Efficient Event Detection by Exploiting Crowds*

Ioannis Boutsis
Department of Informatics
Athens University of Economics and Business,
Athens, Greece
mpoutsis@aueb.gr

Vana Kalogeraki
Department of Informatics
Athens University of Economics and Business,
Athens, Greece
vana@aueb.gr

Dimitrios Gunopulos
Department of Informatics & Telecommunications
University of Athens
Greece
dg@di.uoa.gr

ABSTRACT

Encouraging users to participate in community-based sensing and collection for the purpose of identifying events of interest for the community has found important applications in the recent years in a wide variety of domains including entertainment, transportation and environmental monitoring. One important challenge in these settings is how significant events can be detected by exploiting the data sensed, gathered and shared by the crowd, while respecting the resource costs. In this paper we investigate the use of dynamic clustering and sampling techniques that allow us to significantly reduce utilization costs by clustering low-level streams of events based on their geo-spatial locations and then selectively retrieving the ones that depict the highest interest. Our experimental results illustrate that our approach is practical, efficient and depicts good performance.

Categories and Subject Descriptors

C.2.4 [**Distributed Systems**]

Keywords

Distributed Systems; Mobile Systems; Community-based Participatory Sensing; Sampling; Clustering; Event Detection

1. INTRODUCTION

"Community-based Participatory Sensing" systems where all the members contribute data to the system with the purpose of identifying events of interest for the community, are increasingly gaining popularity in recent years. The data are typically produced in the form of data streams and are generated on ubiquitous and portable devices, such as smartphones and tablets which are outfitted with a wide range of sensing capabilities, like GPS, WiFi, microphones, cameras and accelerometers. By combining data streams from different devices, important information and events of interest can be extracted such as congestion detection or real-time delay estimation as in the VTrack system [35]. Similar examples can be found in a number of domains including location-based services such as personalized weather information and for identifying areas of good WiFi connectivity [12], determining fuel efficient routes [13] and earthquake warning detection systems[27].

One important observation in these systems is that many people may gather in one place when an important event occurs. This happens in various situations such as social events (*i.e.,* conferences, concerts), traffic events (*i.e.,* traffic congestion) or emergency events (*i.e.,* earthquakes, floods). In these situations we are interested in identifying instantly that an event occurs, from the "gathering" of the people in a specific place, and then to analyze the event in more detail to define its nature.

Hence, one fundamental question is how to achieve efficient event detection by exploiting the data observed and provided by the crowd. This is a challenging problem considering the fact that mobile devices often produce more data than the network can deliver or the system can process, while only a subset of the data suffices to provide useful information about the events. In some cases, a small number of observations may yield confident event detection, while in other cases, a large number of observations are required to identify an event precisely.

Supporting efficient event detection in participatory sensing systems is a challenging process, as one needs also to consider the respective costs for the devices when producing and delivering the data, such as the energy consumption and the monetary cost (*i.e.,* 3g cost). It is vital that energy resources are used efficiently, especially when data stream sources are energy-constrained mobile devices and the amount of the users in participatory sensing systems depend on their costs. Thus, a fundamental question is how to define a suitable subset from the available data, to provide results of high quality, with small cost.

Our main idea is that the efficient identification of the important events where many participants are gathered can be achieved through clustering, and the selection of the most important data streams requires an efficient sampling ap-

*This research has been co-financed by the European Union (European Social Fund ESF) and Greek national funds through the Operational Program "Education and Lifelong Learning" of the National Strategic Reference Framework (NSRF) - Research Funding Program:Thalis-DISFER, Aristeia-MMD Investing in knowledge society through the European Social Fund, the FP7 INSIGHT project and the ERC IDEAS NGHCS project. We would also like to thank the anonymous reviewers and our shepherd, Prof. Mohamed Y. Eltabakh.

DEBS'13, June 29–July 3, 2013, Arlington, Texas, USA.

proach. Clustering[1, 4] and sampling[10, 5, 22] in large multidimensional datasets are two major data analysis tasks. However, there is inherent complexity of the clustering techniques, especially when we deal with stream data to detect events that can potentially evolve over time. Sampling on the other hand can reduce the size of the problem by selecting a subset of the data for processing, so that the available resources would be able to process the size of the sample. However, the sample needs to be decided with respect to the input data and the selection criteria. Thus, defining the optimal selection criteria to export a representative sample is a fundamental task when efficient event detection is required. Our goal is to use clustering as well as sampling techniques to select the stream data to be processed, that will enable us to identify core real-world events of interest over time in an accurate and efficient manner.

