Design and Architecture of Cloud-based Mobile Phone Sensing Middleware

Shunsuke Mori*§, Yu-Chih Wang†, Takaaki Umedu* ‡,
Akihito Hiromori* ‡, Hirozumi Yamaguchi* ‡ and Teruo Higashino* ‡
*Graduate School of Information Science and Technology, Osaka University
†Department of Computer Science National Tsing Hua University, Taiwan
‡Japan Science Technology and Agency, CREST
§Research Fellow of the Japan Society for the Promotion of Science

***Abstract*—**
Recently smartphones are in widespread use and they have large storage space and processing power. Thus, the smartphone-based networks with cloud server can be used as a cost-efficient sensing platform with high capable of processing complex, cooperative tasks just in time. However, low level implementation of cloud-based mobile phone applications needs a lot of human efforts, and has a considerable gap with high-level requirement given by application developers. To fill the gap, we propose a support middleware to execute cloud-based mobile sensing applications. Since we have proposed in our previous work, a language to describe high-level specification of cooperative applications on WSN, we extend the concept to manage and control multiple smartphones that participate in the system. We have shown some example descriptions of high-level specifications and have implemented the prototype system to confirm its usefulness.

I. Introduction

Recently mobile phones such as smartphones are in widespread use and have high functionality with multiple sensors. Therefore they can be used as sensing devices, using much richer storage space and processing power than networked sensor nodes in WSN, which are cheaper and simpler for massive deployment. Since their features enable to sense many types of data at various location wherever human can visit, useful information such as crowd of walking people, air condition and pollution in human-living area and real-time public transportation information can be obtained if a large number of mobile phones can participate in sensing activities. However, it is not an easy task to manage, organize and control a large number of mobile phones and a large volume of sensor readings to accomplish a given task. Although cloud-based solution is a reasonable option to store data, we still need software-support to accomplish such complex tasks that involve particular mobile phones at particular time and locations and to have those mobile phones under control. An example sensing scenario is a real-time public transportation location system. Bus passengers at a bus stop may want to know real-time location of the bus, and it can be estimated by the collective GPS traces of some passengers on the bus. To implement this, we need to identify mobile phone users on the bus by finding a set of GPS traces moving together along the bus route and stopping at bus stops. A naive approach is to collect all the traces from all the users, which is too unrealistic due to privacy concern. Therefore, we need a mechanism to send a request including time and location conditions toward mobile users to ask the corresponding users (i.e. bus passengers) to participate in this collaborative task and provide their GPS traces. However, few approach has been considered to achieve this requirement.

In this paper, we propose a middleware to support mobile phone cooperative sensing with a cloud server. Since we have designed in our previous work a methodology to support design and development of collaborative WSN applications [1], we use the basic high-level specification language specification part to describe the behavior of whole sensing system. However, it is very different from [1] in terms of the target architecture where we need to tackle (i) cloud-server architecture and (ii) mobility into consideration, while [1] assumes homogeneous, decentralized architecture without centralized servers. The middleware to achieve the mobile phone cooperative sensing consists of applications on mobile phones and the server-side module. Each mobile phone and the server communicate through WAN (e.g. 3G), and even two mobile phones through short-range communication such as Bluetooth or WiFi-direct.

Our method automatically translates the given sensing query into *server-side queries* which need to involve multiple mobile phones and *phone-side queries* which are executed by single mobile phones. We provide a concept that hides the details of network configuration, communication and processing inside the network but all the event occurrences are visible. The sensing query contains time, location and network-based constraints (conditions) and their processing. The process to achieve the given sensing query is very complex since it requires cooperation among mobile phones and servers. Thus, our method hides the physical placement of mobile phones and enables to execute cooperative sensing specified by abstract query descriptions. The proposed method reduces the effort to design and implementation of complex cooperation protocols by this developer-friendly form of behavior specifications format.

We provide a set of event sensing and communication

978-0-7695-4943-9/12 $26.00 © 2012 IEEE
DOI 10.1109/NCCA.2012.12

primitives to achieve the given sensing query in the networks. Especially, since the proposed method is extended for mobile phone sensing, we have designed interface and mechanisms to handle mobility and human-mediate processes. Mobility predicates enables to handle mobility conditions about velocities, trajectories and so on. Opt-in predicates enables to human-mediate sensing to ask owners to work for sensing. For example, an owner of mobile phone is required to take a video from the opt-in interface and he takes the video if he agrees with it.

The following simple crowd sensing example helps to understand the concept; each mobile phone sends beacon to each other and thus can detect neighbors. When a crowded situation is detected from the number of neighbors, the system reacts and starts sampling the neighbor count of the surroundings. Based on the sample readings, the system predicts the crowded area and informs to users. This system requires mobile phones collaboration to obtain samples from appropriate location at required intervals.The proposed scheme allows us to write the system in a simple form that consists of three steps, (i) start sampling on detection with required density and intervals, (ii) crowd prediction on obtaining enough samples and (iii) notification, without being aware of physical configuration of mobile phones and the server. We have shown some examples of mobile phone sensing system by our proposed method to show its usefulness. We have also demonstrated the performance of our proposed method in terms of successful data collection and generated packet to validate the quality of processing the given sensing query.

