REPC: Reliable and Efficient Participatory Computing for Mobile Devices

Zheng Dong*§, Linghe Kong*, Peng Cheng*†, Liang He*, Yu Gu*, Lu Fang††, Ting Zhu‡, Cong Liu§
*Singapore University of Technology and Design, ††University of Science and Technology of China,
†Zhejiang University, ‡State University of New York, §The University of Texas at Dallas
Email: {dong_zheng, he_liang, jasongu}@sutd.edu.sg, linghe.kong@sjtu.edu.cn, pcheng@iipc.zju.edu.cn,
fanglu@ustc.edu.cn, tzhu@binghamton.edu, cong@utdallas.edu

***Abstract*—Smartphones and mobile devices have greatly penetrated the daily lives of many people. While participatory/pervasive sensing has gained wide adoptions by leveraging various onboard sensors on mobile devices, another powerful resource, the computational power on these mobile devices has been less frequently harnessed by researchers and practitioners. To fill this gap, we propose in this work the modeling, analysis, and implementation of *participatory computing*. Specifically, we propose REPC, a generic randomized task assignment framework for the participatory computing paradigm, which guarantees the overall system performance with close to minimal workload at individual participating devices. To achieve these design objectives, we model the intrinsic relationship between the workload of individual devices and the probability they complete their assigned tasks. Based on our modeling results, we analyze the maximal system capacity for any given participatory computing system and derive the minimal workload for individual participating devices to achieve the overall system performance requirement. We have fully implemented our design on the Android platform and demonstrated its performance through a representative participatory computing application. Extensive experiments and simulation results demonstrate that our design is able to achieve more than 90% task completion ratios with only 10% system overhead in practice.**

I. Introduction

According to a recent study of ABI research [1], the number of smartphones around the world will be 1.4 billion by the end of 2013. By leveraging various sensors on these smartphones, participatory sensing has harnessed the power of this largest and already deployed sensor networks to enable applications in areas such as air pollution [19], urban public transports[24], and crowd counting [17]. However, the computational power of mobile devices, which increases dramatically in the last decade, has to be effectively utilized in order to advance participatory sensing from a potential to a reality.

In this paper, we propose the modeling, analysis, and implementation of *participatory computing*, where the computational power of mobile devices can be leveraged to execute sensing tasks that are computational. To realize efficient participatory computing, we have to tackle several critical research challenges. First, different from traditional computing devices, computational power on mobile devices such as smartphones have highly unpredictable and intermittent availabilities. Moreover, many sensing tasks such as merchandise price tracking [10], offloading video transcoding [16] and real-time crime search [21] are delay-sensitive and often possess a tight response time requirement in the range of a few minutes or seconds. Last but not the least, different from many distributed systems that achieve fault tolerance by checkpointing or replication [9], [22], which either incurs high task management cost or high overhead for individual machines, system reliability in participatory computing has to be ensured in a way such that overheads at individual participating devices are minimized.

978-1-4799-4657-0/14/$31.00 © 2014 IEEE

Motivated by above observations, in this paper we consider the problem of assigning and executing a set of sensing and computing tasks on participating smartphones in a reliable and timely manner. We develop a simple yet highly effective task assignment framework, which we call REPC, based upon the real-time statistics of participating devices in the system. Through theoretical modeling and analysis, as well as extensive experiments and simulations, we demonstrate that even though participating devices may join or leave the system intermittently with limited durations, our design is able to guarantee a specified threshold of completed tasks in the task pool while minimizing the number of tasks executed on individual participating devices. The main contributions of this paper can be summarized as follows:

- We propose an effective randomized task assignment framework, REPC, to achieve reliable and timely computations using mobile devices for participatory computing with close to optimal overhead. To the best of our knowledge, this is the first work that presents a practical participatory computing framework.
- To understand the intrinsic properties of participatory computing, we explore the fundamental relationship between the workload at individual devices and the probability for individual devices to complete their assigned tasks. Furthermore, we analyze the maximal system capacity with statistics of participating devices.
- Aided by the system modeling results, we minimize the number of tasks to be executed by individual devices while guaranteeing the system reliability requirement.
- We have fully implemented an application of searching for wanted criminals with participatory computing on the android platform with up to 20 mobile users. The experimental results, combined with large-scale simulation results, demonstrate that our design achieves reliable participatory computing with very low system overhead.

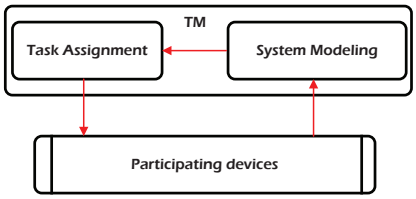


Fig. 1: Participatory Computing System Architecture

II. Design

We present our main design in this section. The basic architecture for participatory computing system is shown in Fig. 1. Specifically, the system is composed of a task manager (TM) and participatory devices. The TM serves as a centralized server, assigning tasks to participatory devices within its communication range. The TM also estimates the statistics of participatory devices in a real-time manner. Based on online estimation, the TM dynamically assigns tasks to each individual participatory devices in order to satisfy the requirements of the participatory computing system. The notation used throughout the paper is summarized in Table I.

An overview of our design is given as follows. We first formally model the TM and participating devices in Secs. II-A and II-B. By analyzing these modeling results, we obtain the maximal system capacity (i.e., the number of tasks that are expected to complete on a given set of participating devices) in Sec. II-C. Then by utilizing properties of the derived maximal system capacity, we present in Secs. II-D and II-E techniques that minimize the number of assigned tasks on individual participating devices while guaranteeing overall system performance and reliability requirements.

