ARCHIVIO ISTITUZIONALE
DELLA RICERCA

Alma Mater Studiorum Università di Bologna
Archivio istituzionale della ricerca

ParticipAct: A Large-Scale Crowdsensing Platform

This is the final peer-reviewed author’s accepted manuscript (postprint) of the following publication:

Published Version:
Cardone, G., Corradi, A., Foschini, L., Ianniello, R. (2016). ParticipAct: A Large-Scale Crowdsensing Platform. IEEE TRANSACTIONS ON EMERGING TOPICS IN COMPUTING, 4(1), 21-32 [10.1109/TETC.2015.2433835].

Availability:
This version is available at: https://hdl.handle.net/11585/554183 since: 2021-03-18

Published:
DOI: http://doi.org/10.1109/TETC.2015.2433835

Terms of use:

Some rights reserved. The terms and conditions for the reuse of this version of the manuscript are specified in the publishing policy. For all terms of use and more information see the publisher's website.

This item was downloaded from IRIS Università di Bologna (https://cris.unibo.it/).
When citing, please refer to the published version.

(Article begins on next page)

28 July 2024

This is the final peer-reviewed accepted manuscript of:

Cardone, G., et al. "ParticipAct: A Large-Scale Crowdsensing Platform." ***IEEE Transactions on Emerging Topics in Computing*****, vol. 4, no. 1, 2016, pp. 21-32.**

The final published version is available online at: http://dx.doi.org/10.1109/TETC.2015.2433835

Rights / License:

The terms and conditions for the reuse of this version of the manuscript are specified in the publishing policy. For all terms of use and more information see the publisher's website.

This item was downloaded from IRIS Università di Bologna (https://cris.unibo.it/)

When citing, please refer to the published version.

ParticipAct: a Large-Scale Crowdsensing Platform

Giuseppe Cardone, Antonio Corradi, *Member, IEEE*, Luca Foschini, *Member, IEEE*, and Raffaele Ianniello

Abstract—In recent years, the widespread availability of sensor-provided smartphones has enabled the possibility of harvesting large quantities of data in urban areas exploiting user devices, so enabling the so-called crowdsensing that allows to realize complex applications impossible without the involvement of the research community. While many efforts have been made to improve specific techniques – spanning from signal processing to the assignment of data collection campaigns to users, and to the entire data processing – to the best of our knowledge, there are no active experiments aimed to explore the challenging issues raised by the management of large-scale crowdsensing campaigns as real-world experiments. This paper presents the ParticipAct platform and its ParticipAct living lab, an ongoing experiment at University of Bologna that involves 170 students for one year in several crowdsensing campaigns that can access passively smartphone sensors and also prompt for user active collaboration. In this article, we describe the guidelines behind the design of ParticipAct, its features, its architecture, and report quantitative results that assess and confirm the feasibility, obtained via intelligent coordination and management of crowdsensing campaigns.

Index Terms—Crowdsensing, distributed systems, location-dependent and sensitive, pervasive computing

1 Introduction

CONTINUOUS improvements in hardware manufacturing, CPU architectures, radio-communication techniques, and software design have constantly lowered the price of smartphones, thus fostering their far-reaching availability. The inevitable tipping point has been reached in 2013, when for the first time smartphones have outsold feature phones [1]: smartphones, while getting cheaper, are increasingly computationally powerful and equipped with more sensors (e.g., accelerometer, microphone, gyroscope, etc.) [2]. This spontaneous, widespread diffusion of Internet-connected sensor-equipped devices has enabled to accurately trace world-related information and (physical) activities of citizens by taking advantage of people willing to collaborate toward a continuous data harvesting process, called *crowdsensing*. That is especially true in smart cities areas where people bring almost constantly their smartphones.

The crowdsensing perspective asks for a powerful sensing platform where smartphones act as data sources sparse over the city and continuously feeding fresh raw sensing data. Moreover, recent advances in machine learning and artificial intelligence, supported by powerful computing resources available aboard, make smartphones intelligent probes able to process and extract high-level inferences from raw data (e.g., detecting user physical activity, voice detection, speaker detection) [3-7]. Recently, users begin playing an active role in crowdsensing, by enriching objective measurements taken via their phones with their subjective impressions and different actions. In addition, people can directly interact and coordinate with other close people, such as moving to places and coordinating with buddies and asking opinions.

Employing both users and devices to collect data from the real world poses significant social and technical challenges. From the social point of view, it is crucial to motivate users to participate, for example by providing useful crowdsensing-based services, handing out incentives, and fostering a sense of participation in a community [8, 9]. It is important not to overload users with duties over the limit of what they can contribute in crowdsensing. From a technical point of view, it is of paramount importance that sensing software that exploit user devices does not negatively impact on user experience, without reducing too much the quantity and quality of collected data. In any case, the boundary between social and technical challenges is not clear cut: for example, the minimization of the global resource overhead by considering a minimal subset of users in a crowdsensing campaign requires also analyzing geo-social profiles, to identify and infer which users are most likely to successfully harvest the required data.

Although there are currently many studies about crowdsensing, they mostly provide ad-hoc solutions and do not provide a field-tested general architecture to be customized for specific settings. Toward this goal, we have designed ParticipAct, a socio/technical-aware crowdsensing platform to investigate in a real-world scenario the social and technical issues of crowdsensing. ParticipAct started within the city of Bologna with students of the University of Bologna. For the ParticipAct experi-

• *G. Cardone, A. Corradi, L. Foschini, and R. Ianniello are with the Department of Computer Science and Engineering (DISI), Scuola di Ingegneria, Università di Bologna, 40135 Bologna, Italy. E-mail: giuseppe.cardone@unibo.it; antonio.corradi@unibo.it; luca.foschini@unibo.it; raffaele.ianniello@unibo.it*

ment, we enlisted 300 students (170 already active at the time of writing) for one year of active participation: we have selected students of different background belonging to different colleges. We provided them with modern smartphones, with unlimited data plan, asking them to allow us to collect data from sensors on the devices and to receive some tasks that require some form of active behavior, which they are free to accept or refuse.

To tackle that significant development effort, we distilled some guidelines from our experience with similar systems [10, 11]:

- *minimal intrusion on client devices*: resource consumption must be minimized to limit impact on user experience;
- *fast processing and feedback*: data collected from user must be quickly stored and processed so to reduce delay from data reception to data availability;
- *data sharing*: whenever possible, if the same data is required by more different processes, the content should be shared to minimize data duplication;
- *openness*: due to its long lasting nature and wide numbers, the architecture must easily expand to integrate new crowdsensing mechanisms;
- *security*: crowdsensing data are sensitive and integrity and confidentiality end-to-end must be granted;
- *complete data management workflow*: ParticipAct must support the whole data management cycle, from collection to transmission, from processing to mining and result provisioning.

These features highlight the most important contributions of ParticipAct that address all relevant issues for crowdsensing. Furthermore, we claim that a pure theoretical approach to crowdsensing systems does not face completely crowdsensing complexity. So, the contribution of ParticipAct is also in the fact that it is under test on the field by a large number of volunteers. In other words, ParticipAct is an extraordinary living lab for smart city managers: for example, by knowing that users in a particular area are moving by car and their speed is low, the city manager can infer that an area is congested. ParticipAct can be useful to coordinate and experiment large-scale crowdsensing activities:

i) it includes in the crowdsensing loop very different people (their set can be differently determined);

ii) it allows to launch and coordinate parallel crowdsensing campaigns over different groups of people;

iii) and it offers an easy-to-use interface for the generation and orchestration of new tasks to smart city managers.