II. Related Work

Several approaches have been presented so far that support the entire process of design and development for WSNs [2], [3]. Liu et.al [2] have proposed a method to break a given single program down into several pieces that are executed by multiple nodes in ad-hoc networks. MacroLab [3] can also derive distributed codes from a given single program, and the developers can concentrate on designing policies to collect sensor readings and manipulate them. Ref. [4] has designed a framework in which a task mapping problem can be abstracted to mathematical formulations and tasks generated from the formulations are mapped to sensor nodes. However, the above approaches do not provide the concept of design support for cooperative event processing with time-, location- or network-dependent conditions. In this context, more relevant approaches with ours are Refs. [5], [6]. In particular, [6] proposes a set based programming approach where a query can be given by a set of nodes, a set of sensor values and even their combination. However, [6] basically adopts a node-centric view of programming, while we allow a node-independent approach where a specification can be fully-independent of nodes and networks including neighbors and sink nodes (our scheme allows higher abstraction in other words).

On the other hand, there are some approaches of mobile phone sensing for several purposes. For example, Ear-phone [7] proposes design of a system for noise mapping and a method to recover the noise map from incomplete and random samples obtaind by smartphones. CSN [8] is a classification system for human activity recognition by providing a unique classifier tuned for each user. The system exploits crowdsourcing to extract inter-person similarity. However, these researches are designed to support mobile phone sensing for particular purposes.

Some approaches are intended to support development of mobile phone sensing applications so that developers can develop such applications easier. Kobe [9] is a tool that aids mobile classifier development and provides a SQL-like interface for sensor data classification which can be used for mobile applications development. EEMSS [10] is an energy efficient mobile sensing system and provides a hierarchical sensor management scheme that specifies a particular sensing criteria and each user state in an XML-format description. Ref [11] proposes a cloud-based integrated framework for mobile phone sensing. This framework is designed as a part of cloud infrastructure, which coordinates a large number of mobile phone users and applications. Medusa [12] is a novel programming framework for crowd-sensing that manages not only computer resources but also human resources by auditing user acknowledgement and giving monetary incentives.

Compared with the above approaches, our contribution is to provide a middleware that supports higher level abstraction for mobile phone sensing. Query developers can specify time-, location- and topology-based conditions to choose a group of mobile node that should participate in the mobile phone sensing task. The developers are not necessary to be aware of physical locations of mobile phones and the middleware can find such a group that accomplishes the given query. In this context, we believe this has a novel concept of supporting distributed query executions.

III. Approach Overview

A. Middleware Architecture

In the proposed method, a mobile phone sensing system is defined to collect sensor data matched with given queries from mobile phones. Thus, this system should satisfy the following requirements.

- To reduce the complexity of sensor data and node management, the system enables to select nodes abstractly by conditions of time, location and sensor data attributes.
- To avoid large consumption of device energy and wireless bandwidth and concentration of traffic and process load, the system should communicate with fewer nodes while obtaining enough data. (Thus, processes which are executed by all nodes should be avoided.)
- To detect conditions concerning multiple nodes, the system executes collaborative processing by nodes and the server.

Fig.1 shows an image of the mobile phone sensing system by our proposed middleware for these requirements. This system is organized by the server on the cloud and multiple

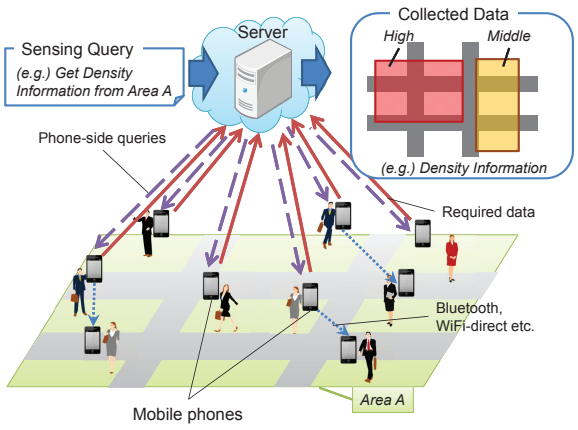

Fig. 1. Mobile Sensing System Architecture

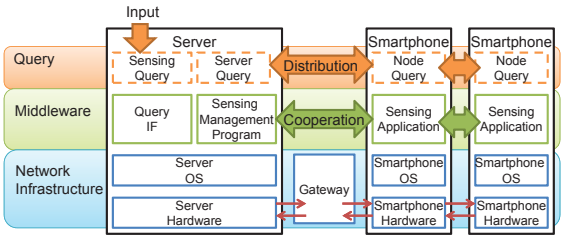

Fig. 2. Proposal Middleware architecture

mobile phones and it enables cooperative sensing by their collaboration. A requestor gives a query, which contains a abstracted requirement for mobile phone sensing such as getting density information of a certain area, to the server and the server distribute the query to mobile phones. Each mobile phone determines whether it participates the sensing or not by itself to reduce the probability to upload waste data. When it participates, it executes sensing and uploading data to the server. The server analyzes the data uploaded by mobile phones to extract information. In addition, sometimes it selects mobile phones based on the analyzed data to satisfy the more complex queries concerning multiple nodes. These processes satisfy the above requirement.