A. Basic Task Manager Design

At any specific time, we assume the Task Manager (TM) has a task set $\mathcal{T} = \{T_1, T_2, \cdots, T_n\}$ with n tasks, where T_i denotes each individual task to be executed by participating devices. For participating devices, we assume they can locally decide whether or not to accept tasks requested by the TM and can abort the execution of assigned tasks at any time. Specially, the participating devices can not return their completed tasks to TM if they move too far away to communicate with TM. Therefore, to ensure all tasks at TM can be successfully computed by the participating devices and returned to TM with high confidence, the TM must make task assignment decisions based upon the behaviours of participating devices. Specifically, for a participating device i, its task assignment at TM can be represented by the following equation:

$$d_i = \begin{bmatrix} a_{i,1} & a_{i,2} & \cdots & a_{i,n} \end{bmatrix} \begin{bmatrix} T_1 \\ T_2 \\ \cdots \\ T_n \end{bmatrix} \tag{1}$$

where $a_{i,j}$ equals 0 or 1. For example, for a task set $\mathcal{T} = \{T_1, T_2, T_3\}$, if the task assignment vector d_i is $[1, 0, 1]$, then both T_1 and T_3 are assigned to device i.

Because the computational capabilities, as well as the available execution times could vary significantly across different devices, there is no guarantee that any participating device can finish its assigned tasks. To shed the light on the basic TM design, let us first assume there are M devices that successfully complete their tasks (In Section II-B2, we discuss how to obtain such M value in practice). Then the computation results received from these M devices can be expressed as:

$$\begin{bmatrix} d_1 \\ d_2 \\ \cdots \\ d_M \end{bmatrix} = \begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \cdots & \cdots & \cdots & \\ a_{M,1} & a_{M,2} & \cdots & a_{M,n} \end{bmatrix} \begin{bmatrix} T_1 \\ T_2 \\ \cdots \\ T_n \end{bmatrix} \tag{2}$$

where d_i denotes the tasks that have been completed by device i. Consequently, $\sum_{i=1}^{M} a_{i,j} = 0$ means that task j has not been successfully completed by any device.

If TM randomly assigns a task to a participating device with a probability p, and any M devices return the assigned tasks successfully, then the expected number of completed tasks in the task set $\mathcal{T}$ can be calculated by

$$E(\mathcal{T}) = \sum_{i=1}^{n} (1 - (1-p)^M) = n(1 - (1-p)^M) \tag{3}$$

where $(1-p)^M$ is the probability that task i is not completed by any participating device, and $1 - (1-p)^M$ denotes the probability that there is at least one device that has completed the task i. By summing the independent probability for all tasks, we then obtain the expected number of completed tasks in the task set $\mathcal{T}$.

For example in Fig.2 , if there are 100 tasks in the task set $\mathcal{T}$, and TM randomly assigns 15 tasks ($p = 15\%$) to individual devices, then the expected number of completed tasks is 96 assuming that there are 20 participating devices that compete their assigned tasks.

Given a threshold R on the number of completed tasks, our design objective in this work is therefore to minimize the work-

TABLE I: Summary of notation.

n	The size of task set
p	Task assignment probability of TM
R	Threshold on the number of completed tasks
λ	The arrival density
μ	The departure density
t_c	The average time for each device completes one task
$M(p)$	The expected number of devices complete their assigned tasks
$E_p(\mathcal{T})$	The expected number of completed tasks in set $\mathcal{T}$

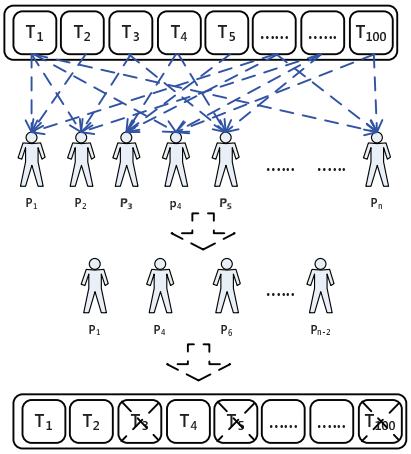


Fig. 2: An Illustrative Example

load at individual participating devices while guaranteeing the given threshold on the completed tasks. Mathematically, it is represented as:

$$\begin{aligned} &\min \quad p \\ s.t. \quad &E(\mathcal{T}) \geq R \end{aligned} \tag{4}$$

From Eq. 3, it can be observed that the expected number of completed tasks depends on both the task assignment probability p and the number of participating devices that successfully complete their assigned tasks M. Meanwhile, the number of participating devices that successfully complete their assigned tasks M is further dependent on the task assignment probability p. Intuitively, the larger the task assignment probability p, the more tasks to be executed on participating devices. Consequently, the number of participating devices that complete their tasks will become smaller due to dynamic flows of participating devices and their limited computational capabilities. In the following sections, we will first model the participating devices in the system and then introduce a task assignment mechanism that satisfies Eq. 4.

B. System Modeling

1) Modeling Participating Devices: The dynamics of participating devices comprise a fundamental factor that affects our design. In this paper, we mainly consider devices carried by pedestrians. This allows us to leverage the techniques proposed in [15] for accurate detection of transportation modes on participating devices.

To model the participating devices, we utilize the queueing theory to capture the dynamics of participating device flows. Because the arrival and departure of participating devices in the system normally follow the Poisson process in the long run [12],[1] we use λ and μ to represent the arrival density and departure density of the system. Specifically, by recording the number of participating devices entering the system within a pre-defined time window t_w, the task manager (TM) is able to estimate the arrival density λ. Similarly, through periodic beaconing [25], TM is also able to estimate the departure density μ.

With the estimated λ and μ, once a participating device joins the system, we can calculate the probability that the device stays in the system for more than a time duration D, which can be represented as $\int_D^\infty \mu e^{-\mu t} dt$. Furthermore, the number of devices that will stay for more than D in the system is:

$$M_D = \lambda \times t_w \times \int_D^\infty \mu e^{-\mu t} dt \tag{5}$$

M_D is essential for estimating the number of devices who complete their tasks under various system settings.

Another important property of the participating devices is the heterogeneity among their computational capabilities. Since different computational capacities yield different task execution times, we first establish a model that obtains the average task execution time.

[1]Note that there exist other distributions that may model arrival and departure of participating devices more accurately under certain scenarios. Due to space constraints, we focus on the Poisson distribution for generality in this paper.