From the technological point of view, ParticipAct is a complete crowdsensing platform consisting of an app running on smartphones and a web application on the back-end. Main supported functions include: *management of the crowdsensing requests* (called *tasks* in ParticipAct); *sensing of data* collected either passively (automatically via smartphone sensors), actively (with direct user participation), or with a mix of these modes; and *evaluating assignment of tasks* to users for future crowdsensing campaigns based on accurate data post-processing and mining. To realize these functions, ParticipAct traverses several different technological stacks, spanning from the smartphone OS Android, to Web technologies, to distributed data processing.

ParticipAct is the first real-world crowdsensing deployment that addresses not only technical issues, but considers also human resources and their involvement. ParticipAct is available to the community as an open-source platform that allows for fast development and deployment of large-scale experiments with minimal intrusion and resource usage on both smartphone and server sides. This paper focuses, after presenting ParticipAct architecture and main characteristics, on analyzing and evaluating different algorithms for assigning tasks based on user movements history. The main contribution is an on-the-field comparison in a real scenario of these algorithms in order to evaluate them and understand which is better and in which situation.

The rest of the paper is structured as follows. In Section II we describe our crowdsensing scenario, in Section III we describe the architecture of the client site, in Section IV we delve into the details of the platform back-end and in Section V we describe some core functionalities that make ParticipAct a customizable crowdsensing platform. Section VI reports some of the experimental results collected during the first deployment of ParticipAct. An analysis of related work (Section VII) and an assessment of the current state of the project and its future goals conclude the article.

2 Crowdsensing Model

In general, the goal of crowdsensing is to coordinate a (possibly large) group of people to gather a given – possibly complex – type of data, either by accessing sensors on user devices or asking users for active collaboration. Examples of possible crowdsensing campaigns are collecting geolocation data, asking for a photo of a given target, harvesting geotagged noise level, and so on.

There are several desirable features for a crowdsensing model that should aim at minimizing the complexity for participating users, while maximizing its expressiveness:

- *user freedom protection*: users should be able to accept/refuse any data collection campaign;
- *user privacy*: users should be able to stop and resume sensing at any time;
- *data transparency*: users should be fully aware of the kind of data collected in a crowdsensing campaign;
- *focused user load*: crowdsensing requests should go only to people likely to execute them, sharing the workload fairly among participating users;
- *geographical and temporal expressiveness*: crowdsensing should provide fine-grained parameters to define geographical and temporal bounds.

Following these principles, we designed the ParticipAct crowdsensing model. We represent any crowdsensing campaign as a *task* that is notified to users that may be available to collect requested data. This approach permits the definition of fine-grained simple operations (more likely to be completed by users) by dividing the burden between users: this is a common choice between other crowdsensing systems [11-14]. A task defines *what*, *when*, *where*, and *by whom* data should be collected. **What** should be collected is defined by fine-grained *sensing actions* that identify unambiguously the desired to be collected data. ParticipAct supports both data that can be automatically collected without user intervention - such as accelerometer, Wi-Fi scans, and ambient noise level (*passive sensing actions*)- and data that requires active user contribution - for example taking a photo, answering a survey, tagging a place (*active sensing actions*). ParticipAct allows to freely composing sensing actions, thus enabling the creation of tasks tailored to the needs of the manager of the crowdsensing campaign. The time metrics that dictate **when** data should be collected is expressed by three fields*: availability window*, *task duration*, and *sensing duration*. The availability window is the temporal interval when the task is available to users either to accept it or to refuse. Task duration represents the time users have to complete the task. Sensing duration defines how much sensing actions over sensors should be active to successfully complete the task. ParticipAct also allows defining **where** a task should be executed via *geonotification* and *geoexecution*. Geonotificated associates tasks to one or more geographical areas that must be traversed by users to receive the task notification. Geoexecuted, instead, associates tasks to one or more geographical areas and data collection can occur only to users located within. Finally, **by whom** is the set of people available for crowdsensing that had tentatively assigned to the task to either accept or refuse its execution.

We want to stress that ParticipAct tend to assign tasks to people more likely to successfully complete them, but workload should also be fairly distributed over users, to avoid disaffection in participating. The possibility of selectively choosing users for a task further enhances ParticipAct flexibility, because it allows – for example – to launch two campaigns (two tasks) with the same sensing actions to two different categories of people (e.g., students vs. employed, teen-agers vs. adults, etc.). The expressiveness of the crowdsensing model of ParticipAct makes it an ideal platform for crowdsensing in smart city scenarios, due to its capability of easily designing complex tasks (e.g., collaborative journalism, urban mapping) and quickly deploy it over a large population.

3 Task Lifecycle and Distributed Architecture

Any feature in the task structure plays a different role in the task lifecycle: this section describes all managing details of task lifecycle and its effects on the ParticipAct distributed architecture.

The task lifecycle is complex (Fig. 1) due to several nuances of the ParticipAct crowdsensing model. In particular, it requires to grant some state coherency to keep the task state synchronized between clients and the fixed server infrastructure (e.g. state transitions on clients should be completed if and only if the server acknowledges them) and to decouple sensed data and task lifecycle management (e.g., the uploading of sensed data should run independently and in parallel to other task-related communications). When a task is *created*, the server holds it back until it reaches its availability window, then it makes *available* and notifies it to the assigned users. Geonotified tasks are pushed to assigned users but they are set in a *hidden* state that withholds their availability until the smartphone verifies that the user is within the geonotification area. When a task is in the *available* state, users are free to decide either to accept it or not, based on task description. When a user refuses to run a task, it goes to the *refused* state, while if a user fails to decide to accept a task within the availability window, the task is automatically flagged as *ignored*. When a task is accepted, it goes to the *running* state that is the only state that enables data collection. At any moment users are free to temporarily pause data collection for the sake of privacy: in that case, the task is temporarily switched to *paused*, ready to be resumed later again. Tasks with geoexecution periodically check the current location, pausing data collection if outside the designed areas for data collection (*geostopped* state) and resuming it when again within

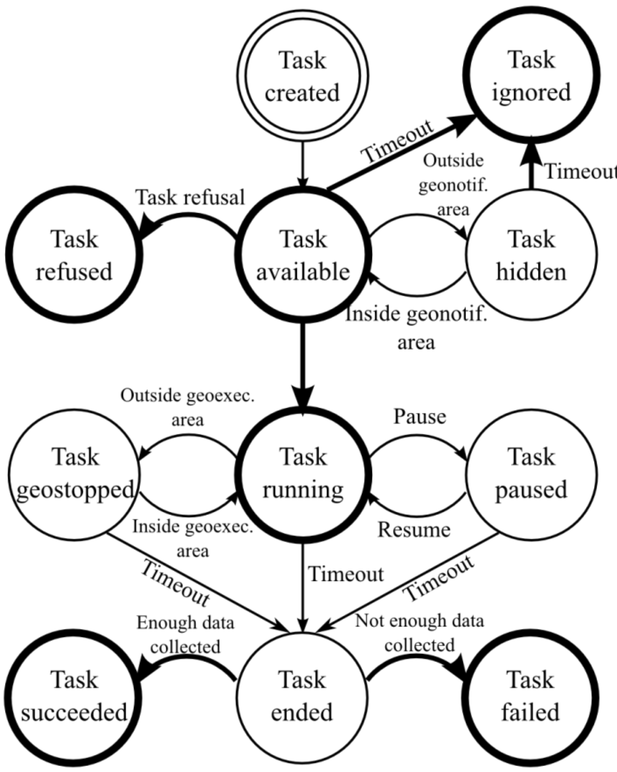


Fig. 1. Task state lifecycle. Data collection is enabled only in the *running* state. States with a bold stroke reached through transitions represented with a bold arrow are kept in sync between client and server.

bounds. When task duration time expires, the task switches to *ended* state, and evaluates data collection progress: if enough data has been collected, then the task switches to the *succeeded* state, otherwise it switches to the *failed* state. The evaluation whether a task has been successful requires to check the successful status of all its active and passive sensing actions: active actions are considered successfully completed whether the user carried out all required sensing actions (e.g., taking a picture, answering a survey), whereas passive actions are successful if, during the *task duration* window, required sensors have run for at least the *sensing duration*.