Fig.2 is the architecture of our proposed middleware to realize such systems. The middleware is composed of the program on the server and the application on each mobile phone. The collaboration between the mobile phones and the server is achieved by communication through WAN (e.g. 3G) and the collaboration among mobile phones is achieved by short-range communication such as Bluetooth or WiFi-direct. The server program provides functions for distributing queries, analyzing uploaded data, managing each mobile phone, and selecting some mobile phone to participate the more complex sensing based on analyzed data. The applications on each mobile phone has functions such as determinating to participate the sensing, sensing according to the query information, uploading the sensing data to the server. The functions provided by this middleware enable such mobile phone sensing.

```
nodegroup CrowdDetector
  condition:
      TestEach(neighborCount, ">10")
      && InFloatCircle(100)
      && Size(30, INFINITY)
  action:
      centroid = GetCentroid()

nodegroup SamplingSpot
  condition:
      InGeoCircle(CrowdDetector.centroid, 200)
  action:
      OutputData(
          GetSamplingDataSelect(
              neighborCount, 1min, 10min, 3))
```

Fig. 3. A Query Description of Crowded Sensing

B. Query Description Outline

Our method can execute sensing on each mobile phone (hereinafter called node) and the server for a given sensing query that describes actions to be taken by a group of nodes and conditions to be examined before the actions. Requestors can easily describe systems by specifying such conditions that should be checked by cooperation of nodes. We show a simple but essential example in Fig. 3 where a group of nodes that satisfy the following conditions is defined as the first group to detect a crowd; (1) all the nodes in the group have detected numbers of neighbors higher than 10, (2) they are located in a circle whose radius is 100m and whose center is one of given positions of intersections, and (3) the size of the group is more 30 nodes (we explain these predicates in Section IV, but readers may refer to Table I). In addition, in order to sample information of the crowd, the second, larger group of nodes that contains the previous group having the similar center with the previous group (*CrowdDetector*) but with larger radius 200m is defined. The nodes in the second group (*SamplingSpot*) sample and upload numbers of neighbor as crowd information. In this way, conditions on geometry, sensing data values and their manipulations can be written in our query description.

However, such a sensing query is not easy to satisfy since checking conditions and executing actions need cooperative operations among nodes. For example, in order to check a condition on sensing data, (i) a node group needs to be organized, (ii) the sensing data needs to be collected onto the server, and (iii) it needs to be checked if the condition is met or not. The action should be executed if the condition is satisfied, or the group is dismissed.

Thus, our method can automatically derive the single-node and multi-node queries which are processed by cooperative nodes and the server from given sensing queries. This hides the details of node behavior, which are often complex, from the developers. Therefore they can concentrate on system logic.

C. Distributed Execution on mobile phones and the Server

For a given sensing query described by a set of nodes with pre-conditions and post-actions, we classify the predicates that constitute the condition into two categories, single-node predicates and multi-node predicates. An example of single-node

predicate is *TestEach* that checks if variable on each node satisfies a given condition (see *TestEach*(neighborCount, "> 10") in Fig. 3). Meanwhile, both *InFloatCircle*(100) and *Size*(30, INFINITY) are multi-node predicates since they cannot be examined by single nodes. For example, *InFloatCircle(100)* needs distance calculation for every pair of nodes, meaning that it can be checked only when a group of nodes is given. Considering this fact, we take the following strategy; Firstly, from the given sensing query, the server generate single-node queries which contain information of single-node predicates in it and multi-node queries which contain information of multi-node predicates in it. Each node contacts to the server and get single-node queries periodically. We let each node periodically check single-node predicates, and let the node be a potential constitute of the group if it satisfies the conditions. If a node becomes a potential constitute of the group, it report to the server. The server constructs a node group based on these reports. Then the data values to check the multi-node predicates are collected to the server, and it checks if all the multi-node predicates are satisfied or not. If true, those nodes take actions as specified. Moreover, we allow describing conditions of groups that depend on some other groups. For example, the second group (*SamplingSpot*) in Fig. 3) is such a group that refers to the "center" of *CrowdDetector* as a part of its conditions. In this case, the centroid of coordinates of nodes in *CrowdDetector* has been calculated by the server to prepare for creation of *SamplingSpot*, and the information is distributed to potential constitutes of *SamplingSpot* (in this case, all the nodes). by the server in two ways. (1) polling by each node or (2) broadcasting by some nodes which are nearby the target location. broadcast

In summary, each node needs periodically to check the single-node conditions of each group, and then report the result and some data to the server. The server forms the nodes whose report is received into a potential group with nodes which also satisfy the same conditions. During the server collects all the data necessary to check multi-node conditions. Then it actually checks the conditions and executes the post-actions in cooperation with the nodes. During the process, it prepares and calculates the data for the other groups' conditions if any.

