Understanding the Potential of Edge-Based Participatory Sensing: an Experimental Study

Breno Dantas Cruz, Junjie Cheng, Zheng Song, and Eli Tilevich
Software Innovations Lab, Virginia Tech
Email: {bdantas, ejunjie, songz, tilevich}@vt.edu

Abstract—Participatory sensing uses both local devices for data collection and cloud-based servers for processing. However, transferring the collected data to the cloud can lead to draining device battery power and cause network bandwidth bottlenecks, especially for large multimedia files. In this paper, we investigate how the processing resources at the edge of the network can be leveraged to enable efficient participatory sensing that avoids heavy network traffic. In particular, we report on the experiences of designing, implementing, and evaluating a sensing system that constructs indoor maps by recognizing door signs. A distinguishing characteristic of our system is an almost exclusive use of edge-based processing for tasks that include ML-based image recognition, human-assisted data verification, data model retraining, and administrative data flow aggregation. Our evaluation shows that our system architecture effectively leverages the available edge resources, while greatly reducing network traffic. Based on our experiences of implementing and evaluating our system prototype, we identify several open research directions for further advancing edge-based participatory sensing.

I. INTRODUCTION

Participatory sensing, first proposed more than a decade ago [1], introduces a radical idea of engaging ordinary citizens in collecting and sharing sensory data from their surrounding environment by using their smartphones [2]. Participatory sensing has been shown effective in capturing environmental and behavioral information for data analysis and knowledge discovery [3]. In traditional participatory sensing architectures, data collected by local devices is transferred to a remote server for processing. Recently, the increasing volumes of participatory data, particularly the rich multimedia formats for photo and video, often render cloud-based processing prohibitive due to the required network transfer, which causes the energy consumption and bandwidth utilization bottlenecks. Limited networks can be unsuitable for transferring large data volumes, as they get clogged while also draining the limited battery budgets of mobile devices.

As an alternative to dividing the data collection and processing responsibilities between local devices and remote servers, we present an alternate systems architecture, in which the data is processed locally at the edge of the network, close to where it is collected. The system leverages the computational and communication power of universally available WiFi routers. Nearby WiFi routers coordinate edge devices to cooperate on collecting and processing participatory sensing data. Specifically, each participating edge device executes a microservice to perform a certain functionality, and the output of a microservice can become input to another microservice, executed on another edge device, as coordinated by the WiFi router. Furthermore, edge devices dynamically download the executable packages of microservices from a cloud-based repository, thus enabling our system design to perform highly dynamic participatory sensing tasks. A cloud-based coordination server controls geographically dispersed WiFi routers in different areas. In our system design, the cloud-based coordination server only needs to schedule the sensing jobs for each WiFi router, instead of controlling and communicating with individual participants directly. As a result, our design reduces the communication complexity of transferring sensing tasks, obtaining results, and handling administration, such as distributing incentive rewards.

We report on our experiences of designing, implementing, and evaluating a sensing system that constructs indoor maps by recognizing door signs. In our demonstration use case, the system leverages edge-based processing not only to generate sensing results, but also to verify data, retrain machine learning models, and distribute incentive rewards. Our evaluation shows that our system architecture effectively leverages the available edge resources, thereby increasing the accuracy of the sensing results, while greatly decreasing network traffic. We also discuss the problems we encountered while designing and implementing our solution, and then identify several open research problems that stand on the way of fully realizing the vision of edge-based participatory sensing.

The rest of this paper is organized as follows: Section II summarizes the related state of the art. Section III introduces our system design and a running participatory sensing example. Section IV explains our use case’s implementation. Section V presents our evaluation results, and Section VI presents concluding remarks.

II. RELATED WORK

Participatory sensing is a methodology for collecting environmental and behavioral data by leveraging personal mobile devices [4]. Participatory sensing avoids the necessity of deploying and maintaining sensors, while the ubiquity of mobile users and their diverse mobility patterns enable collecting data from a variety of environments [3]. However, participatory sensing tends to generate large multimedia files or large collections of small-size sensory readings, types of data that are known to overwhelm the available network bandwidth, clogging the transmission [5].
The rise of edge computing can potentially solve the aforementioned problem. Edge computing processes data by means of edge devices near the data source [6]. In fact, edge computing has been applied to participatory sensing to reduce the required network transmission by processing raw sensory data on nearby edge devices [7], [8], [9]. However, these prior approaches either require that additional network devices be deployed on the edge [8].