In [6], [3], around 90% of smartphone subscribers choose smartphones from one of the five biggest OEMs, including Apple, Samsung, HTC, Motorola and LG. Each OEM provides a limited number of types of smartphons with different hardware. We profile the task execution time of each task on all major smartphones mentioned in [6]. The execution time of a task on a type i device is denoted as t_i, and thus the average execution time becomes:

$$t_a = \sum_{i=1}^{M} (p_i \times t_i) \tag{6}$$

where M is the number of types of participatory devices recorded during the previous time window t_w and p_i is the percentage of type i devices among all participating devices. For the clarity of presentation, we assume that all tasks consume the same execution time with the same CPU in this paper, however, the method can easily be extended to deal with different tasks.

The computational power provided by participatory devices can not always be fully utilized by REPC, because those devices may be running other applications at the same time, which may delay the execution of our assigned tasks and increase their average task execution times . In [11], Falaki*et al.* showed the statistical results of CPU usage of smartphones during a whole day. According to this CPU usage distribution, we can calculate the expectation of the available CPU usage for participatory devices, which is denoted as e_a. In Eq.6, the average task execution time is t_a when the CPU is fully utilized by REPC. Therefore, in consideration of the additional workload, the actual average computing time t_c for a single task can be obtained by:

$$t_c = \frac{t_a}{e_a} = \frac{\sum_{i=1}^{M} (p_i \times t_i)}{e_a}. \tag{7}$$

2) Modeling the Number of Devices Completes Assigned Tasks: As discussed in Section II-A, the number of participating devices that are able to complete their assigned tasks is negatively correlated to the number of tasks assigned to individual devices. In this section, we formally model this relationship between the number of devices that complete their assigned tasks $M(p)$ and the task assignment probability p.

For a device to complete its assigned tasks, the time duration in which it stays in the system should be longer than the time it requires to complete all assigned tasks. Consequently, assume there are n tasks, given a task assignment probability p, the duration D, during which a device stays in the system should satisfy the following equation:

$$D \geq npt_c \tag{8}$$

where npt_c is the expected computational time for np number of tasks at a participating device.

Consequently, the number of devices that complete their assigned tasks can be calculated based on Eq. 5 and Eq. 8:

$$\begin{aligned} M(p) &= \lambda \times t_w \times \int_{pnt_c}^\infty \mu e^{-\mu t} dt \\ &= \frac{\lambda t_w}{e^{\mu pnt_c}} \end{aligned} \tag{9}$$

From Eq. 9, we can see that $M(p)$ is a decreasing function with the task assignment probability p.

For the participatory computing, we should at least have one participating device that completes its assigned tasks, therefore $M(p)$ should satisfy the requirement $M(p) \geq 1$. As $M(p)$ is monotonically decreasing with the increase of p, we can derive the upper bound p_f of the feasible task assignment probability p as:

$$p \leq p_f = \frac{\ln(\lambda t_w)}{\mu n t_c} \tag{10}$$

C. Obtaining the Maximal System Capacity

In this section, we analyze the maximally achievable expected number of completed tasks within a task set $\mathcal{T}$ for a given set of participating devices. We term such maximally achievable expected number of completed tasks as the maximal system capacity. As shown in Eq. 3, the number of completed tasks is dependent on the task assignment probability p and the number of devices that complete their tasks. Furthermore in Section II-B1, we show the number of devices completing their tasks is further dependent on the task assignment probability p in Eq. 9. Consequently, the maximal system capacity is also a function of task assignment probability p. Formally, we represent the maximal system capacity as:

$$E_{p^*}(\mathcal{T}) = \max_{p}\{E_p(\mathcal{T})\} \tag{11}$$

where p^* is the corresponding task assignment probability that achieves the maximal system capacity.

In order to obtain the maximal system capacity, we first derive the relationship between $E_p(\mathcal{T})$ and task assignment probability p by substituting Eq. 9 with Eq. 3 and have:

$$E_p(\mathcal{T}) = n \times (1 - (1-p)^{\frac{\lambda t_w}{e^{\mu p n t_c}}}) \tag{12}$$

To find the maximal value of Eq. 12, we can take the derivative of $E_p(\mathcal{T})$ and have:

$$E'_p(\mathcal{T}) = n(1-p)^{\frac{\lambda t_w}{e^{\mu p n t_c}}} \frac{\lambda t_w}{e^{\mu p n t_c}} \left[\mu n t_c ln(1-p) + \frac{1}{1-p}\right] \tag{13}$$

As $n(1-p)^{\frac{\lambda t_w}{e^{\mu p n t_c}}} \frac{\lambda t_w}{e^{\mu p n t_c}}$ in Eq. 13 is always greater than zero, it is sufficient for us to investigate only the values of the remaining components of Eq. 13 to analyze the properties of Eq. 13. Specifically, let us defined $f(p)$ as:

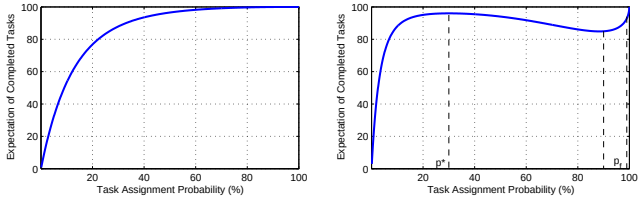
$$f(p) = \mu n t_c ln(1-p) + \frac{1}{1-p} \tag{14}$$

Then we can have the following Lemma for the optimal task assignment probability p^* that achieves the maximal system capacity.

Lemma II.1. *For any $p \in (0,1)$,*

- *If $\mu n t_c \leq e$, then $E_p(\mathcal{T})$ is an increasing function and the maximal system capacity is achieved at $p^* \to 1$;*
- *If $\mu n t_c > e$, $f(p) = 0$ has two roots p_1 and p_2. the maximal system capacity is achieved at $p^* = Max(E_{p_1}(\mathcal{T}), E_{p_2}(\mathcal{T})), E_{p_f}(\mathcal{T}))$, where p_f is the maximal feasible p value to ensure there is at least one participating device completes its task. At $p \in (0, Min(p_1,p_2))$ and $p \in (Max(p_1,p_2), p_f)$, $E_p(\mathcal{T})$ is an increasing function.*

The detailed proof of Lemma II.1 is omitted due to the space constraints.