We argue that definition of ParticipAct tasks is flexible enough to correctly describe a wide range of crowdsensing campaigns. For example, by combining passive and active sensing actions it is possible to design a wide range of different crowdsensing campaigns focused either on technical or social aspects, or a mix of both. For instance, by combining geoexecution, geolocation, and cell signal strength it is possible to measure what zones of a given urban area could use one cell tower more (or less); combining geonotification, geoexecution, and taking pictures, it is possible to ask users to take a picture of a landmark.

The nature of ParticipAct requires a client-server architecture: a client runs on user devices to take care of receiving tasks and running them and sending results to a server that can store, analyze and use them. Thus, although all task status changes happen on clients, they should be shared with the server, so that crowdsensing managers always have accurate, updated information about the state of crowdsensing campaigns. To keep task state synchronized between clients and server, state changes should be allowed only when clients can communicate them to the server, thus making state transitions very costly, because network availability on mobile device is often spotty, thus delaying sync operations. Conversely, some state changes should not be synchronized between clients and server infrastructure because they are useful only on the client side (e.g., when a task is temporarily paused). That is the motivation of the state machine in Fig. 1: only important task state transitions are synchronized between clients and the server. In particular, transitions to *available*, *refused*, *ignored*, *running*, *succeeded*, and *failed* states are always synchronized with the server. If one of those transitions happens when there is no data connection, to avoid stalling the client, task state is implemented in a soft state that is finalized as soon as the server acknowledges it. In the next two sections, we describe the client and server architecture, whose design has been derived from these observations.

4 Client Architecture

The ParticipAct client is the component in charge of receiving tasks, asking users whether they want to run them, managing data collection, and uploading results. Functionally, the ParticipAct client consists of two macro-components: the *task management* one and the *sensing management* one (Fig. 2). These components orchestrate the full lifecycle of tasks on user devices and are responsible for interacting with users and accessing smartphone sensors.

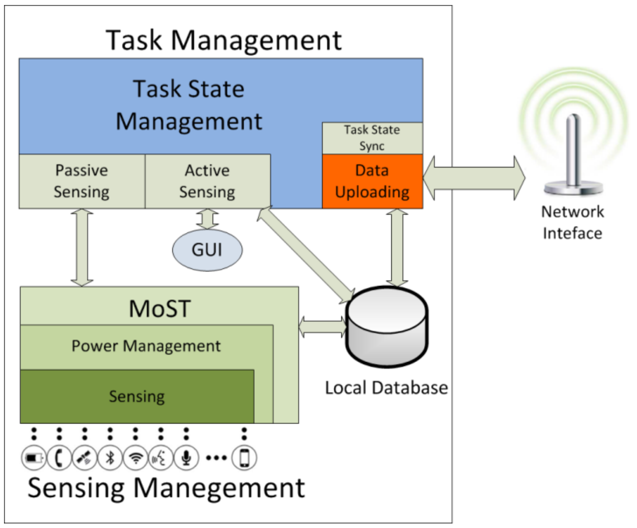


Fig. 2. Architecture of ParticipAct client.

4.1 Task Management

The *task management* macro-component takes care of overseeing the whole task lifecycle on smartphones. It has five main responsibilities realized by its components: i) receiving tasks from the server and keep their state synchronized; ii) providing users with an interface to control task execution; iii) implementing the Graphical User Interface (GUI) for active sensing actions with user interaction; iv) driving sensing actions; and v) uploading sensed data.

Task State Sync and *Task State Management* components are in charge of the first duty, by taking care of receiving new tasks and, in acceptance, by driving their full lifecycle. As stated previously, only important state transitions are communicated to the server and they occur only if the server acknowledges them.

The *Task State Management* component gives users the opportunity of completely controlling the sensing process: whenever an available task is pushed to user devices, the task management component gives users the opportunity to accept or refuse it, thus allowing different level of engagement in crowdsensing campaigns.

The *GUI* component enables sensing actions that require active user participation according to the third responsibility. Currently, ParticipAct supports four active sensing actions that allow to collect data available only through explicit user collaboration, hence enabling crowdsensing scenarios such as collaborative journalism, urban photographic mapping, and geotagging. The active sensing actions are *survey*, *take a picture*, *tag a place*, and *move to a place in a specific time*. ParticipAct implements a custom GUI for any of them.

Task Management also drives sensing actions: it starts/stops passive actions based on the current state of

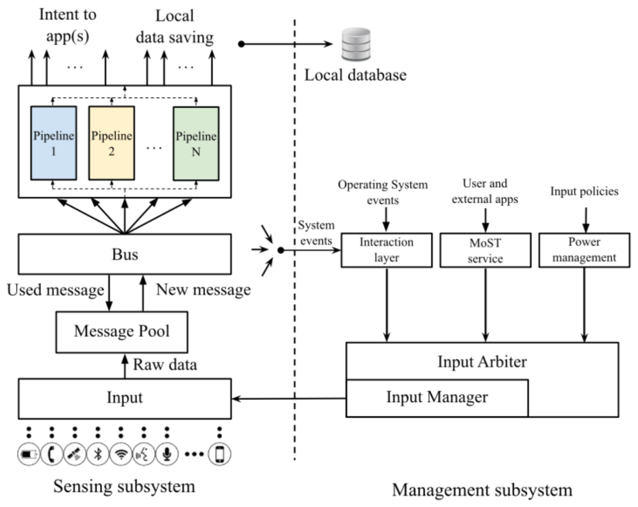


Fig. 3. MoST architecture.

the task, i.e., when the user pauses data collection or when the user moves outside the target area, for geoexecuted task. Actual activation/deactivation of sensors is demanded to the *Passive Sensing* component that sends appropriate requests to the *Sensing Management* macro-component.

Finally, the *Data Uploading* component is in charge of retrieving sensed data and uploading them on the server. This process has to balance between uploading data as soon as possible to the server and minimizing the power drawn by radio interfaces. According to the minimal intrusion principle, ParticipAct data upload is geared towards minimizing its impact on battery lifetime. To accomplish this, ParticipAct batches data uploads and requires a minimum interval of five minutes between two consecutive uploads, which grants the radio interface enough standby time [15]. The *Local Database* temporally stores data until the server acknowledges their reception, thus guaranteeing no data loss even in presence of unreliable data connections and client device shutdown.

4.2 Sensing Management

The sensing management component plays a pivotal role in ParticipAct crowdsensing because it manages the access to all sensors available on smartphones and the collection and processing of their output. Sensing is a power-hungry process that should be carefully driven to avoid negative impact on user, again following the minimal intrusion principle; in ParticipAct we have distilled three design guidelines. First, sensing should promote *availability of high-level inferences*, meaning that while accessing sensors on a smartphone (e.g., accelerometer) is a relatively trivial process, providing high-level inferences (e.g., the user is walking/running/standing) is a much more valuable feature; second, sensing should be *resource aware*: sensing management should put effort in minimization of resource consumption to both reduce impact on battery lifetime and limit detrimental effects on the device performance. Third, sensing system should be *system aware*: the sensing system coexists with the Operating System and other applications; with them it competes for unshareable resources (e.g., microphone and camera can be used only by one application at the time); the sensing system should resolve conflicts and promote a non-intrusive approach.

These principles drove us in the development of our sensing system, called MoST (Mobile Sensing Technology). MoST is our open-source Android sensing library that provides an uniform access layer to all physical and logical sensors, that eases the burden on app developer that want to use sensor data by providing data processing and power management, while taking into account concurrency issues due to access to shared resources, thus making sensing un-intrusive and minimizing impact on user experience. While MoST is a general-purpose library, in ParticipAct it effectively implements all the passive sensing actions[1].