In this paper we present DENSE, a community-based participatory sensing system that aims to stimulate user participation by encouraging users to be members of the system as part of a dynamic group, so as to identify events of interest as they occur. Users participate in the community by sensing and sharing streams of data. Our technique uses the user's GPS readings to dynamically determine clusters that evolve over time, to identify the locations where important events take place. Then we perform sampling, by selecting a subset of the devices that participate on these clusters to retrieve their data streams. The sampling is based on the amount of data streams that the system can handle, and the selection aims to retrieve the most representative data streams in every cluster. This way we are able to extract important events, with minimal cost. We summarize our contributions below:

- We present **DENSE**, our system that exploits the data gathered and shared by members of the participatory sensing system to achieve efficient detection of events. Our focus is on social events such as concerts, theatrical performances, etc., that involve the gathering of people in dense spatial regions.
- We develop a dynamic clustering technique that allows us to cluster data streams generated from user mobile devices. Our technique extracts and updates the clusters dynamically using the stream data shared by the participants, to identify the locations where events of interest occur.
- We present a sampling scheme that selects k good representative streams with respect to the number, shape of the clusters and the data points distribution.
- We provide a detailed experimental evaluation of our approach on PlanetLab[30] using the T-Drive trajectory dataset[38, 37].

2. SYSTEM ARCHITECTURE AND MODEL

In this section we present the architecture and main components of our system, DENSE (Dynamic EveNt detection in participatory SEnsing Sytems). We then describe the system model and introduce our clustering and sampling model.

2.1 DENSE Architecture

DENSE is a wide-area stream processing middleware that comprises a set of distributed nodes, denoted as n_i, connected via virtual links, denoted as l_j. DENSE is built as an overlay on top of the Pastry peer-to-peer network and runs on Planetlab [30]. The goal of DENSE is to support the execution of distributed stream processing applications with QoS constraints, while efficiently managing the system resources. A distributed stream processing application is represented as a graph, where nodes represent the functions (that we call *event processing components*) and edges represent the data streaming between the different components based on the application logic. Upon submitting a user request, the system instantiates the appropriate components on the system nodes, in order to perform the processing, required by each application. Examples of application components are streaming components that are responsible to generate the stream data (*e.g.,* accelerometer sensor) or stream processing components (such as aggregation and projection components) that are used to process data streams from multiple mobile devices.

Figure 1 illustrates the DENSE architecture. In our previous work, we have implemented the following modules: i) A *discovery module* used for identifying application components and data streams in the system. ii) A *routing module* that is responsible for routing data streams as well as protocol messages between nodes. iii) A *monitoring module* for building and maintaining resource utilization profiles. iv) A *composition module* that selects and instantiates application components at runtime. v) A *resource management module* named RADAR [7], that attempts to manage the resources dynamically based on the applications' QoS demands and resource availability. We extend the DENSE middleware with: vi) A *clustering module*, which is responsible to determine the clusters dynamically, based on the GPS readings provided by the users. vii) A *k-sampling module*, that determines which of the available data streams will be processed. The sampling and clustering components work in concert, as we describe next.

In the DENSE system, we assume a spatial decomposition of the geographic area into a number of non-overlapping regions, similar to that of [34]. The organization of the area into regions can be done with respect to the size of the geographic area or the number of participants in the region, possibly defining several tiers at different levels of granularity, ranging from small local areas at the lowest tier, to the entire network area at the highest tier; this allows the system to collect streaming data from all users in a scalable manner.

2.2 System Model

In this paper we are interested in identifying events in "dense spatial regions"; these are events that involve the gathering of many people in one place considering the time and location dimensions, such as conferences, theatrical performances, festivals, media events, sporting events, traffic events, natural disasters (*e.g.,* floodings), etc. Our goal is to detect the exact location of such events and then to identify a representative subset of the data streams for further processing. This will allow us to considerably reduce the resource consumption when tracking how these events evolve over time. Note though, that, our technique is general and can be used in multiple application scenarios such as determining congested areas or car accidents, earthquake detection, etc.