(a) Case 1 (b) Case 2

Fig. 3: Typical Trends of $E_p(\mathcal{T})$

To further illustrate Lemma II.1, we use Fig. 3 to demonstrate the change of system capacity with varying task assignment probability p. Fig. 3(a) corresponds to the first case in Lemma II.1. With increasing task assignment probability p, the system capacity increases monotonically. Consequently, the maximal system capacity is achieved when $p^* \to 1$. The second case of Lemma II.1 corresponds to Fig. 3(b). As shown in Fig. 3(b), the system capacity first monotonically increases between $(0, p_1)$, then monotonically decreases between (p_1, p_2), and finally monotonically increases again at (p_2, p_f). The maximal system capacity is achieved at the maximum value between local maximum point p_1 and end point p_f.

D. Minimizing the Workload

After analyzing the maximal system capacity in Section II-C, in this section we discuss how to utilize the properties of the system capacity to minimize the number of assigned tasks on individual participating devices while satisfying the threshold R of completed tasks.

First, let us consider the case when the maximal system capacity $E_{p^*}(\mathcal{T}) \geq R$, which means the current participating devices are able to achieve the specified task completion threshold. If the maximal system capacity is achieved at the local maximum point $p^* = p_1$ in Lemma II.1 and $E_{p_1}(\mathcal{T}) \geq R$, as the system capacity (the expected number of completed tasks) is monotonically increasing from 0 to p^*, then the minimal number of assigned tasks at individual devices can be obtained by solving the equation in the range $p \in (0, p^*)$:

$$E_{p_{min}}(\mathcal{T}) = R \tag{15}$$

However, if the maximal system capacity is achieved at the end point p_f and the local maximal system capacity $E_{p_1}(\mathcal{T}) < R$, then the solution for Eq. 15 is within the range $p \in (p_2, p_f)$. As the value of $E_p(\mathcal{T})$ increases monotonically in $p \in (0, p^*)$ and $p \in (p_2, p_f)$, we can use efficient algorithms such as Newton's Method [20] to obtain the minimal workload P_{min} for individual participating devices.

Occasionally, the system may not be able to achieve the specified task completion threshold even it operates at the maximal system capacity, i.e., $E_{p^*}(\mathcal{T}) < R$. In this case, the application may reduce the number of total tasks in the task set $\mathcal{T}$, depending on specific applications, such that $E_{p^*}(\mathcal{T}) \geq R$, or the system should operate at the maximal capacity with task assignment probability of p^*.

E. Handling System Measurement Errors

In previous sections, we minimize the task assignment probability p by assuming all measurement results such as participating device arrival/departure density and average task processing time are accurate. However in practical system deployments, measurement noises and errors are inevitable. Therefore in this section we elaborate on how to handle system measurement errors in practice.

In our system, only three system parameters are estimated for the participating devices, including arrival density λ, departure density μ and average single task completion time t_c. All these parameters affect the expected number of completed tasks in Eq. 12, especially the part on the expected number of devices complete their tasks $\frac{\lambda t_w}{e^{\mu p n t_c}}$.

Assuming the maximal ratio of these measurement errors is α, then the ranges for arrival density λ, departure density μ and average single task completion time t_c is $((1-\alpha)\lambda, (1+\alpha)\lambda)$, $((1-\alpha)\mu, (1+\alpha)\mu)$ and $((1-\alpha)t_c, (1+\alpha)t_c)$, respectively. By investigating the equation $M(p) = \frac{\lambda t_w}{e^{\mu p n t_c}}$, we can have the following observations:

- If we fix the other parameters and vary λ, $M(p)$ decreases as long as λ decreases.
- If we fix the other parameters and vary μ, $M(p)$ decreases as long as μ increases.
- If we fix the other parameters and vary t_c, $M(p)$ decreases as long as t_c increases.

Therefore, when $\lambda = (1-\alpha)\lambda$, $\mu = (1+\alpha)\mu$ and $t_c = (1+\alpha)t_c$, $M(p)$ reaches its minimal value for any given task assignment probability p and the corresponding system capacity. Consequently we can solve the equation to handle system measurement errors:

$$n \times (1 - (1-p)^{\frac{(1-\alpha)\lambda t_w}{e^{(1+\alpha)^2 \mu p n t_c}}}) = R \tag{16}$$

to obtain the minimal task assignment probability p as in Section II-D.

F. Energy Overhead Analysis

Energy is one of the most precious resources for smartphones. In the following, we use a quantitative example to show that although additional computation and communication loads are introduced by REPC, current smartphones are well capable to handle the associated energy consumptions.

Let us take the Sumsung Nexus S with a $1,500\ mAh$ battery as an example. If the participating devices are only willing to contribute 1% of their capacities, an amount of $1,500 \times 1\% = 15\ mAh$ capacity is available for REPC. The current draw for the WiFi module's receiving state of Nexus S is about $117\ mA$ and that for its CPU processing with the peak frequency is $150\ mA$. As a result, the provided $15\ mAh$ capacity can support REPC to execute at its full frequency for least $\frac{15 \times 3600}{117+150} \approx 202\ s$. This time is far sufficient for REPC's time window (which is only about $15\ s$ according to our experiments, as will be explained in Section III), and thus alleviates our concern on the energy overhead.

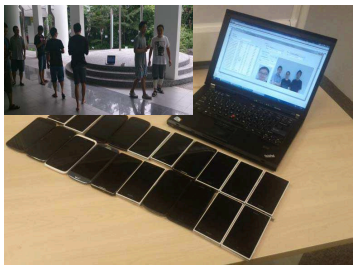


Fig. 4: Snapshots of Experiment

III. Implementation and Experiments

We implement a participatory computing system for neighborhood safety monitoring to demonstrate the effectiveness of the proposed task assignment framework. Specifically, our prototype system is designed for executing image processing tasks to find wanted criminals. In this application, the camera needs to find out whether any wanted criminal appears in its footage. Without sufficient computational capabilities and central servers, the camera recruits nearby participatory computing devices such as smartphones to help.

A. Experiment Setup

We set up a wireless network in an open area, which is composed of up to 20 Android smartphones and a laptop which can utilize computing resources from nearby smartphones.