MoST architecture is based on two building blocks: *Inputs* and *Pipelines*. *Inputs* are any physical or logical sources of sensing data (e.g., accelerometer, gyroscope, GPS, app networking statistics, battery level), while *Pipelines* are components that receive, process, and fuse sensed data collected from one or more *Inputs* and forwards resulting data to client applications. MoST consists of two main subsystems (Fig. 3): the *Sensing subsystem* and the *Management subsystem*. The *Sensing subsystem* has a two-layered architecture and manages all aspects of sensing, from accessing *Inputs*, to wrapping them into easy-to-manage local objects that are dispatched to *Pipelines*. *Pipelines* then forward their results to client applications. The *Management subsystem*, instead, drives the sensing process, providing an entry point to external apps to request MoST services, resolving concurrency issues for non-shareable resources (e.g., the microphone cannot be physically used by MoST during a phone call), and controlling power management. For more details on MoST, we refer interested readers to [5].

5 Server Architecture

The server side of ParticipAct provides management, storage, and analysis of crowdsensed data. At the highest level comprises two main parts, as shown in Fig. 4: the *Back-end* and the *Crowdsensing Manager*. The *Back-end* takes care of receiving, storing, and processing sensed data, while the *Crowdsensing Manager* provides the administrative interface to design, assign, and deploy sensing tasks.

In a more detailed view, the *Back-end* consists of three macro-components: *Data Receiver*, *Post processor*, and *Data Processor*. The *Data Receiver* that receives data from the client (namely, from *Data Uploading* component) via a Representational State Transfer (REST) API [16]. *Data Receiver*

[1] An updated list of sensors wrapped by MoST and the list of high-level inferences on user activities is available at http://participact.unibo.it.

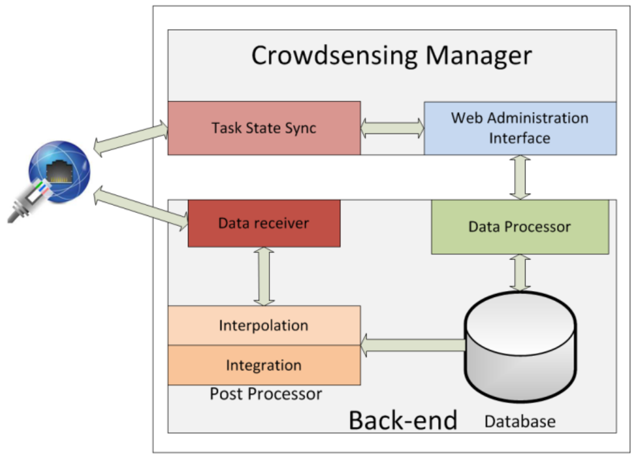
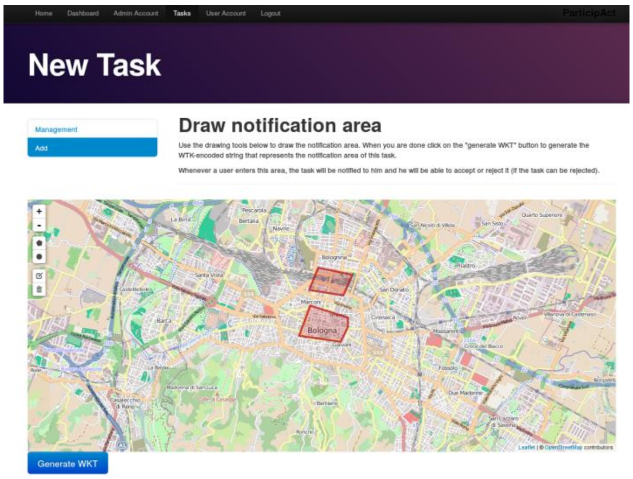


Fig. 4. Architecture of ParticipAct server side.



Draw notification area

Use the drawing tools below to draw the notification area. When you are done click on the "generate WKT" button to generate the WTK-encoded string that represents the notification area of this task.

Whenever a user enters this area, the task will be notified to him and he will be able to accept or reject it (if the task can be rejected).

Fig. 5. Screen capture of the ParticipAct Web Administration Interface. This figure shows the interactive page that allows to define the geonotification area of a task.

acknowledges each received data to allow data removal from the local database at the client side. Data is then cleaned up and prepared for long-term storage by *Post-processor* components: *Interpolation* and *Integration*. *Interpolation* improves data collection by filling in missing data points that can be inferred with sufficient accuracy. A notable case is geolocation data that on Android devices are available via Google geolocation API that dynamically switches between different techniques to infer user position (i.e., GPS, Wi-Fi, and cellular 3G) thus causes location accuracy to range and suddenly change from few meters (e.g., for GPS) to thousands of meters (e.g., for cellular 3G only). Interpolation substitutes data outliers, whose accuracy is significantly worse than the ones of temporally close data points, by substituting them with a linear interpolation of the more accurate data points. *Integration*, instead, aims at aggregating data in time and space. It collapses all data of any type collected in the same 5 minute window in a single row to enable time-based indexing of all sensed data so to speed-up the execution of temporal queries. It also aggregates data in space by creating a geographical view of sensed data, and storing it in a Geographic Information System database (GIS) for spatial querying. Finally, the *Data Processor* exploits those time-based and space-based views to tailor user profiles for fast identification of users that are more likely to successfully execute a task according to ParticipAct assignment policies detailed in the following Section 5.2.

The *Crowdsensing Manager* is the administrator-facing part of ParticipAct. The *Web Administration Interface* (Fig. 5) allows smart city managers to interact with the ParticipAct system and supports full-administration of the whole crowdsensing process, including management of user profiles, design and assignment of tasks, and data review. A core function of the Web Administration Interface is its ability of tapping into results provided by the Data Processor to automatically assign tasks to users that are more likely to successfully execute them. The *Task State Sync*, instead, is in charge of keeping task state synchronized between clients and server by pushing new tasks on designated clients and receiving all state change updates (e.g., task accepted/refused, task completed with success/failure). Among the several server functionalities, in the next two sections we present two core features that highlight unique challenges of the deployment of a real-world crowdsensing system: data transport and task assignment.

5.1 Data Transport

An essential feature of a crowdsensing system is *power-efficient*, *secure*, and *reliable* data transport from clients to the long-term storage hosted on the back-end.

Power efficiency is achieved on the client side (see also Section 4.1) by batching data transfers to minimize the number of times that network interfaces have to turn on to transmit data. Another important factor to reduce power consumption is limiting the amount of data that is actually being transferred. To achieve this, ParticipAct serializes the bulk of data using the highly efficient protobuf format [17], that greatly reduces the CPU consumption and data memory footprint compared to native Java serialization and verbose serialization formats such as JavaScript Object Notation (JSON) and eXtensible Markup Language (XML). Moreover, sensed data has often a low entropy (e.g., geolocation data points in a short time-span are similar), which makes them highly compressible. ParticipAct exploits this aspect by having clients compressing all outgoing data with the lightweight – yet power-effective – gzip algorithm [18].

As regards *security*, authentication is provided by enforcing the usage of HTTP Basic Authentication for all user requests [19]. Data integrity and confidentiality is guaranteed by industrial standards for encryption: all REST and web requests are available only via HTTPS over TLSv1 [20]. Moreover, HTTPS strengthens HTTP Basic Authentication, which by itself is a weak authentication mechanism prone to network sniffing attacks, by guaranteed that users and password are never sent in plaintext over the Internet.

Finally, ParticipAct achieves *reliability* with a two-phase commit protocol to grant that all data are transferred from clients and stored at the back-end.