This work provides general system-level support for executing a variety of participatory sensing tasks. Although our evaluation applies our system to one particular sensing task, our system architecture can support any sensing tasks, which can be expressed as a sequence of microservice invocations. Furthermore, our approach also fully supports the essential administrative procedures of participatory sensing (e.g., data verification [10] and incentive reward distribution [11]), all by leveraging the power of edge-based resources.

III. SYSTEM OVERVIEW

In this section, we first describe our participatory sensing system architecture, and then illustrate a typical workflow enabled by this architecture via a digital image recognition scenario.

![Participatory Sensing System Architecture](image)

**A. System Components**

Figure 1 shows the main components of our system architecture, which includes a cloud-based coordination server, a cloud-based microservice and data repository, a set of WiFi routers distributed across different regions, as well as mobile devices and edge-mounted servers, connected to these routers.

This architecture is designed for task owners, individuals or organizations that need to perform some participatory sensing task to infer some useful information. A typical workflow starts with a task owner interfacing with the cloud-based coordination server to submit a task to perform. A participatory sensing task contains 1) a general description, including the target region, the expected number of results, and the incentive rewards for the final results; and 2) directives for executing a sequence of microservices, which collect sensory data as well as process and evaluate the data to produce the final result.

A microservice encapsulates a simple functionality to execute on an edge device.

The coordination server then contacts the WiFi routers, passing them the directives. Each WiFi router coordinates a collection of surrounding devices to execute a sequence of microservices [12]. Each edge-based device downloads the execution package of its assigned microservice from the cloud-based repository, executes the microservice, returning the execution result back to the WiFi router. Henceforth, we use the term “job” to refer to the procedure performed by a WiFi router to orchestrate the execution of microservices on edge devices to compute one constituent result for a task.

**B. Use Case and Execution Flow**

Next, we introduce a use case that demonstrates the general operation of our system architecture. Consider a large office building with an intricate floor plan, making it hard for visitors to find their way around. To address this problem, the facilities manager decides to make available an indoor navigation map. As room numbers tend to remain constant, creating the navigation map involves mapping room numbers to their occupants’ names. Due to a high turnover rate, name tags change frequently and continuously. Hence, for each room, she needs to obtain its number and the door sign’s text, which displays the name of the room’s current occupant. Instead of manually collecting the names on the door signs for all the numerous rooms in the building, she can submit a participatory sensing task to collect this information. The expected results are a set of pairs of (name, room number). Instead of engineering a non-trivial custom participatory sensing system, one can simply connect the office building’s routers to our cloud-based coordination server, which will perform all required sensing tasks.

A sensing task to obtain the required information can be processed as follows. The central coordination server obtains a list of the WiFi routers in the task’s region. Based on its location, each router receives from the server a set of jobs. Executing each job generates an answer in the form of a pair (name, room number). Edge devices connected to a router cooperate to execute a job in accordance with the task’s workflow graph. The workflow specifies how to collect, process, and verify the sensing data involved by means of the edge devices. Hence, our system design makes it possible to reduce the amount of sensing data that needs to be transferred to the cloud. In the following discussion, we further explain how our system architecture coordinates a WiFi router and its nearby edge devices to execute a job.

IV. EXECUTING JOBS AT THE EDGE

Next, we describe how our architecture orchestrates the execution of jobs by means of edge-based computing devices.