The TM module is developed using Qt SDK [5] on a Linux laptop. We emulate the scenario that a surveillance camera periodically captures images that may contain the face pictures of wanted criminals. In order to avoid missing any suspects, the TM captures images in every second. A number of image matching tasks are generated and assigned to participating devices according to the task assignment scheme in our design.

We implement the task execution module on Android phones, including an image matching function using OpenCV4Android SDK [4]. Once a smartphone receives a task request, it performs image matching to check whether the query image matches any image in the local database, and returns the result to the surveillance camera. Table II lists the smartphone specifications and the average processing times to complete a single image matching task. Fig. 4 shows a snapshot of the user interface in the emulated surveillance camera side, and a snapshot of the experiment devices and scenario.

1) Evaluation Metrics:

- Task Assignment Probability p: it directly influences the number of tasks assigned to each device, and measures what percentage of tasks are assigned to an individual participating device. The number of tasks assigned to individual devices is desired to be small to reduces the device computation loads. During the experiment, we adopt the minimal p that guarantees the 90% task completion ratio as the first metric to evaluate REPC: a smaller p indicates a better system performance.
- The Total Number of Task Assignment at TM: it is measured as the total number of tasks assigned to

TABLE II: Smartphone Specification

Specifications / Model	Sony Xperia S	Galaxy Nexus	Moto Droid RAZR	HTC ONE X	Galaxy 3	LG Nexus4
CPU	Dual-core 1.5G	Dual-core 1.5G	Dual-core 1.2G	Quad-core 1.5G	Quad-core 1.4G	Quad-core 1.5G
Avg. Processing Time	0.91s	0.88s	0.96s	0.92s	0.87s	0.90s
Number	11	4	2	1	1	1

participating devices in one time window. Similarly we adopt the minimal number of totally assigned tasks that guarantees the 90% task completion ratio: the fewer the totally assigned tasks, the more efficient the system is.

- The Number of Unique Completed Tasks at TM: it measures the number of non-duplicate task completion results the surveillance camera receives from participating devices. This metric reflects whether REPC can guarantee a specified task completion threshold.

2) Baseline: Comparison is conducted with the an Oracle task assignment. In the Oracle design, the task assignment is obtained by identifying the minimal p that guarantees the 90% task completion ratio based on the known people moving profiles. Since the moving profiles are normally not available in reality, this baseline serves as an ideal task assignment scheme, thus the closeness to Oracle task assignment reflects the efficacy of REPC.

B. Experiment Results

1) Performance Overview: Fig. 5 presents the experiment results over time. Results demonstrate that REPC approaches approximately satisfies the 90 task completion ratio, while the task number generated in a time window is 100(the duration of a time window is 100s). Fig. 5(a) reports the task assignment probability. We can see that the average number is around 10%. Although the number of participating devices within the wireless access point's range is dynamic and unpredictable, REPC can smoothly handle the dynamics. As seen in Fig. 5(a), REPC can always find the task assignment probability p close to the Oracle p. In Fig. 5(c), around 90% tasks complete, which indicates that, through assigning appropriate workload to each participating smartphone, it is feasible to perform distributed computational tasks under the REPC framework. As shown in Fig. 5(b), the total number of assigned tasks fluctuates over the time window. We notice that the number of assigned tasks is around 1.5 times to the generated number of tasks in each time window and the number of completed tasks is around 1.1 times which is much fewer than the number of assigned tasks. This is partly because participating devices are dynamic and unpredictable, the REPC has to introduce certain redundancy to guarantee the task completion threshold. Introducing only about 10% redundancy can achieve 90% task completion ratio. Fig. 5(c) illustrates the task completion results. As Oracle task assignment has full knowledge, it clearly shows that Oracle can perfectly meet the task completion objective which is 0.9 in our experiment. We also observe that REPC can meet the task completion objective in almost all senarios. Note that the results in Fig. 5(b) does not necessarily fully reflect the task completion ratio in Fig. 5(c). The reason is that, TM module in REPC randomly assigns tasks to smartphones. But for each device, REPC cannot guarantee that the device can complete its assigned tasks and each task completes with different probabilities. Taking time window 14 for example, the total number of completed task in time window 14 is smaller than that in time window 13. However, in Fig. 5(c), we see that the completed task number is bigger in time window 14.

2) Results with Reduced Number of Participating Devices: In practical deployments, the density of participating devices within an area may be quite low and vary with time. In order to understand the performance with small number of participating devices, we conduct the experiments with 10 and 15 participating devices, respectively. The different brands of smartphones we use in this experiment are chosen from Table II. Fig. 6 shows the experimental results with reduced number of participating devices. Fig. 6(a) plots the task assignment probability changes over the time window. Compared to the results in Fig. 5(a), it clearly increases the average task assignment probability from 10% to 20% as the number of participating devices decreases. Comparing the two different settings in Fig. 6(a), REPC achieves a comparable performance to the Oracle. As shown in Fig. 6(b), when the number of participating devices decreases, REPC is required to assign more tasks to guarantee the task completion threshold. It results in a higher possibility that a smartphone moves out of the contact range of the surveillance camera before the assigned tasks are completed. It is interesting to see that when the number of participating devices decreases from 20 to 10, the average number of assigned tasks for each participating device increases. However, the number of completed tasks decreases significantly. This is because the workload is increased on an individual device thus leading to lower task completion probability. In Fig. 7, the Moto is assigned the largest number of tasks with 10 participating devices and completes the smallest number of tasks.

