Recent Developments and Programming Models of Mobile Device-to-Device Computation Offloading

Kert Tali
kert.tali@ut.ee
Institute of computer science, University of Tartu
Tartu, Estonia

Jakob Mass (supervisor)
jakob.mass@ut.ee
Institute of computer science, University of Tartu
Tartu, Estonia

ABSTRACT

The method of choice in today’s world to relieve resource-constrained mobile devices of computational tasks is to delegate them to the cloud. This notion is known as Mobile Cloud Computing. Rising number of mobile devices and cloud services may call for a change in this direction as the transported data volumes exceed network limitations and hinder the applications dependent on it. In addition to Mobile Edge and Fog Computing, an overlooked candidate for handling our computation requirements could be other mobile devices in the area which are underutilized. Although a seemingly great solution for bringing cloud-like services even closer to the end-user, research in this topic has been ongoing ever since smart-phones became commonplace and has identified a multitude of accompanying issues yet to be solved. This report outlines one area of concern for the future of such a computation offloading model – the frameworks and programming models that would be compatible with it. Secondly, it gives an overview of recent efforts made towards solving the issues and speculating what it takes to reach a real implementation.

1 INTRODUCTION

FemtoCloud, as coined by Habak et al. [14], is the vision of a future in which mobile devices opportunistically share work between each other. Under the assumption, that a mobile application or device is bootstrapped with necessary means to delegate parts of their computation to be completed elsewhere, there is potential for increased performance and battery savings. This delegation of computation, commonly known as computation offloading, is employed if there is a target system with abundant resources willing to accept the tasks, such as a cloud. In a FemtoCloud, however, the target is rather unconventionally comprised of neighbouring mobile devices, clustered together to offer a cloud-like service for computation offloading. It relies on the fact, that mobile devices have become ever more powerful to accommodate the requirements of bleeding edge applications, and consequently have great amounts of spare resources when idle.

For an offloading model to be viable for a given task, we can think of the limiting factors imposed by different setups

- **Latency** – Target systems have to be sourced from nearby resources for the round-trip to be quick enough for delay sensitive tasks. A far away datacenter may be unsuitable due to this restriction.
- **Bandwidth** – Tasks may require large sizes of accompanying input data and returned results, with large scale deployments contributing to network congestion.

It is easy to argue that the three points are deeply rooted in cloud and even fog computing models. Positioning nodes on the edge would solve the first two issues, but comes with the most significant infrastructure cost. The affair brings us back to our motivational example – the FemtoCloud, which lends itself to alleviate the bottlenecks and issues of more traditional computation offloading models. Seemingly so – offloading between devices which are all in nearby proximity remedies the network concerns, and the absence of a dedicated offloading target but rather an opportunistic cluster could potentially eliminate most infrastructure cost.

These benefits have been known to the edge computing research community for a while and have motivated numerous works for identifying and proposing solutions to the accompanying problems preventing the idea from being brought to life. Some domain specific problems tackle the social aspect of offering incentive to holders of mobile devices which form the compute cluster [9] or how to schedule and assign tasks between workers in the unstable cluster in a way that would yield optimal resource usage [15]. The instability of the cluster is characterised by the churn of its workers, meaning that if a device leaves the vicinity during calculation, the task has to be restarted. An apparent but more general issue, that applies to any computation offloading scenario, is task decomposition – splitting tasks into subtasks that can be computed externally. In mobile application development, this can be boiled down to a choice between different programming models that could be used in implementing an interface for mobile computation offloading on the client’s side.

In this report, the latter idea is pursued further to seek out an answer to the first research question: *Which programming models could be used for mobile device-to-device computation offloading?* This question is motivated by the feasibility standpoint of such a system, namely from the view of the developer, who would need to use an offloading framework in their application. The second research question relates to the state-of-the-art of opportunistic offloading between mobile devices, asking: *What are the recent advancements in mobile device-to-device computation offloading?* Most of the defining ideas regarding this topic have been published more than 4 years ago. Mapping the fresh directions to see where the area is evolving is the main motivation behind that question.