5.2 Task Assignment

According to [21], a key factor to persuade users into executing tasks is making them easy, i.e. minimizing the effort to complete them successfully. ParticipAct profiling aims at identifying those users who are more likely to accept and complete a task without requiring them to modify their routines. At the current stage, ParticipAct task assignment policies mainly focus on geoexecuted tasks, that are of special interest for smart city managers to enable localized city surveillance and monitoring activities, by trying to involve in the campaigns users who are more likely to visit that area.

In particular, given a task with a geoexecution area, ParticipAct provides four different policies to select users that could be asked to run it: *random*, *recency*, *frequency* [11], and *dbscan*.

The *random* policy simply selects a random subset of all available users, regardless of their position history by allowing to choose the *user ratio* defined as the percentage of all available users to be assigned to the task, from 0% up to 100%.

The *recency* policy assigns to the task to users that have been recently in the geoexecution area under the assumption that they might still be therein, or go back to the designated area without changing their path too much. Moreover, the recency policy ranks all potential candidates according to how recently they have been in the geoexecution area, from the most to the least recent and, similarly to the random policy, also allows to choose the user ratio from 0% up to 100% defined as the portion of candidates (starting from higher ranked ones) to select.

The *frequency* policy assumes that people that spent more time in the geoexecution area in the past are the best ones to select, because it is a place where they stay or regularly attend. Accordingly, it selects users that have been in the target area in the past by ranking them accordingly to the time that they spent there. In addition, like the recency policy, the frequency policy supports the user ratio to further limit the number of assigned candidates.

Finally, the *dbscan* policy uses the Density-Based Spatial Clustering of Applications with Noise (DBSCAN) algorithm to cluster past user location traces so to select users that actually spend a sizeable amount of time in the target should be selected as potential candidates. DBSCAN is a density-based clustering algorithm based on the idea that the density of points inside a cluster is much higher than that of the points outside and the density of points outside a cluster is much lower that the density of any other cluster [22]. DBSCAN has several properties that make it well-suited for the task assignment problem: it does not require knowing the number of clusters a priori, it can detect arbitrarily-shaped clusters, it is robust to noise and outliers, and it is optimized to run on GIS-enabled databases. The dbscan policy runs the DBSCAN algorithm over all past user positions and selects users that are in a cluster that intersects the geoexecution area. Like all other policies, dbscan allows to select a user ratio; however, differently from recency and frequency policies, since DBSCAN does not provide a ranking, the dbscan policy chooses assigned users randomly among the subset of all candidates.

Let us conclude this section noting that all task assignment policies are sensible to the size of location history used; to avoid this problem, ParticipAct considers only a limited window of geolocation history of the last days before task start for each user, which currently is set to the two most recent weeks. At the same time, within that time frame, our policies consider in the candidates selection process all available users, even those ones that visited once in a lifetime a foreign city, because in that timespan they could be the most promising ones for certain areas. For example, tourists tend to spend more time downtown than local citizens for the duration of their visit.

Choosing a policy to select users for a given task has an impact on crowdsensing campaign results: since fewer users are going to execute them there is the risk to collect less data. Therefore, it is important to define some metrics to assess the performances of user selection policies. Some of the possible metrics are the number of users selected by a policy, the ratio of users that successfully completed a task compared over the count of selected users (precision), the ability of correctly predicting which users will execute a task and which not (accuracy), the time necessary to receive the first result, and the computational load due to policy execution. In the following section we experimentally evaluate the performances of ParticipAct policies according to these metrics.

6 Experimental Results

The experimental assessment of a large scale crowdsensing system in a realistic scenario poses significant social, technical, and logistic challenges. In a long-running effort to test ParticipAct, we have bootstrapped and we are currently maintaining a large deployment that involves 171 volunteers, all of them students of University of Bologna, that are attending courses on either the Bologna campus (121 students) or Cesena campus (50 students). Although, as for other similar experiments, it is an open question if obtained results could hold in a more general scenario, we believe the ParticipAct dataset is large enough (in time and space) to draw some first important observation, and rather realistic for urban setting scenarios. In fact, it is important to state that Bologna and Cesena university campuses are not self-contained: they comprise several dozens of different buildings spread over the metropolitan area. For this reason, volunteers path and behavior are not limited to a specific area, but related to the whole

	Task A	Task B	Task C	Task D
Completed	8.2% (14)	11.7% (20)	8.2% (14)	6.4% (11)
Failed	16.4% (28)	12.3% (21)	11.7% (20)	8.8% (15)
Rejected	17.5% (30)	19.9% (34)	31.6% (54)	36.8% (63)
Ignored	57.9% (99)	56.1% (96)	48.5% (83)	48.0% (82)

Table 1. Percentage of users that completed, failed, rejected or ignored the four test task. The number between parentheses is the raw number of students.

urban territory that coincides with the same area of all citizens living in the same smart city.

Of course, we cannot present all the tests we have done in the first ParticipAct period, because we have triggered many different campaigns. Hence, we present two main sets of experimental results: the first one is completely focused on a quantitative assessment of our assignment policies for geoexecuted tasks; the second one, instead, is an analysis of acceptance ratio for different types of non-geoexecuted tasks proposed to our volunteers. A discussion of lessons learnt from the experiments conducted in the ParticipAct living lab ends the section.

As regards deployment aspects, we provided each volunteer with a Samsung I8190 S III mini with pre-installed ParticipAct client and a SIM card with pre-paid unlimited data plan that they are free to use as primary mobile device. The ParticipAct client reports user geolocation every 180 seconds, thus allowing us to have very precise mobility traces. On the server side, ParticipAct was developed as a Spring MVC web application hosted on Apache Tomcat 7.0. The server hosting the web application uses an Intel i5 3210M 2.5GHz CPU, with that 8 GiB of RAM, and is connected via a 100Mbit connection to the server hosting the database that stores all crowdsensed data. The database server uses an Intel Xeon E31240 3.3GHz CPU and 8 GiB of RAM, and runs PostgreSQL v9.1 DBMS, that has been enhanced with PostGIS v1.5 to run geographical queries. All geolocation traces of users have been stored in a GIS-enabled table to allow fast geographical queries.

The first set of experimental results shows some performance figures related to optimizing the strategies for assigning tasks, maximizing success probability, and minimizing cost (i.e., involved candidates and computation overhead). We sent to all volunteers four different geoexecuted tasks over a period of a week, each of them asking to take a picture of a different place: Santo Stefano square in Bologna (task A), Bologna Opera House (task B), Cesena Train Station (task C), and Rocca Malatestiana castle in Cesena (task D). Table 1 shows for every task how many students accepted and successfully completed it on time, how many accepted it but failed to complete it, how many rejected to execute the task, and how many ignored the task (i.e., they neither accepted nor refused it). After receiving tasks results, we analyzed collected data to understand what performances would have been achieved if users had been selected by different ParticipAct policies and by using all possible user ratios from 10% to 100%. In

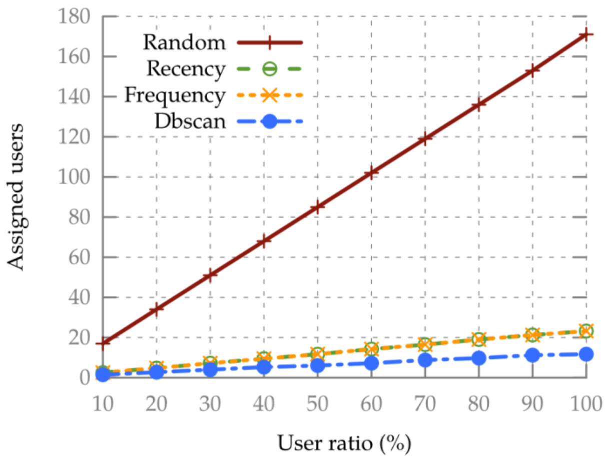


Fig. 6. Number of users assigned to tasks using different policies.

the following, all results have been averaged over the four tasks, because they were executed by the same population, they have comparable completion/failure rates, and they were associated to urban areas with similar characteristics. For policies that use a random user ranking (i.e., random and dbscan policies), results were averaged over 1000 random executions.