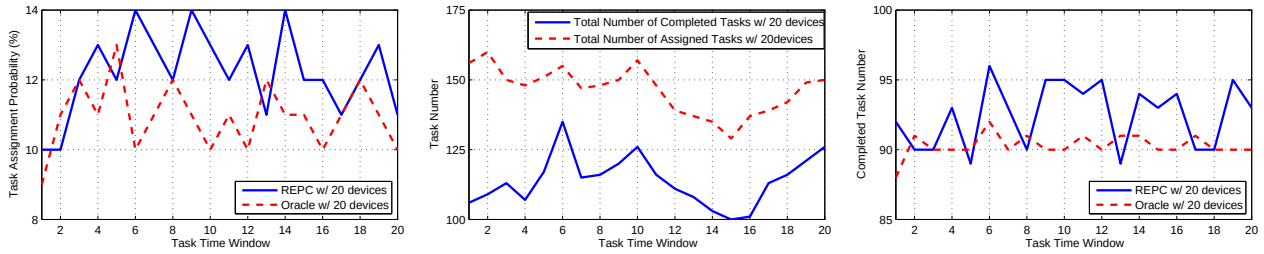
3) Impact of Diverse Computational Capabilities of Participating Devices: To explore the smartphone hardware-dependent issues and details captured in the experiments, in Fig. 7, we show the task execution differences among participating devices with different specifications. Because TM estimates the average computation capability among all participating devices within its coverage range, and makes the assignment without distinguishing individual devices, participating devices with different computational capabilities are supposed to be assigned with same amount of task. In Fig. 5 and Fig. 6, when REPC can meet the task completion objective with 20 and 15 devices, our estimation model can handle the variances of the computation capability well, which demonstrate the robustness of our system with heterogenous devices.

IV. Simulation Evaluations

In this section, we evaluate the performance of the proposed REPC through extensive simulations.

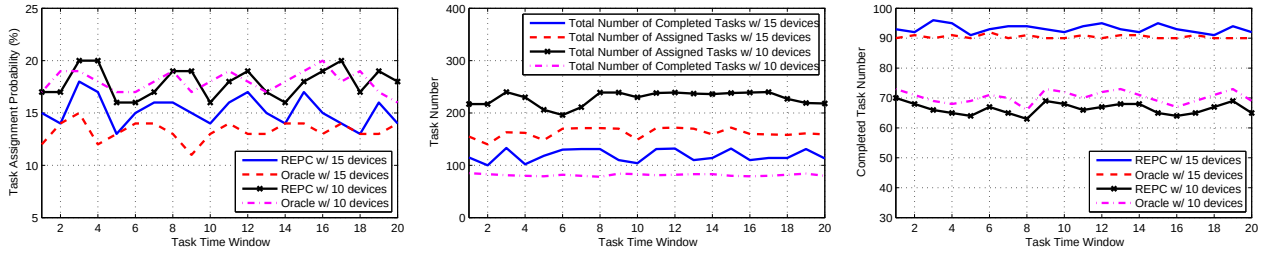
A. Simulation Setup

1) Simulation System: The simulated system is similar to that in the experiment. The TM divides and assigns tasks with



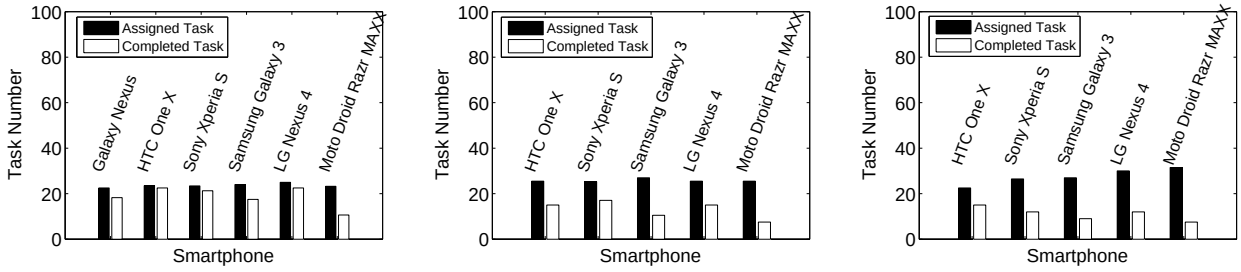
(a) Task Assignment Probability (b) Number of Assigned and Completed Tasks (c) Number of Unique Completed Tasks at TM

Fig. 5: Experimental Results With 20 Participating Devices



(a) Task Assignment Probability (b) Number of Assigned and Completed Tasks (c) Number of Unique Completed Tasks at TM

Fig. 6: Experimental Results With Reduced Number of Participating Devices



(a) 20 Participating Devices (b) 15 Participating Devices (c) 10 Participating Devices

Fig. 7: Impact of Computational Capabilities of Participating Devices

the time window based operation schema, and the participating devices move with randomly generated profiles. A ratio p of total tasks are assigned to each entrance device. These tasks are labeled as *completed* if the device accomplishes them before leaving the system, and *uncompleted* otherwise.

2) Default Settings: The following default settings are adopted in the simulation unless otherwise specified. We consider a square area of $400\ m \times 400\ m$ with totally 100 devices, and simulate a system operation time of $10,000\ s$. The moving velocity of devices is $1\ m/s$. The average computational capability of participating devices is 0.5 task/s and the workload on each device is randomly chosen from the CPU usage distribution presented in [11]. Located at the area center, the TM can detect the entrance and exit of participating devices with a radius of $150\ m$. We divide the entire simulated operation period into 100 windows, and thus each window has a length of $\frac{10,000}{100} = 100\ s$. The TM needs to assign 100 tasks during each window, and the ratio of completed tasks is required to be at least 90%.

3) Evaluation Metrics: Similar as in the experiment, we evaluate the performance of REPC with two metrics: the task assignment probability and the number of assigned tasks that guarantee the required task completion ratio. The resultant task completion ratios in the simulation are also investigated.

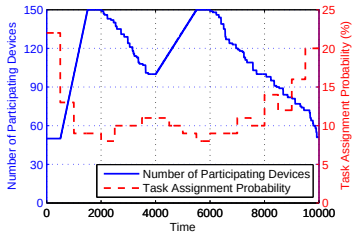
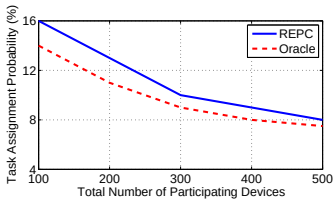


Fig. 12: System Performance Over Time.

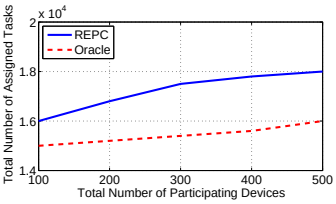
4) Baseline: Again, we compare REPC with an oracle task assignment method, which is obtained by identifying the smallest p satisfying the 90% task completion ratio based on the randomly generated device moving profiles.