This report is structured as follows. Section 2 outlines the previous work relevant to the research questions. In Section 3, programming models for mobile computation offloading are compared and discussed. Section 4 gives an overview of the relevancy of the topic today by detailing the latest developments. Based on insight
from the preceding survey, the biggest issues and obstacles are reiterated, and some concerns left unhandled in research, are raised in section 5. Work is then concluded in Section 6.

2 RELATED WORK

One of the first justifications for offloading computation from and to mobile devices was published in 2012 by Shi et al. [22]. The motivator of at the time was the chance to speed up computing while saving battery, doing so by opportunistically seeking out surrogates for computation in an unstructured intermittent cloud of mobile devices. Serendipity, as it was called, pushed the edge research community to investigate the idea further in the coming years.

Before such offloading models, cloud, fog, and edge offloading was the state-of-the-art for improving execution time of mobile applications. Several efforts were made for accomplishing code offload with elaborate programming models supporting it [4, 6, 13, 16, 17, 19, 20, 26]. Frameworks that specify an API and target offloading to mobile devices are less plentiful, but include P3-mobile [24], Honeybee [7], and PMDC [25]. It does still help to analyse cloud specific models to assess whether they would be adaptable to our model – this it detailed further in section 3.

Research of opportunistic computation offloading is well summarized in a 2018 survey by Xu et al. [27]. They agree, that the vision of offloading computation from mobile devices to a cluster comprised of other mobile devices has seen several takes over the last decade. The unifying idea is to eliminate the dependence on centralized or edge clouds in the more traditional mobile cloud computing model, where computation may be offloaded to remote servers. Benefits of this include better latency, less infrastructure cost, and utilizing otherwise idle but capable mobile devices. Proposed solutions often tackle some concrete problem associated with this approach, be it the social aspect of offering incentives to participating devices, mitigating the effect of churn in the unstable system, or optimizing battery usage of the devices. While task partitioning and relevant programming models are not specific to this topic, they are often explored in these works.

Flores et al. [9] experiment with hybrid offloading, where the place to offload can be decided between a remote cloud, nearby cloudlet, or other mobile devices. Arguably the most important question deciding the success of such visions of mobile clusters is a viable incentive mechanism for participants to even lend their mobile device’s resources for communal use. In their work, they discuss a sustainable and general system, combined of both credit and reputation. Credit is a resource that is transferred from the offloading client to the offloadee peer. It is not specified whether credit should have a monetary value or can be bought – its sole purpose is to discourage leeching. On the other hand, reputation is a value accumulated to a user for positive feedback after computation. Users with better reputation value are more visible in a sense that they are better potential candidates for offloading work to.

3 PROGRAMMING MODELS

For a task to be offloadable, it has to be able to be represented as a self-contained unit of work with inputs and outputs. The API for communicating these tasks may vary in flexibility and application specificity. For example, we might provision an offloading target running an object detection algorithm on demand, which takes an image as input and outputs the positions of detected objects. While referred to as offloading, such application specific approaches rather fall into the field of mobile cloud computing. The many articles focusing on task scheduling and system topology generally do not focus on the underlying framework that defines and provides the tasks, due to the considerations there already presenting a problem difficult enough. Nevertheless, it is clear that in a real scenario, a scheduling algorithm and specific framework go hand-in-hand in creating the ecosystem – prompting also a search for the ideal and agreed on programming model that would support it.

Ideally, a technique which enables communication of work and inputs over a transparent low-level or generic standard interface is needed. Moreover, it should not hinder the developer in the process of adding offloading support to their applications. In this section, we analyse different takes on offloading framework implementations that are suitable for mobile device-to-device systems like the Femto-Cloud. Succinct overview of the discussed technologies is given in table 1 Analysis is split into subsections – first looking at the more concrete APIs that have been proposed, and secondly exploring completely transparent solutions implemented in middleware. The bulk of this section lends from the compiled list of technologies by Xu et al. [27] for cloud and cloudlet based offloading, but here the focus is on evaluating the potential of those technologies in a device-to-device offloading case.