IV. Language and Algorithm Details

A. Query Description Language

A sensing query consists of two types of profiles, *node profiles* and *nodegroup profiles*.

The *node profiles* define the attributes of sensor nodes. For example, if a WSN consists of wireless sensor nodes and base stations, then we prepare two profiles that correspond to them. In their profiles, local variables (storing sensor data and so on) and methods they hold are defined. We omit example descriptions here because they just consist of definitions of variables and functions.

In *nodegroup profile*, each block of description starts with a keyword **nodegroup** (words highlighted by bold fonts are reserved words hereafter). Conceptually, this corresponds to a group of nodes that cooperatively execute tasks. Developers can define pre-conditions with **condition** keyword and post-actions with **action** keyword. The condition part must be a logical formula using predefined *single-node* and/or *multi-node* predicates, and the action part must be a list of functions (or procedures).

The example query description of a crowd sensing system shown in Sec.III-B (Fig. 3) is an example of formal description. *CrowdDetector* and *SamplingSpot* are nodegroup definitions blocks. In *CrowdDetector*, three predicates are speficied in the condition part. *TestEach* is a single-node predicate, and *InFloatCircle* and *Size* are multi-node predicates. As we explained in the previous section, groups may refer to other groups by directly specifying their group names. For example, *SamplingSpot* group refers to the *CrowdDetector* group. Since the node groups defined by *CrowdDetector* may not be unique (*i.e.* there may appear multiple groups), *CrowdDetector* is assumed to be the reference to the first-generated group in our language definition. The condition of *SamplingSpot* contains a single-node predicate *InGeoCircle* with 2 parameters. In the sensing description, the value of 1st paramter has been determined in *CrowdDetector* group by *GetCentroid* functions.

Tables I and II show the list of predicates and functions, respectively. As for the predicate table, we add how the predicates are examined in distributed environment in the last column. *Single* means it can be tested by each node independently (*i.e.* single-node predicates), while *Multi* means cooperation among nodes is necessary (*i.e.* multi-node predicates). For example, *InGeoCircle* can be examined by each node independently based on its own coordinates and the given center and radius information. On the other hand, *InFloatCircle* needs to know the coordinates of all the nodes in the group since it does not relate to the specific geographical area but to relative locations among nodes. These attributes will be used in the execution algorithm in the following section. Due to space limitation, we omit some of predicates and functions, and the complete list can be found in [13].

Additionally, the proposed method is designed for mobile phone sensing. Thus, we have designed some mobile-device-specific predicates. For example, *KeepUpWithCircle* is a mobility predicates which is evaluated based on the device's trajectory, and *AllowsToProvideVideo* is a opt-in predicates which requires the owner of the device to determine to take video for sensing through the GUI of the device. These type predicates are suitable for mobile phone sensing systems.

B. Mobile Phone Sensing Execution

In this subsection, we explain how to execute a given sensing query on each node and server. As shown in Section III, each node and server repeats the following step sequence; (i) query generation from given sensing queries, (ii) periodic polling to get single node query and notification, (iii) periodic sensing from sensors, (iv) periodic evaluation of single-node predicates, (v) data collection and potential group generation (vi) evaluation of multi-node predicates and (vii) execution of actions, to check if conditions are satisfied or not, and to execute actions if satisfied.

TABLE I
PREDICATES FOR CONDITION PART (EXCERPT)

Type	Predinate	Description	Examined by
General	TestEach (v, exp)	true *iff* variable v satisfies exp at every node	Single
General	DurationTestEach (v, exp, t)	true *iff* variable v satisfies exp at every node for the duration t	Single
Location	InGeoCircle (c,r)	true *iff* all the nodes in the group are within the circle centered at c with radius r	Single
Topology	InFloatCircle (d)	true *iff* all the nodes in the group are within a circle with diameter d	Multi
Location	InGeoRectangle ($c1$, $c2$)	true *iff* all the nodes in the group are within the rectangle determined by two coordinates $c1$ and $c2$	Single
Topology	InFloatRectangle (w, h)	true *iff* all the nodes in the group are within the rectangle with width w and height h	Multi
Location	InGeoStreet (c,r)	true *iff* all the nodes in the group are in a street at c and within distance r from c	Single
Topology	InFloatStreet (d)	true *iff* all the nodes in the group are in a street and within distance d	Multi
Topology	Size (min, max)	true *iff* the number of nodes in the group is in [min, max]	Multi
Mobility	KeepUpWithCircle (d, t)	true *iff* all the nodes in the group keep up within a circle with diameter d for t seconds	Multi
Mobility	IsFollowingPath (p, t)	true *iff* all the nodes in the group follow the path p in this t seconds	Single
Opt-In	AllowsToProvideVideo (m, c)	true *iff* owners of all the nodes in the group show the caption c and allow to take and upload a movie m	Single

TABLE II
FUNCTIONS FOR VALUES AND ACTIONS (EXCERPT)