B. Performance Comparison

1) System Performance Over Time: The number of participating devices is usually dynamic over time for practical

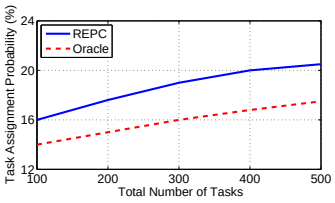


(a)

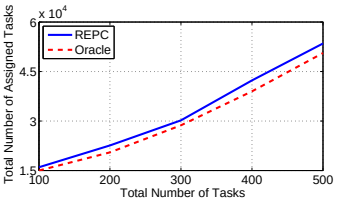


(b)

Fig. 8: Impact of Device Number

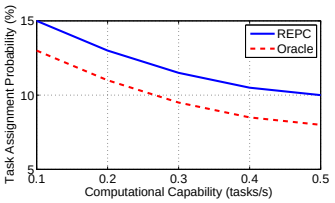


(a)

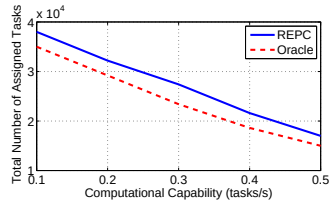


(b)

Fig. 9: Impact of Task Number

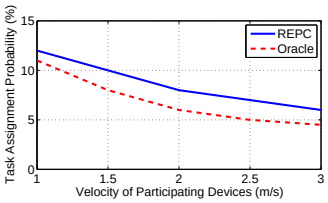


(a)

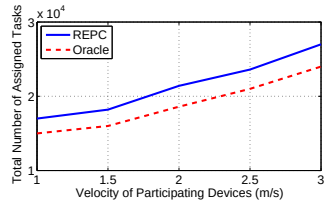


(b)

Fig. 10: Impact of Device Capability



(a)



(b)

Fig. 11: Impact of Device Velocity

systems. For instance, for systems deployed in the urban area, the number of participating deices may be large during rush hours, but it could be very small during night time. Therefore, we first evaluate whether REPC can adjust p adaptively according to the change in the number of participating devices, and the results are shown in Fig. 12. The solid curve represents the number of participating devices over the simulated time period, which varies from 50 to 150, and the values of p returned by REPC over the $10,000$ s are shown with the dot curve. We can see that REPC adaptively adjusts p according to the number of participating devices. Furthermore, we observe another interesting pattern that, while guaranteeing the required 90% task completion ratio, p changes in the opposite direction with the fluctuation of participating device numbers. For example, the number of devices increases from 50 to 150 during the time period from 500 s to $2,000$ s, while p decreases from 23% to 8%. During the period between $5,500$ s to $10,000$ s, the number of participators decreases from 150 to 50, and p increases from 8% to 20%. The reason is that when more devices are available to the system, each of them will be assigned a lighter computation loads, and vice versa.

2) Impact of Device Number: The number of participating devices is a critical factor affecting the performance of REPC, and thus in the following, we evaluate whether REPC can operate effectively with a wide range of participating device. The resultant values of p by REPC and the Oracle baseline are shown in Fig. 8(a), with the device numbers varying from 50 to 500. We can see that REPC operates with p values that are quite close to the Oracle baseline, which indicates a near-optimal performance. Another observation is that for both methods, the task assignment probability decreases as more devices are involved in the system. For example, the p values returned by REPC and the Oracle baseline are reduced from around 8% to 3.4% and 3.0% respectively, when the number of devices increases from 50 to 500. This means it is enough to assign only $100 \times 3.4\% = 3.4$ tasks to individual entering devices to guarantee the 90% task completion ratio.

The resultant total number of assigned tasks with both REPC and the Oracle baseline are shown in Fig. 8(b). Again, the proposed REPC performs quite closely to the Oracle baseline with all the explored cases. Furthermore, a decreasing trend of slope can be observed as more devices are involved in the system, which is a little unintuitive. This is because the amount of assigned tasks is jointly determined by both the device number and the number of tasks assigned to individual devices. When the former is increasing, the task assignment probability decreases (as shown in Fig. 8(a)), which in turn reduces the latter.

3) Impact of Task Number: Fig. 9(a) and Fig. 9(b) show the simulation results investigating the impact due to the variation of task number. In this experiment, the number of tasks is varied from 100 to 500. As seen in Fig. 9(a), the task assignment probability increases as more tasks are involved in the system. This is because when task number increases, the number of devices that can complete all assigned tasks decreases. Thus, in order to guarantee the task completion ratio, the task assignment probability increases. As observed in this figure, when both task number and task assignment probability increase, the number of assigned tasks significantly increases, which is shown in Fig. 9(b).

4) Impact of Device Computational Capability: Nowadays mobile devices have difference computational capabilities. To investigate the impact of different device capability on the performance of REPC, we perform the simulation with device capability varying from 0.1 tasks/s to 0.5 tasks/s, and the results are shown in Fig. 10(a). Intuitively, with a higher computational power and a fixed number of assigned tasks to individual devices, it is more likely for devices to accomplish (and return) the assigned tasks before leaving the system. This in turn indicates that we can assign fewer tasks to devices while still guaranteeing the required task completion ratio. This reasoning is verified in Fig. 10(a) and Fig. 10(a), where a clear decreasing trend for both the task assignment probability and the total number of assigned tasks can be observed. Again, the REPC achieves a quite close performance to the Oracle baseline: the task assignment probability and the total number of assigned tasks obtained by REPC are about 8% and 15% larger than that returned by the Oracle baseline, respectively.

5) Impact of Device Moving Velocity: In the next we investigate the impact of device moving velocity on the performance of REPC, and the resultant task assignment probability and

the total number of assigned tasks are shown in Fig. 11(a) and Fig. 11(b) respectively, with device moving velocities varying from $1\ m/s$ to $10\ m/s$. We can see that the task assignment probability decreases as the devices move faster. This is because that with a faster travel speed, devices enters (and leaves) the system more frequently, and thus from the perspective of TM, there will be more available devices during each time window. As a result, the number of tasks assigned to each devices is reduced. Essentially, this generates a similar effect on the performance of REPC as increasing the number of participating devices, as discussed in Fig. 8(a) and Fig. 8(b).