3.1 Offloading APIs

The simplest method to construct a computation delegation framework would be to leave task partitioning to be a responsibility of the application’s developer. A developer would manually identify any subroutines consuming significant computational resources and notify the corresponding offloading service [6]. This may mean simple changes, like annotating suitable functions in code, or be much more hands-on, requiring explicitly defining the task partitioning logic. The benefits of such APIs include fine-tuned task granularity and size of the inputs and results, as they can be optimized for the specific task, resulting in less traffic and energy savings. A down-side is of course the development cost, and the resulting risk of never reaching wide-spread adoption.

One way to define parts of offloadable code would be to define interfaces connecting the application to the compute-intensive services. Cuckoo [16] is an example of a framework employing such programming model. With the developer defining their services in Android Interface Definition Language (AIDL), Cuckoo generates an implementation beside it that can be executed in a remote environment. At runtime, depending on the conditions, the decision is made whether to use the local service or use a remote instance instead.

A very similar take exists on the AIOLOS framework [26] for Android systems, dependent on the OSGi rather than AIDL. Here, instead of defining interfaces manually, AIOLOS aims to improve developer experience by providing IDE (Integrated Development Environment) plugins for automatic generation of the module code containing the logic to offload. Specifically, the OSGi module would

\[\text{https://www.osgi.org/resources/what-is-osgi/}\]
Recent Developments and Programming Models of Mobile Device-to-Device Computation Offloading

<table>
<thead>
<tr>
<th>Programming Model</th>
<th>Method</th>
<th>Implementation Effort</th>
<th>Device-to-Device Compatible</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cuckoo [16]</td>
<td>interfaces</td>
<td>RPC</td>
<td>Medium</td>
</tr>
<tr>
<td>AIOLOS [26]</td>
<td>interfaces</td>
<td>RPC</td>
<td>Medium</td>
</tr>
<tr>
<td>MAUI [6]</td>
<td>method annotation</td>
<td>RPC</td>
<td>Low</td>
</tr>
<tr>
<td>ULOOF [19]</td>
<td>method annotation</td>
<td>RPC</td>
<td>Low</td>
</tr>
<tr>
<td>ThinkAir [17]</td>
<td>method annotation</td>
<td>RPC</td>
<td>Low</td>
</tr>
<tr>
<td>P3-mobile [24]</td>
<td>parallel</td>
<td>proprietary</td>
<td>Very high</td>
</tr>
<tr>
<td>PMDC [25]</td>
<td>microservices</td>
<td>HTTP</td>
<td>High</td>
</tr>
<tr>
<td>CloneCloud [4]</td>
<td>-</td>
<td>thread migration</td>
<td>Automatic</td>
</tr>
<tr>
<td>COMET [13]</td>
<td>-</td>
<td>distributed shared memory</td>
<td>Automatic</td>
</tr>
</tbody>
</table>

be published to a public registry to be downloaded and run by the offloading target on demand. This leaves the developer only having to add code annotations to candidate offloadable methods.

Bridging the gap between manual and automatic program partitioning and state migration, exist frameworks like MAUI [6], ULOOF [19] and ThinkAir [17]. All work similarly, making an effort to identify methods in an application tagged with a [@remotable], @OffloadCandidate or @Remote annotation respectively. The MAUI framework, implemented for .NET and targeted at the Windows Mobile OS, makes effort to identify methods with static and runtime analysis which have potential to optimize battery usage when offloaded (e.g. transferable state in relation to required CPU cycles is small). [@remotable] methods are constrained to such that do not interact with the user interface or external devices (like the device’s sensors or GPS), requiring the developer to just do the initial filtering.