Function	Description
Average(v)	Calculate the average of variables p among all the nodes in the group
AverageSelect(v, n)	Calculate the average of variables v among randomly-chosen n or more nodes in the group
GetCentroid()	Calculate the centroid of the coordinates of nodes in the group
GetDiameter()	Calculate the maximum distance between nodes in the group
GetVelocity()	Calculate the average velocity of all nodes in the group
GetTrajectory(t)	Calculate the the past t trajectory of the centroid of the coordinates of nodes in the group
Sleep(t)	sleep in t
OutputData(d)	Let the server output d
ExecuteEach(f)	Let each node execute function f
GetVelocity()	Calculate the velocity of the centroid of the coordinates of nodes in the group f
OutputSamplingData (d, i, t)	Let the server output d for t every i
GetSamplingDataSelect (d, i, t, n)	Let n nodes in the group upload d for t every i and let the server to output the uploaded data

(i) Query generation from given sensing queries: In this step, the developer give a sensing query of sensing to the server and the server generates a phone-side query and a server-side query. Each sensing query contains single-node predicates and multi-node predicates, respectively. Single-node predicates should be processed by each node and multi-node predicates should be processed by the server since it can collect and manage data of multi nodes. Thus, each sensing query is divided into a part of single-node predicates and that of multi-node predicates. The former becomes a phone-side query and the latter becomes a server-side query.

(ii) Periodic polling to get single node query and notification: Queries and notifications (discussed later) should be distributed to all nodes in the field. But broadcasting to all nodes causes concentrations of a large amount of data traffics on the server and frequency distribution causes a large amount of energy consumption on each node. Thus, the server distributes them by a polling strategy. Each node asks the server if there are phone-side queries and notifications for every interval T. The server sends them to the node if they are updated.

(iii) Periodic sensing from sensors: In this step, each node executes routine tasks like periodic reading from its sensors, which are used in the given single-node and server-side queries. Each node periodically measures data shown in the given sensing query as variables so that the code can refer to these data at steps (iv), (vi) and (vii). Especially, each node stores the history of its position since its trajectory may be required for single-node predicates.

(iv) Periodic evaluation of single-node predicates: In this step, each node checks if single-node predicates in each group are satisfied or not. Since each predicate can be examined by single node or multiple nodes as indicated in Table I, the code checks if only single-node predicates are satisfied or not. If necessary, the code manipulates local variables according to equations such as addition, subtraction and multiplication. If all other single-node predicates are satisfied but opt-in predicates are not, each node asks its owner to determine to work for sensing and they becomes true if the owner agree. If the predicates are met, the node continues executing the following steps since the node may be able to meet all the conditions specified in the group (this is checked later in step (vi)). Otherwise, the code goes to step (i) again. If all the predicates in the group condition are single-node predicates, our method skips the steps from (v) and (vi).

(v) Data collection and potential group generation: In this step, each node reports the result of checking the single-node predicates and its data for checking multi-node predicates. The server receives reports from nodes and organized the sender nodes as a potential node group after a certain interval from receiving the first report.

In addition, our proposed method also provides tree-based data collection protocol by using ad-hoc communication facility (like Zigbee or Bluetooth) for reduction of the communication cost of data uploading. Nodes which satisfy single-node predicates construct a tree and they collect data to root node through the tree. The root node uploads the collected data to the server. We can select direct uploading or tree-based uploading.

(vi) Evaluation of multi-node predicates: Once the server collects all data required for checking the rest of the predicates, it can know all the nodes which meet the condition and become true members of the group. In this step, multiple groups may be created according to a definition of a group because there are many combinations of nodes that can satisfy the given multi-node predicates. For example, if one of the predicates is *Size*(8, 10), at least three different groups with 8, 9 and 10 nodes can be considered. The server generates groups with the calculated average of *temperature* variables by using a pre-

defined function *GenerateSets* which derives all possible sets of nodes satisfying a given condition. Thus, the server sends a special packet to nodes, which do not satisfy the condition, to eliminates and also generates several sets of nodes that can meet the multi-node predicates from the rest of nodes. These derived sets become groups specified in the given sensing query.

(vii) Execution of actions: After organizing groups, the server executes actions in the given specification. Besides, if the group is accessed by another group, the server has to notify the values of variables to those nodes which need them. Therefore, it not only executes the actions but also notifies those data. This notification is performed in two ways. If the group of nodes which accesses those values are explicitly identified at that moment and if the locations of those nodes are known, the notification can be delivered to the location via geocasting from a certain node in the group to reduce redundant messages. Otherwise, the notification is distributed by polling of each node in step (ii).

V. Performance Evaluation

We first demonstrates the benefit from our method in terms of developers design effort-saving. This is done by introducing example systems of our proposed method. Then we measure the communication performance in order to show proposed middleware works well by simulations. In addition, to demonstrate our method is available in real environment and provides useful interface for developers, we performed experimentations in real environment.

A. System Examples

To show that our proposed method have a system as a development environment for mobile phone sensing, we introduce two systems.