In our system we exploit streams of data generated by application modules running on the smartphones, for each

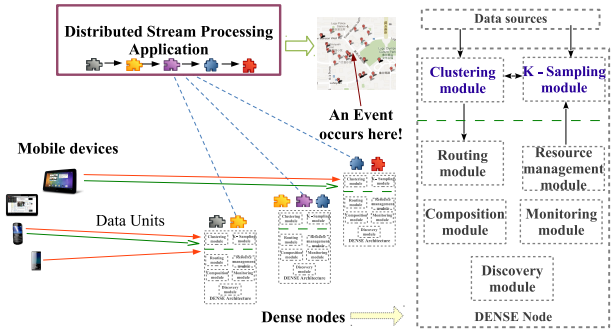

Figure 1: DENSE architecture.

$user_i \in U$. A stream of data consists of a sequence of individual chunks of data, called *Application Data Units* (ADUs); these are messages triggered locally at the phone using sensors present on mobile phones such as microphone, camera, GPS, accelerometer, motion sensors, etc., and their exact form is application dependent. An example of such ADUs is: <user_id, accelerometer data, microphone samples, timestamp, latitude, longitude> (for earthquake monitoring). The data units from multiple mobile sensors are streamed into the distributed stream processing system for further processing. The ADUs may vary in size since they may combine several types of data with different characteristics (*e.g.,* they might contain audio samples and accelerometer data for analyzing the congestion levels in a location) and so we must ensure that the amount of ADUs will not exceed the system's processing and communication resources. Thus, only a subset of the users that belong to the clusters, provided by the clustering component, are selected to provide their ADUs. Smartphones are powerful enough to do some local processing instead of sending the raw data streams. For example, in an earthquake monitoring application, the mobiles may process the data sensed from the accelerometer to define if they exceed an "alarm" threshold or to determine if their GPS location is similar to their previous location. The advantage is less communication overhead at the expense of higher processing overhead for the smartphones. However, local processing is application dependant, since it does not provide a benefit for every application (i.e. video processing on smartphones is prohibitive).

We note that the clustering and the sampling components are triggered in our system periodically, and the value of the time period is based on the events that we want to track. Thus, whenever the defined time period expires, both components are utilized to process the received GPS locations, extract the clusters and select the users to sample from.

The **clustering component**, implemented at the nodes of DENSE, will be using the data that include the GPS location of each $user_i$. In order to conserve energy and reduce the cost we state that users should not transmit their data units when their GPS location is identical to their previous one. Thus, our system uses a cache to preserve the users' previous locations, for some time. Nevertheless, after the defined time period has been exceeded and the user has not provided a new location, the system considers that the user has gone offline, so it stops using his last location when defining the clusters.

The **sampling component**, implemented at the source nodes of DENSE is responsible for obtaining the application data streams after deciding which of the mobile nodes will have their data streams processed. This component works in concert with the clustering and the rate allocation component to determine the users that will submit their data.

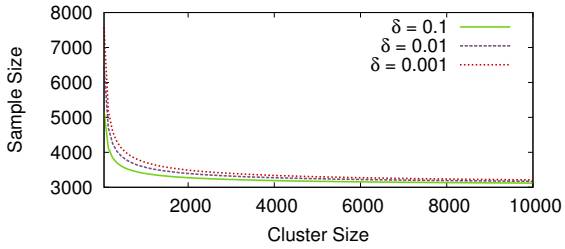

Figure 2: Required sample size to guarantee that a fraction of the cluster will be included in the sample.

2.3 How to Select Good Event Representatives

In this section we consider the implications of sampling for our technique. We note, that, random uniform sampling may not be sufficient in our system and we choose to implement and use a method for biased sampling. The problem with random sampling is two-fold: First, if applied in the original dataset with the purpose of speeding up the computation, the use of uniform random sampling may lead to loss of events. To illustrate this, consider the following situation:

Guha *et al.* in [14] present a formulation to link the sample size with the probability that a fraction of the cluster is included in the sample, based on Chernoff bounds. For a dataset D of size n, let u be a cluster of size $|u|$. They consider that a cluster u is included in the sample when more than $\varphi * |u|$ points of the cluster belong to the sample, with $0 \leq \varphi \leq 1$.