With MAUI, Cuckoo, and ThinkAir, the server environment must contain the same application from which the offload task originates. This is due to their dependence on Remote Procedure Calls, which work by having the logic already hosted at the other end waiting for invocation. In a mobile device-to-device scenario, it might be the case with only popular applications to find surrogates in the proximity with the respective service installed. ULOOF is more problematic to integrate with this idea, as workers have to obtain a separate library which has the Java Remote Method Invocation skeleton and business logic. The question of when or how these libraries could make their way to the mobile device could be done dynamically on-demand, however it would require the offloaders to fetch the binaries from a remote library or be sent with the task – both increasing the network traffic overhead.

Song and Tilevich propose an ecosystem of offloading microservices, specific to be deployed in mobile clusters [25]. In their programming model, developers are expected to build their applications around reusable logic blocks from the ground up. By either creating the blocks themselves or leasing from the work of others, all microservices get uploaded into a central registry to be downloaded by the workers on demand and executed with some input provided by the offloading device. The created prototype works within a local wireless network on a cluster of Android devices, relying on the gateway to do device capability assessment and scheduling. On one hand, requiring major code overhaul for existing applications, and a dependence on a central service may discourage adoption. On the other hand, for general tasks that are small in granularity (like matrix multiplication), it could be considered. Realistically, it might be too difficult to host such instances on mobile devices to be practical due to the sheer amount of different applications and size requirements.

Splitting a parallelizable task between nearby mobile devices is another way to achieve computation offloading, and is an alternative to plain method offloading. While several proposed systems consider such task splitting [20, 22], they consider cases in which the task is embarrassingly parallel (i.e. meaning that its subtasks are all independent of each other) in order to do the splitting decision automatically. A concrete API for manual definition of parallel tasks is P3-mobile [24]. The API exposed to the developer for defining parallelizable jobs requires implementing methods for the setup of I/O and data structures, algorithm itself, task splitting, restarting, and completing.

### 3.2 Middleware Assisted Offloading

Contrary to an API for manual task partitioning and offloading, a middleware solution that detects and offloads potential tasks might be a more desirable and convenient approach. The middleware might profile the runtime characteristics of an application and translate identified subroutines that could be remotely executed to a state (inputs) and instructions (code). The following approaches require no source code or binary modifications by application developers to support – rather they use mechanisms to detect and offload computation transparently.

Straying further from concrete tasks, one could potentially delegate computations with thread migration. Thread migration and merging is one way to transparently move an application’s state and required computations to complete them externally, doing so in the granularity of a single worker-thread. The CloneCloud [4] framework achieves just that by first cloning the device to an application-layer virtual machine (DalvikVM in the case of Android) at the offloading target, providing an environment to trigger computation at. The serving node is meant to be deployed in a cloud environment and is hardly adaptable to device-to-device – cloning a device to another which has similar resources is unimaginable due to required traffic and space requirements. Furthermore, as we are looking at an ever changing cluster in which we do not have long-lasting connections, the constant environment set-up might pose as significant overhead.

Closely related to CloneCloud, but arguably an improved approach would be to implement distributed shared memory work.
sharing. COMET is an extension to the Android DalvikVM with support for distributed shared memory and VM synchronization based offloading, allowing arbitrary threads to be offloaded to other devices [13]. The high-level architecture can be seen on figure 1. Authors present a working prototype with the client being able to offload computations of various existing applications from the Google Play Store to a local server over Wi-Fi or cellular networks and report significant speedups.

Originally, both COMET and CloneCloud target the mobile cloud computing model, where computations are offloaded to a more capable dedicated server. In both works, it is noted that the transferred state can sometimes be unnecessarily large on the account of data that is not used in the computation, but sent unknowingly. With COMET however, as the runtime environment is already present in other Android devices and application specific binaries do not have to be present, it could be viable even in the mobile cluster scenario. Therefore, ideally, in terms of developer experience and viability in mobile workers, it would be the best choice. Further work also has to be conducted to assess possible deployment of their worker service on mobile devices.