V. Related Work

Our work on participatory computing extends the volunteer computing paradigm to mobile devices, which allows people to donate the computation resource when their mobile devices are idle. BOINC is a representative volunteer computing middleware framework that has received considerable volunteers since 2002 [2]. The objective of BOINC is to make it possible for researchers to utilize processing power of personal computers around the world. In [8], DroidCluster system is proposed to build a local smartphone-based computing cluster. The authors in [8] show that it is highly feasible to perform distributed computational tasks on collaborative Android systems. As well as in [7], the authors developed a CWC infrastructure to leverage idle smartphones being charged overnight. This work suggests that smartphones are energy-efficient and cost-effective alternative for running tasks. The computing system Hyrax [18] has demonstrated the concept of allowing computing jobs executed on networked Android smartphones by porting Hadoop Apache to Android smartphones.

The most related work in the field of volunteer computing is [14], which investigates methods for computing low latency task batches in a pull-style: computing devices request tasks from the initiator. However, their main focus is how to control the reconnection rate of unreliable workers to satisfy the deadline requirement. To the best of our knowledge, our work is the first to minimize the workload of each device under the deadline constraint while considering both the mobility and computing ability of individual devices.

Most existing commercialized parallel computing models, such as MapReduce [9] and Hadoop [22], achieve fault tolerance by frequent checkpoints of completed tasks, which is the typical method for handling unreliable workers [13], [23]. However, implementing such technique is challenging in participatory computing due to random contact and short task deadlines. In this paper, different from the existing techniques which mainly focus on failure recovery, we propose a random task assignment mechanism that achieves fault tolerance by optimizing the number of assigned tasks at individual devices.

VI. Conclusion

In this paper we propose REPC, a novel random task assignment scheme for participatory computing. REPC is designed to achieve reliable and timely computations for participatory computing with close to optimal overhead. We have fully implemented the application of searching for wanted criminals with participatory computing on android platform with up to 20 mobile users. Both prototype experiment results and large-scale simulations demonstrate that REPC is able to achieve reliable participatory computing with very low system overhead in various system settings.

VII. Acknowledgments

This work was supported by iTrust IGDSi1301013, Singapore-MIT International Design Center IDG31000101, US National Science Foundation (NSF) grant CNS-1217791, Natural Science Foundation of China (NSFC) grant under contract No. 61303151 and 61331015.

References

[1] Abi research. http://www.abiresearch.com/.

[2] Berkeley open infrastructure for network computing (boinc). http://boinc.berkeley.edu/.

[3] comscore reports may 2013 u.s. smartphone subscriber market share. http://www.comscore.com.

[4] Opencv4android sdk. http://docs.opencv.org/.

[5] Qt sdk. http://qt.digia.com/Product/.

[6] Worldwide smartphone shipments top one billion units for the first time. https://www.idc.com/.

[7] M. Y. Arslan, I. Singh, S. Singh, H. V. Madhyastha, K. Sundaresan, and S. V. Krishnamurthy. Computing while charging: building a distributed computing infrastructure using smartphones. In *CoNext 2012*.

[8] F. Busching, S. Schildt, and L. Wolf. Droidcluster: Towards smartphone cluster computing – the streets are paved with potential computer clusters. In *Proc. ICDCSW '12*, pages 114–117, 2012.

[9] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. In *Proc. OSDI'04*, 2004.

[10] L. Deng and L. P. Cox. Livecompare: grocery bargain hunting through participatory sensing. In *Proc. HotMobile '09*, pages 1–6, 2009.

[11] H. Falaki, R. Mahajan, and D. Estrin. Systemsens: a tool for monitoring usage in smartphone research deployments. In *Workshop on MobiArch2011*.

[12] D. Gross. Fundamentals of Queueing Theory (4th ed.). *New Jersey: John Wiley & Sons*, page 232, 2008.

[13] Y. Gu, Z. Zhang, F. Ye, H. Yang, M. Kim, H. Lei, and Z. Liu. An empirical study of high availability in stream processing systems. In *Middleware'09*.

[14] E. M. Heien, D. P. Anderson, and K. Hagihara. Computing low latency batches with unreliable workers in volunteer computing environments. *Journal of Grid Computing*, 7.

[15] S. Hemminki, P. Nurmi, and S. Tarkoma. Accelerometer-based transportation mode detection on smartphones. In *SenSys*. ACM, 2013.

[16] S. Imai and C. A. Varela. Light-weight adaptive task offloading from smartphones to nearby computational resources. In *RACS*. ACM, 2011.

[17] P. G. Kannan, S. P. Venkatagiri, M. C. Chan, A. L. Ananda, and L.-S. Peh. Low cost crowd counting using audio tones. In *SenSys2012*.

[18] E. Marinelli. Hyrax: Cloud computing on mobile devices using mapreduce. In *Master's Thesis, Carnegie Mellon University*, 2009.

[19] D. Mendez, A. Perez, M. Labrador, and J. Marron. P-sense: A participatory sensing system for air pollution monitoring and control. In *Proc. PERCOM Workshops '11*.

[20] M. Minoux. *Mathematical programming: Theory and algorithms*. Wiley, 1986.

[21] M. Satyanarayanan. Mobile computing: the next decade. In *ACM SIGMOBILE Mobile Computing and Communications Review*, 2011.

[22] T. White. *Hadoop: The definitive guide*. Yahoo Press, 2010.

[23] Z. Zhang, Y. Gu, F. Ye, H. Yang, M. Kim, H. Lei, and Z. Liu. A hybrid approach to high availability in stream processing systems. In *ICDCS'10*.

[24] P. Zhou, Y. Zheng, and M. Li. How long to wait?: predicting bus arrival time with mobile phone based participatory sensing. In *MobiSys '12*.

[25] R. Zhou, Y. Xiong, G. Xing, L. Sun, and J. Ma. ZiFi: wireless LAN discovery via ZigBee interference signatures. In *Mobicom'10*, 2010.