Figure 2 illustrates the sensivity of the sample size required by uniform random sampling to guarantee that u is included in the sample, with a probability no less than δ, $0 \leq \delta \leq 1$, for a database of $n = 10{,}000$ points. As can be observed, in order to guarantee with probability 99%, that 30% of the points of a cluster with 500 points will be included in the sample, we need to sample 35.3% of the database.

However, in [22] they show that using biased sampling we can get a sample that has the same probability to include points from a given cluster with smaller sample size than uniform random sample. Second, since our goal is to retrieve data from devices which are located in small but dense clusters, using uniform sampling we would have to include a large fraction of data items, and this would result in high resource consumption. Furthermore, as we show in our experimental evaluation, even if we consider only the subset of points that belong to the clusters as the dataset D, the use of random uniform sampling provides a worse than biased sampling selection of points, to analyze the events.

3. OUR PROPOSED APPROACH

In this section we illustrate how we employ clustering and sampling to provide efficient event detection.

3.1 Clustering

Clustering is the task of grouping a set of objects in such a way that objects in the same group, denoted as cluster, are more similar to each other, meaning that their attributes are closer in the multidimensional space, than to those in other groups. We take advantage of clustering to define the groups

that contain a large number of users in a geo-spatial location. The selection of the clustering approach plays an important role to the identification of the locations where important events take place. Several clustering techniques have been proposed in the literature [1, 32] such as the K-means[16] and K-medoids[29] algorithms. However these algorithms cannot be used in our setting since they typically perform better when the clusters are spherical, and are incapable to detect noise and outliers. Hierarchical clustering[19] has several advantages over K-means and K-medoids algorithms. It can be used to discover clusters of arbitrary shape and is insensitive to the size of the cluster. However, the runtime complexity of hierarchical clustering algorithms is quadratic.

Density-based clustering is a natural and attractive clustering approach for our situation since it can identify arbitrarily shaped clusters, it can handle noise and outliers, and one-pass algorithms that need to examine the raw data only once exist, thus the method's complexity is low. We note also that density corresponds well to human perception of clusters in the Eucledian space. Finally, density based clustering does not demand a prior knowledge of the amount of clusters k as the k-means algorithm does.

There are two well-known clustering algorithms for density-based clustering: (i) DBSCAN [32] and (ii) Optics [4]. DBSCAN is one of the most common clustering algorithms. In DBSCAN, the definition of a cluster is based on the notion of density reachability. The algorithm's main idea is to instantiate a cluster when at least a predefined number of points are reachable from a point p, meaning that these points are within a specific range from p. All of these points are integrated to the cluster, and the cluster is expanded recursively by examining which of these new points possess the same property, so that their reachable points will be included in the cluster. Optics is similar to DBSCAN, but it addresses one of DBSCAN's major weaknesses, that is the problem of detecting meaningful clusters in data of varying density. This is achieved by ordering (linearly) the points of the database so that that points which are spatially closest become neighbors in the ordering. Additionally, a special distance is stored for each point that represents the density that needs to be accepted for a cluster in order to have both points belong to the same cluster. Afterwards, the algorithm can export the clusters from the ordered list. However, none of these approach deals with mobile data where the clusters need to be managed and updated in real-time.

3.2 Dynamic Clustering

Assuming a data set D, containing a number of points (x_1, x_2, ..., x_m), the goal is to extract a number of non-overlapping subsets x_1, x_2, ..., x_n, with $n < m$, identified as *clusters*, whose points are close to each other in the multi-dimensional space, based on the clustering criterion.

Our approach for dynamic clustering shares the same logic with Optics, but is designed to extract efficiently clusters from stream data.