**A. Work Flow Overview**

Figure 2 shows a workflow script that describes how to execute a job. The central coordination server transfers such job scripts to individual WiFi routers. A job script uses the
Figure 2: Work Flow Script

```
Service : {
    task_name: DoorSignCollection,
    incentive : 100,
    number: 100,
    expiration: 2018−10−15 22:00:00,
    microservices: {
        TakePhoto: {
            device: mobile phone,
            instruction: take_photo.pdf,
            on_success: PreprocessPhoto
        },
        PreprocessPhoto: {
            device: mobile phone,
            instruction: preprocess.pdf,
            on_success: RecognizePhoto
        },
        RecognizePhoto: {
            device: mobile phone,
            instruction: recognition.pdf,
            on_success: {
                in_sampling: threshold: VerifyPhoto,
                on_sampling: {
                    confirmed: exit,
                    refused: {
                        return: correction,
                        TrainModel
                    }
                },
                TrainModel: {
                    device: edge server,
                    on_success: [return: model]
                },
                TrainModelOnCloud: {
                    device: cloud server,
                    on_success: [return: model]
                }
            }
        },
        VerifyPhoto: {
            device: mobile phone,
            instruction: verification.pdf,
            on_success: {
                confirmed: exit,
                refused: {
                    return: correction,
                    TrainModel
                }
            },
            TrainModel: {
                device: edge server,
                on_success: [return: model]
            },
            TrainModelOnCloud: {
                device: cloud server,
                on_success: [return: model]
            }
        }
    }
}
```

The taking photo phase involves executing a photo-taking microservice on a selected mobile phone. This phase is not automatic, as it requires the phone user to use the device’s camera to capture the image in question. To that end, a descriptive instruction message is sent to the selected mobile phone, thereby asking the user to participate in executing the photo-taking microservice. If the user agrees to participate, she can follow the sent instructions and take the photo of the assigned room’s door sign. To complete the microservice’s execution, the runtime system on the mobile device then uploads the taken photo to the router.

The content recognition phase involves executing two microservices on the taken photo: 1) pre-processing and 2) recognizing. The pre-processing microservice transforms an image to make it suitable to be recognized using a machine learning model. To that end, the microservice converts the raw image to gray scale and applies filters to remove noise. Then, the recognition microservice takes the pre-processed image as input, and outputs the recognition result as a text string and a confidence value. The confidence value indicates the probability of the recognition result being correct. In our reference implementation, we use the OpenCV library for the image preprocessing microservice, and the TensorFlow library for the image recognition microservice.

### C. Edge-based Data Verification

Data quality has always been an overriding concern in participatory sensing [13]. Errors can be introduced to the collected results for the following two reasons: 1) hardware deficiencies or software bugs; 2) participants cheating providing fake data to earn rewards. Our system design helps ensure data quality by including a human-based verification microservice into the workflow. In this use case, as getting humans into the loop is costly, in terms of both execution time and incentive rewards, the system only applies this microservice to inspect randomly picked samples and to verify the recognized results with low confidence values.

Hence, if the reported recognition confidence is lower than the user-defined threshold, or the job is randomly selected for inspection, the human-based verification microservice is triggered. If a device owner agrees to perform the verification, she receives the same instructions as those sent to the human-assisted photo-taking microservice, as well as the actual taken photo and the corresponding recognition results; the user is asked to confirm whether the recognized result is correct for that image, and if not, what the correct result is. The returned result is used in three ways: 1) providing a verified recognition result; 2) updating the machine learning model of the image recognizing microservice; and 3) calculating the incentive rewards of the involved participants.

### D. Recognition Model Update

Despite the power of machine learning based image recognition, this technique is known to be vulnerable to errors. Hence, we use the results obtained from the data verification phase to dynamically improve the accuracy of the image recognizing microservice. If in the data verification phase, the user marks an image recognition result as wrong, while also providing the correct result, we use this information to update our image recognition model.

Our reference implementation makes use of federated learning. The original design of federated learning trains and evaluates machine learning models on distributed nodes. When a node observes an obvious improvement in recognition accuracy, it merges the model’s delta with the central model base. In our system design, the router finishes all jobs assigned to it by the coordination server, and feeds the pairs of images and their user-labeled content as training data to an edge server that executes the model update microservice. The microservice
outputs the delta of the image recognition model, and sends it to the cloud-based microservice repository.