4 RECENT DEVELOPMENTS IN OFFLOADING TO MOBILE DEVICES

Mobile device-to-device computation offloading has been a rather prominent research topic over the last decade, ever since the widespread adoption of smartphones. Recent work and novel proposals related to this problem have slowed down in recent years. In a recent overview, the authors of the FemtoCloud [14] give a comparison between Mobile Cloud Computing and Mobile Device Clouds for computation offloading and map research challenges [12]. The challenges include scheduling, incentive, privacy, code migration (i.e. what to offload?), and scaling. The following section summarizes the main research directions and results of papers published from 2019 to 2021, which are oriented towards solving these concerns.

4.1 Scheduling and Task Assignment

The FemtoCloud authors have since been actively publishing follow-up articles addressing the scheduling and task assignment algorithm. In [10], the FemtoCloud ecosystem and scheduling algorithm is extended to provide service in a wider area, pooling computing resources from multiple FemtoClouds in vicinity, which is reminiscent of crowdsourcing. In their latest effort, the original scheduling heuristic, which maximized computation throughput, is adapted to a multi-objective algorithm – namely accounting for two modes: latency-minimization and energy efficiency [11].

There has been input from other authors to better the FemtoCloud. Anglano et al. first proposed the WQR-UD (Work Queue with Replication and User Driven Checkpoint) algorithm for specifically for scheduling completely independent parallel tasks [2]. The algorithm was later relabeled as UDFS (User-Driven FemtoCloud Scheduler) and published along with more in-depth explanation and evaluation [1]. With regard to the volatile churn environment of a mobile device cloud, the proposed algorithm implemented checkpointing and replication of tasks, which showed improvement in task completion rates over the original FemtoCloud algorithm.

Fernando et al. propose a work stealing model named Honeybee to overcome the node selection problem in offloading to mobile devices [8]. Its main idea rooted in the fact, that the devices themselves should be most aware of their up-to-date resources and presence time, meaning that they can compute tasks that are broadcasted in the vicinity. This allows eliminating any coordinating nodes in the system and makes way for pure peer-to-peer communication – in the case of Honeybee, Wi-Fi Direct for nodes to exchange tasks.

A set of code offloading schemes and task assignment algorithms was designed for Apple devices by Lin et al. [18] under the name Circa. Circa uses iBeacons discover nearby devices to form a cluster.

4.2 Incentive Mechanisms

A feasible incentive mechanism is required for users to offer their resources to foreign devices, mainly to compensate their own costs on the account of device battery, network charges (in the case of mobile data), and possibly inconvenience if the in-progress computation slows down other applications. Saha et al. have set out to explore a bidding mechanism for worker devices to acquire tasks from the local cluster, but combined with a quality-of-service metric. Provided that more capable devices bid higher amounts, respective to the quality-of-service (e.g. how quickly they are able to complete the computation) they can provide to the client, they are rewarded on their successful task completions. If they were to lie in their bid, the reimbursement would be reduced. This motivates more high-powered workers to participate in the system, while allowing clients budget their expenses if they are okay with less quality-of-service. An auction environment does require a local intermediary node to decide the result of the bidding.

An analogous auction based quality-of-service incentive mechanism was published by Roy et al. [21] around the same time. Rather than penalizing workers for misplaced bids, it builds their reputation whenever they provide the service they promise. Furthermore, there is no need for a separate auctioning node since the clients hold the auctions and choose winners themselves. Reputation is calculated in a decentralised manner using federated learning, in which clients rely on their own, and co-located clients’ experiences with a specific worker.
4.3 Energy Management

Energy considerations for battery powered devices are crucial when considering joining it to a compute cluster. Recent advances in this topic have been scarce, however. Balasubramanian et al. design a framework for energy management within a mobile device cloud, based on reinforcement learning [3]. They propose a decision control mechanism for devices offering to do the calculations that keeps the whole cluster running within the bounds of their energy usage policy.