The clustering approach requires two parameters when processing a point: ϵ, that describes the maximum distance to consider, and $MinPts$, describing the minimum amount of points required to form a $cluster_c$. Thus, a point p is considered as a core point if at least $MinPts$ points are found within its $\epsilon - neighborhood$, $Nb_\epsilon(p)$. Similar to Optics we consider a core distance and a reachability distance variable for each point. Each point is assigned a core distance, core(p), that basically describes the distance to its farthest point among the closest $MinPts$ as:

$$\text{core}(p) = \begin{cases} \emptyset & \text{if } |Nb_\epsilon(p)| < MinPts \\ \text{distance to farthest MinPts} & \text{otherwise} \end{cases}$$

The reachability distance of a point p, reach(p), from another point o is defined as the distance between p and o, or the core distance of o:

$$\text{reach}(p,o) = \begin{cases} \emptyset & \text{if } |Nb_\epsilon(o)| < MinPts \\ \max(\text{core}(o), \text{distance}(o,p)) & \text{otherwise} \end{cases}$$

Intuitively, if o and p are nearest neighbors, this distance is used to examine if $\epsilon' < \epsilon$ to decide if o and p belong to the same cluster.

The clustering process works as shown in Algorithm 1. It traverses through not-processed points, and when a core point p ($core(p)! = \emptyset$) is found, and thus a cluster can be instantiated, it identifies the points within its $Nb_\epsilon(p)$, adds them in a Priority Queue and updates their distances on the defined order. Additionally, it also adds in the Priority Queue and updates the distances, of the points in $Nb_\epsilon(q)$, for every core point q within the $Nb_\epsilon(p)$ recursively. Thus, similar to Optics, the points are ordered based on their spatial distance. The goal of the algorithm is to extract the clusters from the ordered list by traversing through the points and checking the reachability distances among consecutive points. Thus we denote as a $cluster_c$, a group of consecutive points where their reachability distance does not exceed an application dependant predefined value, which plays a crucial role in the amount of extracted clusters [4]. In our experiments we denote that two consecutive points belong to the same cluster when their reachability distance is defined and less than ϵ. Thus, we iterate through points and define as noise the points whose $core(p) == \emptyset$ and $reach(p,o) == \emptyset$, we initiate a $cluster_c$ when p has the attribute $core(p)! = \emptyset$ and $reach(p,o) == \emptyset$ and add all the following points to the same cluster until we find a point with the attribute $reach(p,o) == \emptyset$. We note that, as it is obvious from the algorithm, each point p has a defined reachability variable, either if it has more than $MinPts$ points within its $Nb_\epsilon(p)$ (core point), or if it belongs to the $\epsilon - neighborhood$ of a core point.

Clustering Mobile Data. In our system all the mobile devices define their gps location by providing the following information : <user_id, timestamp, latitude, longitude>, where the *user_id* is generated through SHA-1 hashing, the timestamp represents the unix time when the location was retrieved, and the latitude, longitude represent the actual location. In this paper we assume that the noise in the readings of the sensors is negligible.

Periodically all participants examine if their gps locations differ from their previous geographical location and in that case they transmit their new location to the clustering component. Our clustering component receives the GPS readings, and updates the clusters, by using two functions, insert and remove. Thus, for each time period the clustering component: (i) Inserts newly arrived users. (ii) Removes and re-inserts the points for an updated user location. (iii) Removes the points that represent users that have not participated with a new location for a recent time period. Thus, we update the new distances and derive the corresponding

Algorithm 1 Clustering

```
ClusterData(ϵ, MinPts)
Initialize ordered list L
for (∀p ∈ DB if p not-processed) do
   N = points inside Nb_ϵ(p)
   Set p processed; L.add(p);
   Seeds = new Priority Queue
   if (core(p)! = ∅ ) then
      update (N, p , Seeds, ϵ, MinPts)
      for (q ∈ Seeds) do
         N' = points inside Nb_ϵ(q)
         Set q processed; L.add(q);
         if (core(q)! = ∅ ) then
            update (N', q , Seeds, ϵ, MinPts)

update(N, p, Seeds, ϵ, MinPts);
coredist = core(p);
for (∀o ∈ N) do
   if (o! = processed) then
      new-reach = max(coredist, distance(o, p))
      if (reach(o) == ∅) then
         o.reach = new-reach; Seeds.add(o, new-reach);
      else
         if (new-reach < reach(o)) then
            o.reach = new-reach ; Seeds.move_up(o, new-reach);
```

clusters (explained below). Finally the formed clusters, extracted for that time period, are provided to the sampling component to define the mobile devices that will provide the application specific ADUs through k-sampling.