E. Aggregated Incentive Distribution

The final phase of a job is distributing incentive rewards. Our system makes use of the hierarchical service contract strategy. For each task, the coordination server establishes contracts with routers to specify the expected results and their corresponding incentive payments. When a router finishes all assigned jobs, it uploads the results to the coordination server, and receives the incentive payment. Then, for each participant, the router further calculates its payment, as guided by the contract between them. In other words, the cloud-based coordination server has no need to negotiate with each participant about the incentive payment; instead, it negotiates only with routers, which expose the execution of participatory sensing jobs as an edge service.

V. Evaluation

The following research questions drive our evaluation: What are the performance/energy consumption tradeoffs offered by our system architecture? What is the level of accuracy that our system architecture can achieve?

A. Reference Implementation

The specific hardware used in our experiments are as follows: a Macbook Pro laptop as the central server, the TP-Link TL-WDR3600 as the router, a ZTE Android 4.0 smart phone for computationally non-intensive edge computation, and a Lenovo Ideapad 600 laptop for computation-intensive edge processing. We flash the OpenWrt system image on the TL-WDR3600 router, so the router can be used as a regular Unix-based operating environment. Specifically, we install on it an Nginx HTTP server and a SQLite database. All devices not only connect to the router, but also register their resource status with it.

B. Results and Analysis

1) Data Transmission: Table I shows the measured input and output data volumes for each microservice. In the taking photo microservice, the used photo comes from the nearby door signs, two of which appear in Figure 3. Pictures capturing large areas with extraneous information can be as large as 1.9 MB size. Our instructions explicitly request that users upload high-quality pictures.

<table>
<thead>
<tr>
<th>Microservice</th>
<th>Input size</th>
<th>Output Size</th>
<th>Avg. Energy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Taking Photo</td>
<td>0 KB</td>
<td>1.9 MB</td>
<td>949 uAh</td>
</tr>
<tr>
<td>Pre-processing Photo</td>
<td>1.9 MB</td>
<td>4 KB</td>
<td>80 uAh</td>
</tr>
<tr>
<td>Photo Recognition</td>
<td>4 KB</td>
<td>8 bytes</td>
<td>23 uAh</td>
</tr>
<tr>
<td>Photo Verification</td>
<td>4 KB</td>
<td>8 bytes</td>
<td>522 uAh</td>
</tr>
</tbody>
</table>

Table I: Data Transmission in Each Microservice

The pre-processing microservice converts the raw image to grayscale and applies filters to remove noise. After pre-processing, the size of the photo is compressed to 4 KB. In the photo recognition microservice, when the four digit number is successfully recognized, the result is returned as an 8-byte string. Therefore, edge-based processing requires transferring only 8 bytes of data vs. 1.9 MB that it would take to process the image in the cloud.

2) Recognition Latency: The recognizing photo microservice requires that the recognition model be updated whenever a new version becomes available. Due to the large size of recognition models (av. 17MB), they can take more than 7 seconds to transmit, as per our measurements. However, the time taken to download models can be used instead to recognize multiple photos using the old model. The shortened recognition time amortizes the cost of downloading an updated model. As the number of recognized photos reaches 20, the average execution time goes down to 0.68 seconds.

<table>
<thead>
<tr>
<th>MS</th>
<th>Downl.</th>
<th>Exec.</th>
<th>Human</th>
<th>Upl.</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Take Photo</td>
<td>0 s</td>
<td>0.24 s</td>
<td>8.12 s</td>
<td>0.22 s</td>
<td>8.58 s</td>
</tr>
<tr>
<td>Pre-process</td>
<td>0.23 s</td>
<td>1.28 s</td>
<td>0 s</td>
<td>0.1 s</td>
<td>1.612 s</td>
</tr>
<tr>
<td>Recognize</td>
<td>0.17 s</td>
<td>0.1 s</td>
<td>0 s</td>
<td>0.05 s</td>
<td>0.32 s</td>
</tr>
<tr>
<td>Verify</td>
<td>0.16 s</td>
<td>0.05 s</td>
<td>7.95 s</td>
<td>0.03 s</td>
<td>8.19 s</td>
</tr>
</tbody>
</table>