Fig. 6 shows the number of candidate users selected by each policy. Obviously, the random policy has the worst performances, indiscriminately selecting a large number of users that have no chance to execute a task. Recency, frequency, and dbscan policies always select about 20 users or less, with dbscan selecting about half the users compared to the recency and frequency policies. All the policies (except the random one) significantly limit the number of users assigned to a task, thus reducing the workload of users; this result is very significant, especially if considered together with the following results shown in Figs. 7-9, that all refer to these limited sets of candidates.

In addition, the number of selected users is related with the rate of success, whether selected users where those that actually executed successfully the tasks or not. For this evaluation, we consider as true positives (TP) users that have been selected by a policy and actually carried out the task, while false positives (FP) are users that have been selected by a policy and did not execute the task. Conversely, true negative (TN) users are the ones that have not been selected and did not execute the task, while false negatives (FN) the users that have not been selected but did execute the task. The rate of success is given by precision, calculated over the subset of selected candidates, and more formally defined as the ratio between TP and TP+FP. Fig. 7 shows the precision for the different policies implemented by ParticipAct. The random policy has a very low precision regardless of the user ratio while dbscan has a constant high precision (the highest compared to all other policies), which means that about half of the selected users actually executed the proposed task. Let us also note that the very regular behavior

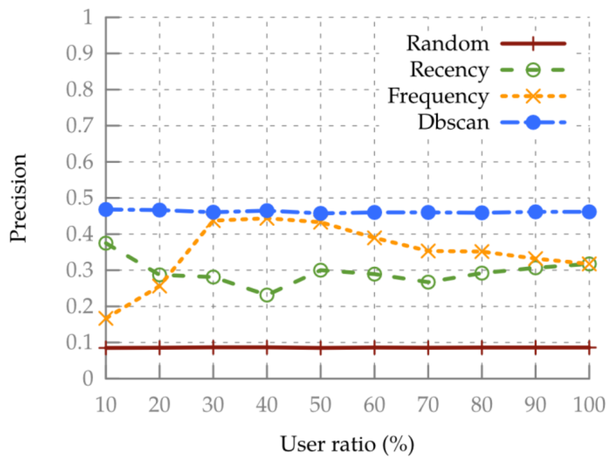


Fig. 7. Precision rate for different policies.

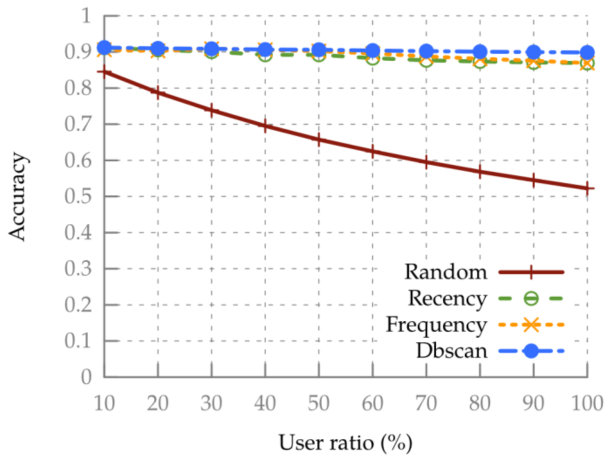


Fig. 8. Accuracy rate for different policies.

of the random and dbscan curves is due to the fact that since they do not rank users, as stated before we report average values for the four tasks and multiple (i.e., 1000) executions. As for the recency and frequency policies, instead, they are able to rank users, and we preferred to report the obtained values over the four executed tasks only. That is the main reason for the fluctuations and crossings, especially for low user ratios below 30% (with a corresponding total number of assigned users very low, below 7); for a higher number of executions, we expect these curves stabilize and have a more regular trend. So, focusing on the more significant [30-100]% user ration interval, the recency policy shows an almost-constant precision value, but performs worse than dbscan. The frequency policy, instead, works better (similar to dbscan) when the user ratio is between 30% and 60%, while selecting more than 60% users it includes too many users that will not cross again the target area in convenient time for task completion.

Another important metric is accuracy: the proportion of true results (both true positives and true negatives) in the population. It can be expressed by the ratio between TP+TN and TP+FP+TN+FN, that quantifies how good is each policy in correctly classifying user behavior and predicting whether they will (TP) or will not (TN) execute a task. Fig. 8 shows accuracy rate for all different policies. While the random policy has a bad accuracy, all other ones have a comparably high accuracy. Let us note that the denominator (TP+FP+TN+FN) is constant and equal to the total number of users and TP is relatively small, being the subset including the 6-12% of the users that successfully execute a task (see Table 1). That means that all informed policies (apart random) are able to obtain a high numerator (TP+TN) by correctly isolating those users that will not execute the task (TN), which in turns causes a high accuracy rate.

A negative impact of selecting only a subset of users for a task is that the time to receive the first result may increase. Fig. 9 shows that time for each policy to receive the first result. Missing datapoints signal that in at least one of the four tasks there were no results, thus making impossible to measure the time necessary to collect the first result; in addition, we use a logarithmic scale for the sake of presentation. The value of the random policy with a 100% user ratio is the lower bound: it is not possible to receive a result earlier than that. The recency and frequency policies have similar performances and quickly approach the lowest possible time to collect the first result, which is a very desirable property. In particular, the recency policy performs slightly better, but it is more at risk of not receiving any result when the user ratio is less than 50%. The dbscan policy always correctly selects at least one user that will collect the required data, but it approaches more slowly the lower bound. At the same time, let us remind that the overall number of assigned users selected by dbscan (at 100% user ratio) is nearly half the number of candidates at the same ratio for frequency and recency and dbscan grants also a higher precision and accuracy.

Finally, it is important to measure the computational time of different policies, in other words the CPU time required to select users given a task. On our deployment, running the random policy required on average 75ms to run, because it has been realized as a simple random selection on the table that stores users. The recency policy, instead, requires on average 187ms. That is still a short time because this policy requires to select geolocation data of users within the target area of the task and then to sort them by date; both these selections are greatly sped up by PostgreSQL and PostGIS indices. The frequency policy requires on average 24s, two orders of magnitude more than the recency policy. The main reason for that longer time is that the frequency policy needs to access all geolocation data for each user to evaluate the relative probability of being in the target area, and this process cannot be accelerated by database indices. Finally, the dbscan policy requires on average 4257s (more than 1 hour and 10 minutes) to run. This is due to the complexity of the DBSCAN algorithm, which needs to calculate the distance of all geolocation data couples of each user to

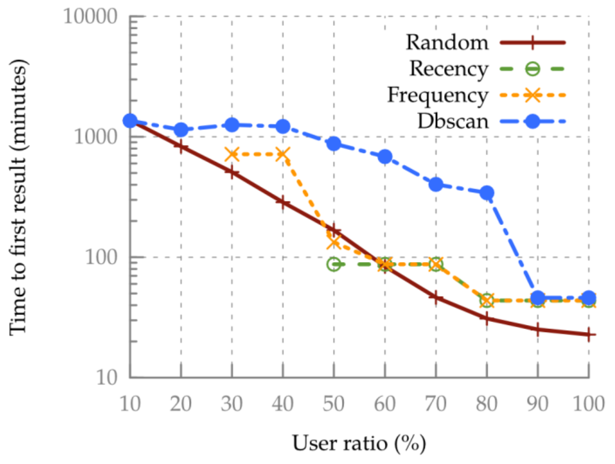


Fig. 9. Time to receive the first result for different policies. Missing datapoints mean that no user successfully completed the task for at least one of the four tasks.

identify clusters.

