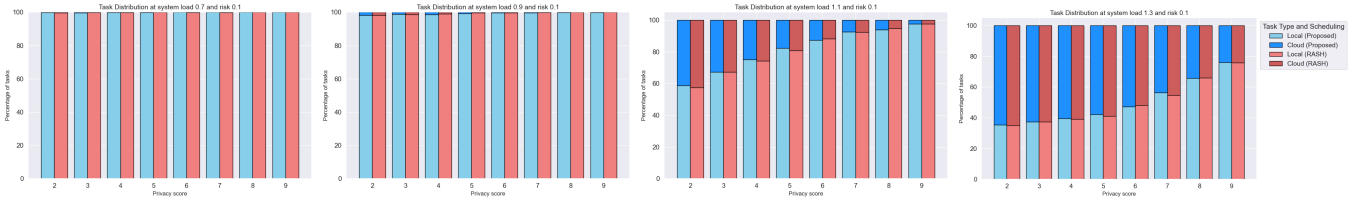
[3] Sara Gøthesen, Moutaz Haddara, and Karippur Nanda Kumar. 2023. Empowering homes with intelligence: An investigation of smart home technology adoption and usage. *Internet of Things* 24 (12 2023), 100944. https://doi.org/10.1016/J.IOT.2023.100944

[4] Huan Liu, Shiyong Li, and Wei Sun. 2020. Resource allocation for edge computing without using cloud center in smart home environment: A pricing approach. *Sensors (Switzerland)* 20 (2020). Issue 22. https://doi.org/10.3390/s20226545

[5] Tina Rezaei. 2024. "RASH: Resource Allocation for Smart Homes Considering the Privacy Sensitivity of IoT Applications". Retrieved November 22, 2024 from https://github.com/Tina-Rezaei/RASH

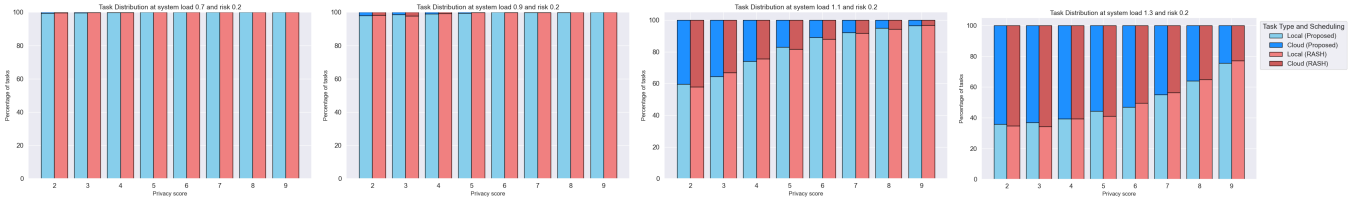
[6] Tina Rezaei, Suzan Bayhan, Andrea Continella, and Roland Van Rijswijk-Deij. 2024. ERAFL: Efficient Resource Allocation for Federated Learning Training in Smart Homes. In *NOMS 2024-2024 IEEE Network Operations and Management Symposium*. 1–5. https://doi.org/10.1109/NOMS59830.2024.10575706

[7] Shahin Vakilinia, Mohamed Cheriet, and Jananjoy Rajkumar. 2016. Dynamic resource allocation of smart home workloads in the cloud. 367–370. https://doi.org/10.1109/CNSM.2016.7818449



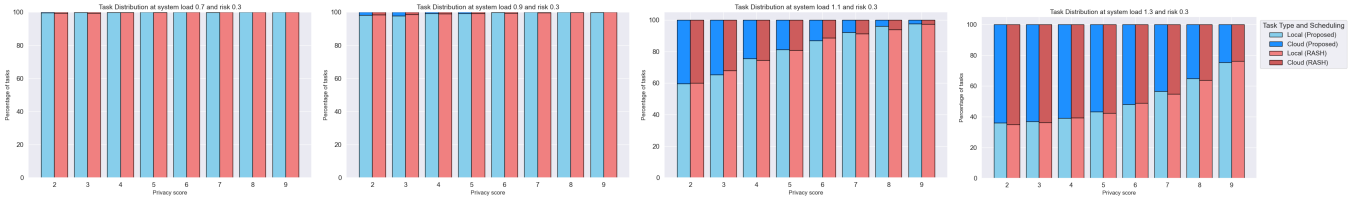
(a) Load: 70% (b) Load: 90% (c) Load: 110% (d) Load: 130%

Fig. 2. Percentage of tasks ran on cloud/local, at risk 0.1



(a) Load: 70% (b) Load: 90% (c) Load: 110% (d) Load: 130%

Fig. 3. Percentage of tasks ran on cloud/local, at risk 0.2



(a) Load: 70% (b) Load: 90% (c) Load: 110% (d) Load: 130%

Fig. 4. Percentage of tasks ran on cloud/local, at risk 0.3