The first one is simple and similar with the crowd detection and alert system in Figs.3, but it can present applicability of our method to various systems. It is a traffic jam monitoring system (Fig.4). If a traffic jam is occurred, some mobile phone user takes videos of surrounding situation of the jam. This system is useful to understand details of the traffic jam and to support driver's determination. If mobile phones are car-mounted mode and detect a traffic jam by detecting that their velocity are continuously low, then the mobile phones in the surroundings are organized to assign tasks to taking videos and upload them. Each node has a facility to do it, but we would like to limit the number of mobile phones to upload data to only 10 node in the group since duplication of video uploading means waste of computation and communication resources. There are two groups called *TrafficJamSpot* and *MonitoringStreet* which represent the first-detectors of traffic jam, and the group of mobile phones in its surrounding area.

Another example is a public transportation monitoring system. For a public transportation (*e.g.* bus), each of passengers who ride on it has its mobile phone. If the information of the transportation is required, the system on the server finds passengers on it and collects real-time position data from them. The sensing query is given in Fig. 5.

```
nodegroup TrafficJamSpot
  condition:
      InFloatStreet(500m)
      && TestEach(mode, "==CAR_MOUNTED")
      && DurationTestEach(velocity, "<20kmph", 3min)
  action:
      centroid = GetCentroid()

nodegroup MonitoringStreet
  condition:
      InGeoCircle(Initiator.centroid, 2km)
      && AllowsToProvideVideo
  action:
      OutputData(
          GetSamplingDataSelect(movie, 1min, 1min, 10))
```

Fig. 4. An Query Description of Traffic Jam Monitoring

```
nodegroup TransportationPassengers
  condition:
      IsFollowingPath(TRANSPOTATION_PATH, 10min)
      && KeepUpWithCircle(10m, 10min);
  action:
      OutputData(
          GetSamplingDataSelect(position, 1min, 10min, 2))
```

Fig. 5. An Query Description of Public Transportation Monitoring

TransportationPassengers is a group of nodes which are corresponding to passengers of transportation. The group provides real-time position data. We assume nodes, which follow the same path as the transportation and keep within a certain distance between each node, as passengers of the transportation. Thus, each node monitors its trajectory and, if it follows the path, report it to the server. After the server receives reports from some nodes, node groups whose nodes keep their distance are organized and 10 nodes in the group is selected to upload the real-time position to the server for 10 minutes.

These systems supported by our middleware are useful example of mobile phones sensing. These demonstration show broad utility of the middleware.

B. Performance Analysis of the Middleware

1) Performance Evaluation in Simulated Environments: We have conducted simulation experiments to observe that our middleware performs well. We have used the Scenargie network simulator [14] version 1.4 where IEEE802.11g have been used in the MAC and PHY layers as wireless wide area network communication and all nodes can connect with the server by this network. We have targeted the crowd detection system and the simulation was performed for 50 seconds. The size of the area was $200m \times 200m$ and there are 4 cross point (as shown in Fig.6). Nodes are moving at a constant speed 1 m/sec and a crowd detection event occurs in the intersection at (150m, 150m) after 20 seconds. Nodes nearby the event report it to the server and it sends request packet to nodes in the target area.

To present that the middleware can achieve reasonable performance levels, we have evaluated the following metrics.

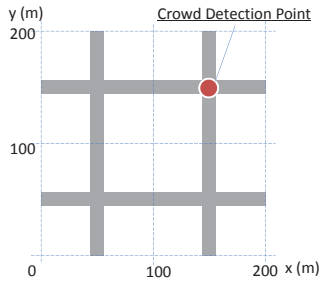

Fig. 6. Experiment Field

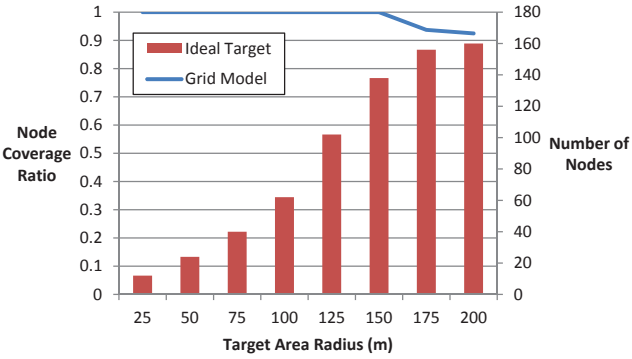

Fig. 7. Node Coverage Ratio

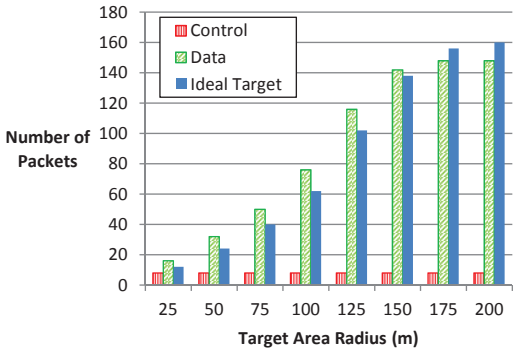

Fig. 8. The number of L2 packets

TABLE III
PERFORMANCE EVALUATION IN REAL ENVIRONMENT EXPERIMENTATION

R	20	30	40	50
Node Coverage Ratios	1.00	0.88	0.73	0.84
Delay (Sec)	13.963	17.812	21.443	20.598

TABLE IV
THE NUMBER OF GENERATED PACKETS IN REAL ENVIRONMENT EXPERIMENTATION

R	20	30	40	50
Control Packets	33	52	139	164
Data Packets	2	2	9	13

- *Node coverage ratio*, which is the ratio of the number of actually-found nodes in the simulation to the number of nodes to be found according to the query and node deployment. In other words, it shows the "completeness" of data collection.
- *Number of packets*, which is the total number of data and control packets in the network layer.