Table II: Microservices’ Execution Time

3) Energy Consumption: Since our edge server is not battery operated, we only evaluate four microservices executed on the mobile phone: 1) taking a photo, 2) pre-processing the photo, 3) recognizing the photo, and 4) verifying the photos recognized content. We use a Monsoon power monitor to measure the amount of energy consumed by the mobile phone executing each of these microservices. Since microservices executed on the mobile phone are human-assisted, the actual time taken and energy consumed depend on human behavior. To simulate realistic human behavior, we asked three different users to execute each microservice three times and averaged the results. Table II shows the execution time of each microservice. The total energy cost of executing microservices on the mobile phone is 1574 uAh. On the contrary, if the photo were processed by the cloud, we would only need to take a photo and upload, whose total energy consumption is 949 uAh.

4) Recognition Accuracy: The experimental results show that the confidence value is high in most cases. However, Table III shows a sensing outcome that produces a misclassified
digit. In this outcome, the digit 1 was incorrectly recognized as 2 due to its surrounding shadow. Since our system employs a confidence threshold procedure that verifies the correctness of the results, a human verifier was able to discover and correct the mistake, an action subsequently used to update the recognition model. Having executed the photo verification and the model retraining microservices, our system is able to correctly recognize this and similar photos. This experiment shows that the participatory sensing system eliminates as much as 99% of the data transmitted between the central server and the router, while providing the accurate recognition and reliable model updating mechanisms at the cost of additional 66% energy consumed by edge-based processing. We consider this trade-off acceptable, as the significant amount of additional energy consumed would not exhaust the mobile phones battery, while engaging edge-based processing helps increase the accuracy of the sensing results.

C. Discussion

We discuss our design options and how they impact the evaluation results. Generally, in a traditional participatory sensing system, the cloud-based central server is solely responsible for both coordinating the involved edge devices and processing sensory data. We divide the responsibilities of the central server into two parts: 1) data collection and processing; 2) administrative procedures. Our design introduces two basic ideas: 1) adopt edge computing to process the sensory data near the source, in order to reduce the volume of sensory data being uploaded to the cloud; 2) push some parts of the administrative load onto the edge.

As demonstrated by our evaluation, one benefit of edge-based participatory sensing is the ability to push much of the data processing functionality to the edge, thereby avoiding the necessity to transfer large volumes of unprocessed data to the cloud. Besides, we also demonstrate the benefits of refining the edge-based data processing module: by updating this module incrementally, one can improve the processing accuracy with minimal networking costs. The reference implementation also demonstrates that the administrative procedures required to manage participatory sensing tasks (e.g., incentive distribution, data verification, and participant selection) can be scheduled to execute distributively on edge-based WiFi routers. Compared with traditional participatory sensing systems that schedule such procedures to execute at cloud-based servers, our system design can also reduce the network traffic and the server’s computing load.

However, replacing the central server with distributed edge-based routers may introduce new problems. As the routers may overlap in their coverage ranges, some data can be collected more than once by devices connected to different routers. To avoid collecting redundant data, the participant selection procedure needs to be made aware of the overlapping coverage ranges, which presents a promising future work direction. Furthermore, our implementation relies on a fixed price incentive mechanism. Existing systems with central servers use online auctions to collect data with minimal costs. To increase sensing utility without transferring large amount of bidding data, the system design also needs a distributed dynamic price incentive strategy.

VI. CONCLUSIONS

As edge computing is steadily gaining prominence, researchers and practitioners alike are exploring the benefits of this computing modality for different distributed application domains. In this paper, we have investigated the tradeoffs of leveraging edge-based processing resources for participatory sensing. We have reported on our experiences of designing, implementing, and evaluating a realistic sensing system for constructing indoor maps by recognizing door signs. What makes our system design unique is its heavy reliance on edge-based processing for a variety of tasks. Our evaluation results indicate that our system architecture can indeed leverage diverse edge-based resources, thereby reducing network traffic.

ACKNOWLEDGMENT

This research is supported in part by the National Science Foundation through the Grant 1717065.

REFERENCES