Updating the Clusters. The advantage of our approach is that it modifies only the portions of the ordered list that needs to be updated when the data change. Thus, when the points of a cluster remain static, they do not need to be updated. However we can insert and remove points that might affect the existence, the shape or the size of the clusters, and thus we should provide an updated view over time.

The points that are processed can be assigned to three categories: (i) **Core points** which are the points that belong to a cluster, and there are more than $MinPts$ points found within their $Nb_\epsilon(p)$, (ii) **Reachable points** which are the points that belong to a cluster but there are less than $MinPts$ points found within their $Nb_\epsilon(p)$, and they are typically found in the boundaries of the cluster, and (iii) **Noisy points** which are the points that they do not belong to any cluster.

Changing one point can affect multiple neighbor points, meaning that their reachability or core distance should be changed. Since DENSE works with stream data, multiple points can be affected over time, when users update their gps location (points). In order to avoid redundant processing on the same points, when several points change in the same neighborhood we mark the changed and the affected points from the updates as not-processed and remove them from the ordered list L. At the end of the time period we execute our Clustering method that processes only the marked points and then we extract the updated clusters. We state that the ordered list of the algorithm is maintained through executions so that the data will be updated. The insert and remove functions are shown in Table 1.

Insert. When a user transmits a new GPS location, its current position has changed and thus the point p should be inserted in the dataset to compute new clusters, while its previous position (if exists) needs to be removed. Point p might be inserted in the spatial region where a cluster exists

Insert(p, ϵ, $MinPts$)	**Remove(p, ϵ, $MinPts$)**
N = points inside $Nb_\epsilon(p)$;	N = points inside $Nb_\epsilon(p)$;
for ($\forall l \in N$) **do**	**for** ($\forall l \in N$) **do**
Set l not-processed;	Set l not-processed;
if (l belongs to cluster c) **then**	**if** (p belongs to cluster c) **then**
Set all $n \in c$ as not-processed;	Set all $n \in c$ as not-processed;

Table 1: Insert, Remove

or a new cluster can be formed or it might be a noisy point. In all cases we have to update the distances of the points that belong to the $Nb_\epsilon(p)$. This is done by setting all the points in the $Nb_\epsilon(p)$ as not processed. However, in the case that one of the points in $Nb_\epsilon(p)$ belongs to a cluster c, we need to set as non-processed all the points of the cluster c as well. This happens because a Reachable point receives its reachability distance from the core point, and if this core point does not belong to the $Nb_\epsilon(p)$, to be re-processed, the Reachable point will be assigned with a wrong distance that may affect the extraction of the clusters. However, typically no more than one clusters will be found in the $Nb_\epsilon(p)$.

Remove. When a user has not transmit a new location for some time or has updated his/her current position, the previous position needs to be removed from the dataset. When we remove a point p in case where it is a core or a reachable point then the integrity of the cluster should be investigated, since it can can lead to the removal of some Reachable points from the cluster or to the division of the cluster, if less than $MinPts$ have left to the core points' range after the removal of point p. Thus, we need to set as not processed all the points that belong to the same cluster with p, if p belongs to a cluster and all the points in the $Nb_\epsilon(p)$. In the case that point p is a noisy point it can be easily removed since it does not affect any clusters.

Complexity. Our complexity is similar to Optics, that is, we also achieve a worst case complexity of $O(n \cdot \log n)$ for initializing the clusters. However, at runtime, typically less than n points need to be updated.

3.3 K-Sampling

Once the clusters are identified we select a subset of the mobile devices, to provide the system with the application specific ADUs that may contain data with different characteristics such as audio feeds, accelerometer data, etc. This will allow us to validate and track the event over time.

Suppose there are l data streams originating from mobile nodes in a specific region. The goal of the sampling component is to select a subset of the application data units k, (where $k < l$) to be processed. The sampling component consults the resource management component to determine the maximum amount of data units (k), that the system can efficiently process, depending on resource availability. These k data units should derive from users that belong to the clusters, formed by the clustering component, to analyze the events that takes place in these clusters. Thus, for each time period we select the k most representatives data streams, to analyze the identified clusters. However the selection of the k devices to retrieve their ADUs is not trivial.