4.4 Offloading to Hybrid Clouds

Offloading to hybrid clouds (i.e. adaptively choosing between cloud, fog, and device-to-device for offloading needs) is also gaining in popularity. Jay, a platform recently proposed by Silva et al. [23], connects multiple offloading targets, with devices having a better chance of there always being an offloading node available and being able to avoid bottlenecks posed by any single offloading target. Similarly, Drop computing is an idea brought forward by Ciobanu et al. [5], achieving mostly the same goals. Neither of the works details quite clearly, what the task exchange medium would be, nor discusses any developmental aspects of applications that could support their frameworks.

5 DISCUSSION

The idea of computation offloading between mobile devices is a great way of combating the network related difficulties that arise from Mobile Cloud Computing. It solves problems such as high latency, bandwidth limitations, and infrastructure cost. With research spanning a decade in length, no such solution has seen a practical implementation, not to mention large scale adoption. This is attributed to the multifaceted and long list of complications that still need solving.

First, it is difficult to convince individuals to volunteer their devices for helping others, if it does not offer them some kind of reimbursement. For services requiring managing from cloud, edge, or fog nodes, it's also difficult to imagine the motivations behind the ones hosting them.

Secondly, from a practical standpoint, no framework yet exists that could: require minimal effort to generically transform existing applications to be offloadable; not expect the worker device to already host or download some application specific binaries on demand; only transfer succinct representations of the inputs and results of the computation.

Lastly, most of the work related to task scheduling and allocation does not make effort to combine their results with an offloading framework to put together a working prototype.

There are some issues about which research has not been too vocal about, but may hinder development of such solutions. One example being the ever-changing rules that handset and operating system vendors impose on their devices. The latest mobile devices enforce increasingly stricter policies on the runtime behavior of users’ applications to save battery or increase security. Even at the moment, application developers are forced to find complicated workarounds for system restrictions2 (e.g. to run background processes). An offloading framework would undoubtedly be caught in such policies and maybe require the support and endorsement of the vendors to even work without extensive device modifications.

Up until now, most of the work does not make significant effort in proposing privacy preserving measures for offloaded inputs and calculations. Nonetheless, the issue of securing such systems might also prevent adoption of it in the real world. Usually, in the case of mobile cloud computing for example, trust is put into the cloud service provider to have the best intentions for the privacy of data provided to them. In the case of mobile clusters, it would limit the amount of devices safe to collaborate with to a single institution or social circle, which is far from the ambitions expressed by the discussed works.

It may be argued, whether it would even be beneficial to have such a technology in use. With the arrival of 5G cellular networks promising lower latency and high bandwidths, offloading needs of mobile devices might be completely serviceable by clouds or cloudlets. This would not solve infrastructure costs of workers, but presents a simpler alternative.

6 CONCLUSION

This report explored two areas of concern for a mobile device-to-device offloading vision. First, to answer the question of which programming models could potentially be used to achieve it, the most relevant existing frameworks were analysed. Due to there not being enough prototypes considering device-to-device specifically, other cloud based frameworks were inspected in hopes of finding a potential candidate. The constraints for the programming model and its accompanying framework were set in the best interest of the technologies’ adoption. Most interesting were the techniques, which required no changes to the application binaries and did not make large assumptions about the environment of the target device (e.g. if they have specific binaries installed). The most fitting candidate was the COMET framework as it had most potential for minimizing worker overhead, while automatically detecting the offloadable parts of an application. Further research should be conducted to figure out deployment possibilities of employing mobile devices as workers in a COMET-like framework.

The second half of this report surveyed recent research done specifically on mobile computation offloading with mobile offloaders. The general overview of publications in the last 4 years on this topic shows, that the direction has slowed down. Existing works have been getting increasingly focused on certain niches and issues of the idea, or has been transitioning to discuss the more general idea of hybrid offloading. Meanwhile, there is no obvious motivation to piece together the parts to create a working prototype.

7 ACKNOWLEDGEMENT

The author of this report has received the IT Academy Specialization Stipend for the Autumn semester of academic year 2021/22.

REFERENCES


2https://dontkillmyapp.com/