This evaluation of the random, recency, frequency, and dbscan policies shows that any policy has its strengths and weaknesses: the recency policy has slightly worse performances compared to the frequency policy, but it is computationally very lightweight, making it more suitable for very large scale deployments. On the other hand, the frequency policy has a significantly better precision and is less at risk of not having results with low user ratio compared to the recency policy. In any case, it is important to notice that the list of assigned users produced by recency and frequency contain the same users but they are ranked differently: this is why at 100% they reach always the same results. The dbscan policy has a high precision and is able to select a low number of assigned users due to its ability to capture and cluster user routinely behaviors, but it is computationally very expensive. In ParticipAct we provide all these policies, because we believe it is important to allow crowdsensing managers to strategically select the policy that more closely satisfy their current requirements depending on the geoexecuted task they have to schedule.

The second set of experimental results, instead, show some statistics about how our volunteers decide to accept (and successfully complete or not) other different types of non-geoexecuted tasks, that do not require to be in a place for completing the task. Table 2 reports collected statistics. Task E is a creativity test (we assigned three, reported numbers in Table 2 are average values) used to determine a human's creative potential, namely, a Remote Associate Test (RAT). For this task, we also give a very tight time limit to complete the riddle as required by this test type. Task F, instead, refers to a much longer and complex task. Users were requested to execute a task of City Mapping. This kind of task demands to users to walk through the city for 3 hours and, while phone sensors gather position and activity recognition information as passive sensing activities, take 5 photos of interesting places and add a comment to each photo. Obtained results show that Task E, that is more gaming-oriented, but especially much faster to complete, has been selected (including both completed and failed, because not completed on time) by a high percentage of volunteers, namely 61.3%. Task F, instead, requiring more commitment and being much more time consuming attracted a very low number of volunteers (below 13%); in addition, many of them failed to complete it, probably because they decided to abandon the task due to either drop of interest or due to some unpredicted event that interrupted them.

	Task E	Task F
Completed	48.0% (82)	2.9% (5)
Failed	13.3% (22.7)	9.4% (16)
Rejected	38.2% (65.3)	76.6% (131)
Ignored	0.6% (1)	11.1% (19)

Table 2. Percentage of users that completed, failed, rejected or ignored the two non-geolocalized tasks. The number between brackets is the raw number of students. For Task E, we report average values collected over three tasks of the same type.

Recalling that ParticipAct has the main objective of fostering new forms of participation for novel e-citizenship models in the smart cities environments and local communities governance, not only did we run several other crowdsensing campaigns, but also collected many surveys for feedback on user satisfaction and analyzed corresponding data. That experience has given us many insights and allowed us to draw some conclusions about the socio-technical management aspects of crowdsensing, which are very useful in designing new campaigns and refining the whole crowdsensing process.

First, crowdsensing platforms should ascertain and manage data quality: crowdsensed data should be refined by keeping into account also user trustworthiness (based on her history and reputation) and enlarging the number of selected users for the same time (to polish data via non-minimal crowdsourcing). For instance, we have observed a minor number of students trying to provide fake data: in most cases, we were able to dynamically detect them, for instance, when a user completes in a few minutes several (non-geoexecuted) tasks that would require taking a photo in places that are several kilometers away. In other situations, only human checking could validate the content, such as when a user, asked to take a picture of a monument, shoots a picture of her monitor that displays the requested monument.

Second, tasks should be as simple as possible to encourage user participation. As shown by our second set of experimental results, simple tasks are more easily accepted than complex ones; hence, crowdsensing platforms should avoid asking for big changes in user behavior by soliciting complex and unacceptable tasks they are likely to refuse. In particular, complex tasks can be difficult to understand correctly, can break users' daily routine or require too much effort that user is not willing or capable of reserve to them. An important topic is what kind of incentive can encourage users to execute more complex tasks; gamification, as demonstrated also by the high appreciation of the RAT test, can help as well.

Finally, although our task assignment policies are quite effective, nonetheless users that accept a task tend to forget about it, thus failing. From surveys conducted on our volunteers, they suggested us to remember them that they agreed to do something. At the same time, it is also important not to interfere too much. A possible solution, with regards to geoexecuted tasks, is to remember the users, when nearby, that a task in that area was requested and accepted.

7 Related Work

Interest in crowdsensing has seen a tremendous growth in the recent years thus promoting the development of several crowdsensing systems. A complete crowdsensing system covers several different research topics, including signal processing, machine learning, distributed systems, and social sciences. There are several research efforts focused on these aspects, considering each of them by themselves, while ParticipAct tries to tackle the whole stack of technical and social problems related to crowdsensing, which poses significant challenges. In this section, without pretension of completeness, we present some of the works that are close to the ParticipAct approach to crowdsensing.

Ohmage is an healthcare-oriented system that exploits smartphones to collect both passively and actively information about users [23]. Ohmage system architecture, similarly to ParticipAct, comprises an Android app to collect data and a back-end that allows to administer data requests and then visualizing and analyzing collected data. Differently from ParticipAct, Ohmage has no means to tie data requests to a specific geographic area, thus reducing its usefulness for smart city scenarios that could require users to be in a specific place to execute a task. Vita is a system that stresses the relevance of providing crowdsensing as a service integrated with usual software services and supports sensing task assignment based on user profiles [13]. To achieve the first goal Vita relies on BPEL4PEOPLE, a Business Process Execution Language extension that enables orchestration of human-driven sensing task within web services [24]. To achieve the second goal Vita assigns to tasks and users a so called "social vector", which is a synthetic representation of user resources and knowledge, and exploits it to assign a task to users whose profile suggests that they may enjoy that task and have enough resources to complete it successfully. While Vita provides a nice support for non-geoexecuted tasks, it completely lacks support for advanced task assignment policies for geoexecuted tasks based on user movement history. Matador is a crowdsensing software that focuses on context-awareness to optimize task assignment while minimizing battery consumption [14]. In particular, Matador assumes that a task is defined by a geographical dimension and a temporal dimension and should be assigned to users that are within the given geographical area in the given temporal window, and drives the sampling time of user positions to minimize battery consumption, dynamically switching between network-based geolocation, power-efficient but inaccurate, and GPS, power-hungry but very accurate. ParticipAct adopts a more proactive approach and, differently from Matador, allows to assign geoexecuted tasks to volunteers based on their past mobility history, without assuming constant communications at runtime, but only requiring lightweight and infrequent geolocalization sampling at the client device. USense is a middleware for community sensing that strongly decouples users collecting data and managers that require crowdsensed data: managers specify which kind of data they need and USense matches them with people meeting the requirements [25]. A notable feature of USense is its flexible policies for smartphone sensors duty cycling, which allow to minimize battery consumption of sensing activities. Similarly to USense also the MoST sensing core of ParticipAct support duty cycling of passive sensing activities. Finally, the Medusa framework focuses on algorithms to define crowdsensing tasks [12]. Medusa is based on a domain-specific programming language that provides high-level abstraction to define crowdsensing tasks, and employs a distributed system that coordinates execution of those tasks between smartphones and a cluster in the cloud. By providing programming abstractions for the definition of the tasks Medusa is complementary to our work, but at the current stage it lacks task assignment management support of geoexecuted tasks and, similarly to Matador and Vita, it also lacks the signal processing and machine learning support to automatically collect high-level inferences about user activities. Let us conclude this section by noting that compared to all above systems ParticipAct has been tested on a much larger user base and for a significantly longer duration.