In order to verify the performance in various environments, we have prepared the scenario with 160 nodes which move along the streets on the field.

Fig.7 show the node coverage ratios. In these graphs, the number of nodes that are expected to be in the group is also shown as bars. We can see that the ratios are very close to 1.0 in all the cases. This shows a certain level of scalability to sense fields.

Finally, Fig. 8 shows the number of packets observed in the network layer and ideal nodes to send data packets. The number of packets grows as the radius of circles becomes larger and shows slightly larger than the number of ideal nodes in cases (less than "150 m") . However, the growing tread is similar to the number of ideal target nodes. From this fact, we can say that our mobile phone sensing middleware can prevent excessive traffic growth during the data collection phase.

The simulated evaluation shows the middleware works well in the ideal environment. Our ongoing works includes more realistic evaluation in various environments to show the utility.

2) Performance Evaluation in Real Environments: To show practical utility of our middleware for mobile phone sensing, we have implemented a prototype of mobile phone sensing system and have evaluated its data collection performance in real environment.

The prototype system is the crowd sensing system as shown in Fig.3 to detect human locations in a certain region. This system asks mobile phones in the region to report their locations by sending queries to them. We have conducted the experiments at a intersection in a campus of Osaka University (Fig.10). 12 mobiles phones are placed in the intersection and report their data by multi-hop forwarding if they are in a target circle. We have evaluated the performance of the system with several target circles, whose radii are 20m, 30m, 40m, and 50m. In our framework, it is easy to change such conditions since mobiles phones are controlled by queries and the queries are distributed to the mobile phones for each sensing.

The simulation results are shown in a GUI, which enables to monitor mobile phones and their locations easily and helps to manage and operate them (Fig.9). We have evaluated node coverage ratios and delays of location collection in the same way as Sec.V-B1. Table III shows the results. We can see that node coverage ratios are higher than 70% in all scenarios. We can see that the delay becomes large as R becomes large. This is because it takes more hops to deliver locations to the cloud server. Table IV shows the number of packets in the system. We can see that the number of control packets increase linearly. As shown in above experiments, our middleware supports not only mobile sensing itself but also analyzing the system performance. We believe that our approach can contribute to reduce whole cost of mobile sensing.

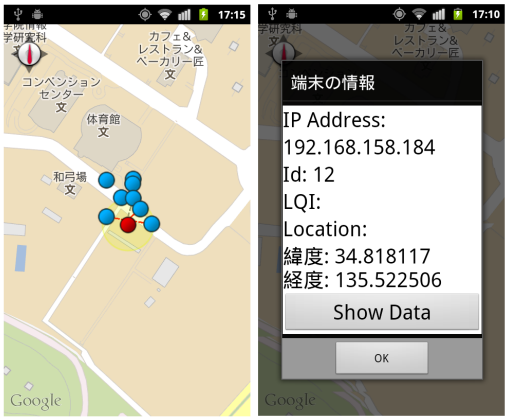

Fig. 9. Screen Shot of An Example Application on Android

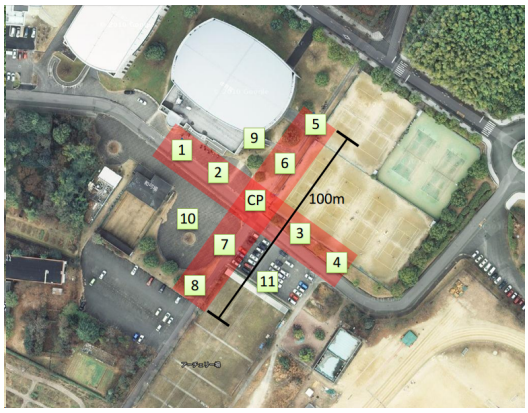

Fig. 10. Field of Experimentation

VI. Conclusion

In this paper, we propose a middleware to support mobile phone cooperative sensing with a cloud server. We have designed a language to describe high-level specification of such systems where we can specify the whole system's behavior from developer-friendly viewpoint based on group of node concept, and the middleware to achieve the mobile phone cooperative sensing consists of apps on mobile phones and the server-side module. Our method automatically translates the given sensing query into a sequence of queries which are executed by the server and mobile phones. We provide a set of event sensing and communication primitives to achieve the given specification in the networks since we have designed in our previous work, a methodology to support design and development of collaborative WSN applications [1]. However, it is very different from [1] in terms of the target architecture where we need to take (i) cloud-server architecture and (ii) mobility into consideration, while [1] assumes homogeneous, decentralized architecture without management by cloud-server. In this viewpoint, we believe this is the first approach to tackle such problems. We have shown some example descriptions of practical systems and have evaluated the quality of our proposed method in the experiments.

