Pushing Participatory Sensing Further to the Edge

Zheng Song, Junjie Cheng, Abhishek Chauhan, and Eli Tilevich
Software Innovations Lab, Virginia Tech
Email: {songz, cjunjie, zxcve, 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 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, and data model retraining. Our evaluation shows that our system architecture effectively leverages the available edge resources, while greatly reducing network traffic.**

I. Introduction

Participatory sensing, first proposed more than a decade ago [1], introduces a radical idea of engaging ordinary citizens in collecting and sharing sensor data from their surrounding environment by using their smartphones. 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. 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 router.

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.

II. Related Work

Edge computing processes data by means of edge devices near the data source. In fact, edge computing has been applied to participatory sensing to reduce the required network transmission by processing raw sensor data on nearby edge devices [2], [3], [4]. However, these prior approaches either require that additional network devices be deployed on the edge [3], or be custom-tailored for specific sensing tasks. In particular, RMCS [4] proposes an architecture that reduces the network load of participatory sensing by validating data at the edge via deep learning. However, the design of RMCS cannot be applied to all participatory sensing tasks, as it 1) only moves data validation onto the edge, while the cloud-based central server handles all other administrative procedures; 2) requires datasets to train deep learning models for each participatory sensing task. Since participatory sensing tasks tend to be highly customized for a given application, their datasets would be hard to predefine.

By contrast, 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.

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.

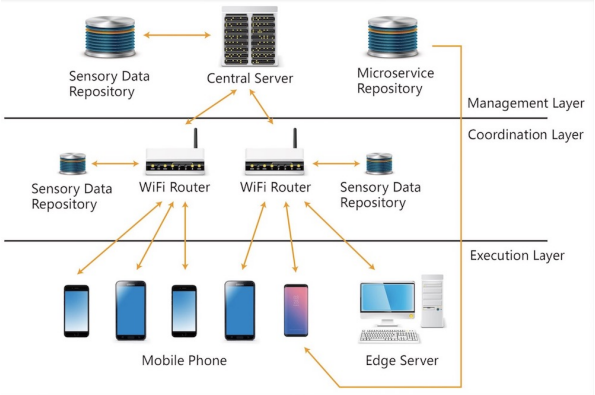

Figure 1: Participatory Sensing System Architecture

978-1-7281-2708-8/19/$31.00 ©2019 IEEE
DOI 10.1109/EDGE.2019.00019

A. System Components

Figure 1 shows the main components of our system: 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 their nearby routers.

This architecture is designed for task owners, individuals or organizations that need to perform a participatory sensing task that infers 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 sensor data as well as process and evaluate the data to produce the final result. A microservice encapsulates some simple functionality that needs to be executed on an edge device.

The coordination server then contacts the WiFi routers, passing them the directives. Each router coordinates a collection of surrounding devices to execute a sequence of microservices. 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 given 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. Workflow Overview

```
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,
      sampling_rate: 0.3,
      low_threshold: 99,
      on_success: {
        in_sampling|threshold_trigger: VerifyPhoto
      },
    VerifyPhoto: {
      device: mobile phone,
      instruction: verification.pdf,
      on_success: {
        confirmed: exit,
        refuted: {
          return: correction,
          TrainModel
        }}},
    TrainModel: {
      device: edge server,
      on_success: {return: model}
      on_failure: TrainModelOnCloud
    }
    TrainModelOnCloud: {
      device: cloud server,
      on_success: {return: model}
    }}}
```

Figure 2: Workflow Script

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 JSON format to provide instructions that specify how to execute a job by invoking a collection of microservices. Each microservice invocation contains the required input data, the

generated output data, the device selection rules, as well as which microservice to invoke next. Each WiFi router hosts a light-weight runtime system that executes the instructions of a given job script.

B. Workflow Execution

The workflow comprises five phases, each of which is implemented as one or more microservice: 1) taking photos; 2) recognizing content within the photos; 3) verifying recognized data; 4) updating the recognition model; and 5) distributing incentive credits to participating devices. Due to the fine-grained division of functionality into microservices, each of them can be executed independently by a variety of edge-based devices, ranging from mobile phones to edge servers.

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.

Data quality has always been an overriding concern in participatory sensing. 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. 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.

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 [5]. 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.

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 payments. 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 the incentive payment with each participant; instead, it negotiates only with routers, which expose the execution of participatory sensing jobs as an edge service.

V. Preliminary Evaluation

The experiment evaluates two system performance characteristics: data transmission and energy consumption. Table I shows the measured input, output data volumes, and the average energy consumption of each microservice. Our evaluation results indicate that our system architecture can indeed leverage diverse edge-based resources, thereby reducing network traffic.

Microservice	Input size	Output Size	Energy Consumption
Taking Photo	0 KB	1.9 MB	949 uAh
Pre-processing Photo	1.9 MB	4 KB	80 uAh
Photo Recognition	4 KB	8 bytes	23 uAh
Photo Verification	4 KB	8 bytes	522 uAh

Table I: Result of Each Microservice

Acknowledgment

This research is supported in part by the National Science Foundation through the Grant 1717065, and the MITRE Corporation.

References

[1] J. A. Burke, D. Estrin, M. Hansen, A. Parker, N. Ramanathan, S. Reddy, and M. B. Srivastava, "Participatory sensing," 2006.

[2] M. Habib ur Rehman, P. P. Jayaraman, M. Medhat Gaber *et al.*, "Rededge: A novel architecture for big data processing in mobile edge computing environments," *Journal of Sensor and Actuator Networks*, vol. 6, no. 3, p. 17, 2017.

[3] M. Marjanović, A. Antonić, and I. P. Žarko, "Edge computing architecture for mobile crowdsensing," *IEEE Access*, vol. 6, pp. 10 662–10 674, 2018.

[4] Z. Zhou, H. Liao, B. Gu, K. M. S. Huq, S. Mumtaz, and J. Rodriguez, "Robust mobile crowd sensing: When deep learning meets edge computing," *IEEE Network*, vol. 32, no. 4, pp. 54–60, 2018.

[5] J. Konečnỳ, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon, "Federated learning: Strategies for improving communication efficiency," *arXiv preprint arXiv:1610.05492*, 2016.