8 Conclusion

This paper describes ParticipAct, an ongoing project of the University of Bologna that involves 170 students that participate to a large-scale crowdsensing experiment. ParticipAct is the first real-world crowdsensing deployment that addresses not only technical issues, but considers also human resources, their use, and involvement. ParticipAct is available to the community as an open-source platform that allows for fast development and deployment of large-scale experiments with minimal intrusion and resource usage on both smartphone and server sides.

This work paves the way to a new generation of real-world large-scale crowdsensing testbeds able to truly verify any step in the whole crowdsensing process, from task scheduling to incentive, and mobile sensing, as an effective monitoring solution for future smarter cities.

Acknowledgements

This research was supported in part by CIRI, Center for ICT technology transfer of the University of Bologna, funded by POR FESR Emilia-Romagna 2007-2013.

REFERENCES

[1] A. Gupta, R. Cozza, and C. Lu, "Market share analysis: mobile phones, worldwide, 4Q13 and 2013," Gartner Inc., 2014.

[2] N. D. Lane, E. Miluzzo, H. Lu, D. Peebles, T. Choudhury, and A. T. Campbell, "A survey of mobile phone sensing," *IEEE Communications Magazine,* vol. 48, no. 9, pp. 140-150, 2010.

[3] H. Lu, J. Yang, Z. Liu, N. D. Lane, T. Choudhury, and A. T. Campbell, "The Jigsaw Continuous Sensing Engine for Mobile Phone Applications," in *SenSys '10: Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems*, 2010, pp. 71-84.

[4] E. Miluzzo, C. T. Cornelius, A. Ramaswamy, T. Choudhury, Z. Liu, and A. T. Campbell, "Darwin phones: the evolution of sensing and inference on mobile phones," in *MobiSys '10: Proceedings of the 8th international conference on Mobile systems, applications, and services*, 2010, pp. 5-20.

[5] G. Cardone, A. Cirri, A. Corradi, L. Foschini, and D. Maio, "MSF: An Efficient Mobile Phone Sensing Framework," *International Journal of Distributed Sensor Networks,* no. 2013.

[6] G. Cardone, A. Cirri, A. Corradi, L. Foschini, and R. Montanari, "Activity recognition for Smart City scenarios: Google Play Services vs. MoST facilities," in *ISCC '14: Proceedings of the 19th IEEE Symposium on Computers and Communications*, 2014.

[7] S. A. Hoseini-Tabatabaei, A. Gluhak, and R. Tafazolli, "A survey on smartphone-based systems for opportunistic user context recognition," *ACM Computing Surveys,* vol. 45, no. 3, pp. 1-51, 2013.

[8] B. J. Fogg, "Persuasive computers: perspectives and research directions," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 1998, pp. 225-232.

[9] B. J. Fogg, "Persuasive technology: using computers to change what we think and do," *Ubiquity,* vol. 2002, no. December, p. 2, 2002.

[10] G. Cardone, A. Corradi, L. Foschini, and R. Montanari, "Socio-technical Awareness to Support Recommendation and Efficient Delivery of IMS-enabled Mobile Services," *IEEE Communications Magazine,* vol. 50, no. 6, pp. 82-90, 2012.

[11] G. Cardone, L. Foschini, P. Bellavista, A. Corradi, C. Borcea, M. Talasila, *et al.*, "Fostering Participaction in Smart Cities: a Geo-Social Crowdsensing Platform," *IEEE Communications Magazine,* vol. 51, no. 6, pp. 112-119, 2013.

[12] M.-R. Ra, B. Liu, T. La Porta, and R. Govindan, "Medusa: A Programming Framework for Crowd-Sensing Applications," in *MobiSys '12: Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services*, 2012, pp. 337-350.

[13] H. Xiping, T. H. S. Chu, H. C. B. Chan, and V. C. M. Leung, "Vita: A Crowdsensing-Oriented Mobile Cyber-Physical System," *IEEE Transactions on Emerging Topics in Computing,* vol. 1, no. 1, pp. 148-165, 2013.

[14] I. Carreras, D. Miorandi, A. Tamilin, E. R. Ssebaggala, and N. Conci, "Matador: Mobile task detector for context-aware crowd-sensing campaigns," in *PERCOM Workshops '13: Proceedings of the IEEE International Conference on Pervasive Computing and Communications Workshops*, 2013, pp. 212-17.

[15] N. Balasubramanian, A. Balasubramanian, and A. Venkataramani, "Energy consumption in mobile phones: a measurement study and implications for network applications," in *IMC '09: Proceedings of the 9th ACM SIGCOMM conference on Internet Measurement Conference*, 2009, pp. 280-293.

[16] R. T. Fielding and R. N. Taylor, "Principled design of the modern Web architecture," *ACM Transactions on Internet Technology,* vol. 2, no. 2, pp. 115-150, 2002.

[17] N. Gligoric, I. Dejanovic, and S. Krco, "Performance evaluation of compact binary XML representation for constrained devices," in *DCOSS '11: Proceedings of the IEEE International Conference on Distributed Computing in Sensor Systems and Workshops* 2011, pp. 1-5.

[18] Y. Chen, A. Ganapathi, and R. H. Katz, "To compress or not to compress - compute vs. IO tradeoffs for mapreduce energy efficiency," in *Green Networking '10: Proceedings of the 1st ACM SIGCOMM workshop on Green networking*, 2010, pp. 23-28.

[19] T. Berners-Lee, R. Fielding, and H. Frystyk. (1996). *Hypertext Transfer Protocol - HTTP/1.0*. Available: http://tools.ietf.org/html/rfc1945

[20] E. Rescorla. (2000). *HTTP over TLS*. Available: http://tools.ietf.org/html/rfc2818

[21] B. J. Fogg, "A behavior model for persuasive design," in *Persuasive '09: Proceedings of the 4th International Conference on Persuasive Technology*, 2009, pp. 1-7.

[22] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise," in *KDD '96: Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining*, 1996, pp. 226-231.

[23] N. Ramanathan, F. Alquaddoomi, H. Falaki, D. George, C. Hsieh, J. Jenkins, *et al.*, "ohmage: An open mobile system for activity and experience sampling," in *PervasiveHealth '12: Proceedings of the 6th International Conference on Pervasive Computing Technologies for Healthcare*, 2012, pp. 203-204.

[24] M. Kloppmann, D. Koenig, F. Leymann, G. Pfau, A. Rickayzen, C. von Riegen, *et al.*, "Ws-bpel extension for people–bpel4people," *Joint white paper, IBM and SAP,* vol. 183, no. p. 184, 2005.

[25] V. Agarwal, N. Banerjee, D. Chakraborty, and S. Mittal, "USense - A Smartphone Middleware for Community Sensing," in *MDM '14: Proceedings of the 14th IEEE International Conference on Mobile Data Management*, 2013, pp. 56-65.

Giuseppe Cardone (M'10) graduated from the University of Bologna, Italy, where he received a Ph.D. degree in in computer engineering in 2013. He is now a site reliability engineer at Google, London. His interests include performance and scalability issues of distributed systems, urban mobile sensing, and power-aware middleware solutions.

Antonio Corradi (M'77) graduated from University of Bologna, Italy, and received MS in electrical engineering from Cornell University, USA. He is a full professor of computer engineering at the University of Bologna. His research interests include middleware for pervasive and heterogeneous computing, infrastructure support for context-aware multimodal services, and network management. He is member of IEEE and ACM.

Luca Foschini (M'04) graduated from University of Bologna, Italy, where he received PhD degree in computer science engineering in 2007. He is now an assistant professor of computer engineering at the University of Bologna. His interests include distributed systems for pervasive computing environments, system and service management, and Cloud computing. He is member of IEEE and ACM.

Raffaele Ianniello graduated from the University of Bologna, Italy, where he received a Master degree in computer engineering in 2013. He is now a research assistant at the same university. His interests include distributed systems, spatial analysis, semantic web, and mobile sensing.