Our ongoing work includes applying the proposed method to various situations such as pedestrian crowds, car traffic, train passengers, and mixture of them. In those platforms, we need to consider mobility, neighbor discovery and security issues keeping the architecture limitation in mind. Thus, we have to provide various methods corresponding to various situations.

References

[1] S. Mori, T. Umedu, A. Hiromori, H. Yamaguchi, and T. Higashino, "Data-Centric Programming Environment for Cooperative Applications in WSN," submitted.

[2] H. Liu, T. Roeder, K. Walsh, R. Barr, and E. G. Sirer, "Design and implementation of a single system image operating system for ad hoc networks," in *Proceedings of the 3rd international conference on Mobile systems, applications, and services*, ser. MobiSys '05. New York, NY, USA: ACM, 2005, pp. 149–162.

[3] T. W. Hnat, T. I. Sookoor, P. Hooimeijer, W. Weimer, and K. Whitehouse, "Macrolab: a vector-based macroprogramming framework for cyber-physical systems," in *Proceedings of the 6th ACM conference on Embedded network sensor systems*, ser. SenSys '08. New York, NY, USA: ACM, 2008, pp. 225–238.

[4] A. Pathak and V. Prasanna, "Energy-efficient task mapping for data-driven sensor network macroprogramming," in *Distributed Computing in Sensor Systems*, ser. Lecture Notes in Computer Science, S. Nikoletseas, B. Chlebus, D. Johnson, and B. Krishnamachari, Eds. Springer Berlin / Heidelberg, 2008, vol. 5067, pp. 516–524.

[5] K. Whitehouse, C. Sharp, E. Brewer, and D. Culler, "Hood: a neighborhood abstraction for sensor networks," in *Proceedings of the 2nd international conference on Mobile systems, applications, and services*, ser. MobiSys '04. New York, NY, USA: ACM, 2004, pp. 99–110.

[6] M. Hossain, A. Alim Al Islam, M. Kulkarni, and V. Raghunathan, "μsetl: A set based programming abstraction for wireless sensor networks," in *Information Processing in Sensor Networks (IPSN), 2011 10th International Conference on*, april 2011, pp. 354 –365.

[7] R. K. Rana, C. T. Chou, S. S. Kanhere, N. Bulusu, and W. Hu, "Ear-phone: an end-to-end participatory urban noise mapping system," in *Proceedings of the 9th ACM/IEEE International Conference on Information Processing in Sensor Networks*, ser. IPSN '10. New York, NY, USA: ACM, 2010, pp. 105–116. [Online]. Available: http://doi.acm.org/10.1145/1791212.1791226

[8] N. D. Lane, Y. Xu, H. Lu, S. Hu, T. Choudhury, A. T. Campbell, and F. Zhao, "Enabling large-scale human activity inference on smartphones using community similarity networks (csn)," in *Proceedings of the 13th international conference on Ubiquitous computing*, ser. UbiComp '11. New York, NY, USA: ACM, 2011, pp. 355–364. [Online]. Available: http://doi.acm.org/10.1145/2030112.2030160

[9] D. Chu, N. D. Lane, T. T.-T. Lai, C. Pang, X. Meng, Q. Guo, F. Li, and F. Zhao, "Balancing energy, latency and accuracy for mobile sensor data classification," in *Proceedings of the 9th ACM Conference on Embedded Networked Sensor Systems*, ser. SenSys '11. New York, NY, USA: ACM, 2011, pp. 54–67. [Online]. Available: http://doi.acm.org/10.1145/2070942.2070949

[10] Y. Wang, J. Lin, M. Annavaram, Q. A. Jacobson, J. Hong, B. Krishnamachari, and N. Sadeh, "A framework of energy efficient mobile sensing for automatic user state recognition," in *Proceedings of the 7th international conference on Mobile systems, applications, and services*, ser. MobiSys '09. New York, NY, USA: ACM, 2009, pp. 179–192.

[11] R. Fakoor, M. Raj, A. Nazi, M. Di Francesco, and S. K. Das, "An integrated cloud-based framework for mobile phone sensing," in *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*, ser. MCC '12. New York, NY, USA: ACM, 2012, pp. 47–52. [Online]. Available: http://doi.acm.org/10.1145/2342509.2342520

[12] M.-R. Ra, B. Liu, T. F. La Porta, and R. Govindan, "Medusa: a programming framework for crowd-sensing applications," in *Proceedings of the 10th international conference on Mobile systems, applications, and services*, ser. MobiSys '12. New York, NY, USA: ACM, 2012, pp. 337–350.

[13] "D-sense web." http://www-higashi.ist.osaka-u.ac.jp/software/WSN/D-sense/.

[14] Space-Time Engineering, "Scenargie base simulator," http://www.spacetime-eng.com/.