First we note that the sample that we will receive from each cluster is proportional to the cluster's size and that at least one point will be selected for each cluster. This happens since when we are limited in resources we would like to extract more data from clusters with a lot of points, since they would represent more important events. Thus we select

from each $cluster_c$: $max(1, \frac{k}{\sum_{i=1}^{c} am_c} * am_c)$ points, where am_c represents the amount of points in cluster $cluster_c$ and k is the amount of points that the system can process, provided by the resource management component.

Although different techniques can be used to select a subset from each cluster, we aim to retrieve the most representative users. Each user typically represents other users within a range, since they will generate similar data. For instance, all the users driving on the same road will most likely have the same traffic speed, etc., and thus we only need to select one user to represent the others. Hence, our goal is to select the users that will represent the maximum amount of other users in the cluster, based on the distribution of the users and their local density. In order to export these users we define the function `Repr(point p, radius x)` that is able to identify the list of points that can be found within a radius x from point p, along with point p, $\texttt{Repr}(\texttt{p}, \texttt{x}) = \{q : \forall q \; s.t. \; distance(p, q) \leq x\}$.

Using Kernels for Density Estimation. Kernel density estimation is based on statistics and more specifically on the kernel theory [11, 36]. Kernel estimation is a generalization form of sampling, where all points have a weight of one but they distribute their weights in the space around them. A kernel function describes the form of this distribution. We choose to implement clustering and then sampling on the clusters based on the kernels, instead of just using the kernels to extract the sample for two reasons. First, the computational function for the kernels depends on the amount of points and thus computing the kernel density estimators for the whole database would increase the complexity compared to computing them only for a small number of points in the clusters. Second, using this technique for the whole database might abandon small clusters where the points are distributed in a sparser manner, although they involve events.

For a data set D, let (x_1, x_2, ..., x_n) be a set of tuples drawn from some distribution with an unknown density f. We are interested in estimating the shape of this function f. Its kernel density estimator is:

$$\hat{f}_h(x) = \frac{1}{n}\sum_{i=1}^{n} K_h(x - x_i) \quad = \frac{1}{nh}\sum_{i=1}^{n} K\Big(\frac{x - x_i}{h}\Big) \qquad (1)$$

where $K(\bullet)$ is the kernel, a symmetric but not necessarily positive function that integrates to one, and h > 0 is a smoothing parameter called the bandwidth. A kernel with subscript h is called the scaled kernel and defined as $Kh(x) = 1/h * K(x/h)$. It has been shown that the exact shape of the kernel function does not affect the approximation [11], and a polynomial or a Gaussian function can work well. Thus, for our experiments we choose to use the common Gaussian function as the kernel function:

$$K(u) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}u^2} \qquad (2)$$

However, the standard deviation of the function, that is the bandwidth, plays an important role. In our experiments we choose the bandwidths according to [33], where it has been shown that if Gaussian basis functions are used, and the underlying density being estimated is Gaussian then the optimal choice for h is:

$$h = \left(\frac{4\hat{\sigma}^5}{3n}\right)^{\frac{1}{5}} \approx 1.06\hat{\sigma}n^{-1/5} \qquad (3)$$

where $\hat{\sigma}$ is the standard deviation of the samples.

In our setting the values of x_i represent the 2-dimensional points $latitude_i$, $longitude_i$. Thus for the distance $x - x_i$ we use the Euclidean distance among 2-dimensional points:

$$dist(x - x_i) = \sqrt{(lat - lat_i)^2 + (long - long_i)^2}. \qquad (4)$$

The kernel density estimator metric for each of the points within a cluster, can be estimated, by combining 1,2,3,4 as:

$$\hat{f}_h(x) = \frac{1}{nh}\sum_{i=1}^{n} \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{dist(x-x_i)}{h}\right)^2} \qquad (5)$$

Algorithm 2 Selection of Representative Points (Radius x)

for ($\forall p \in cluster_c$) **do**
  Compute $\hat{f}_h(p)$ of point p;
  Add p in the ordered list OL based on $\hat{f}_h(p)$;
repeat
  Select point p with the highest $\hat{f}_h(p)$ in the OL;
  $rep_p = |\texttt{Repr}(\texttt{p}, \texttt{x})|$;
  Remove all points $q, q \in \texttt{Repr}(\texttt{p}, \texttt{x})$ from OL;
  Add p to SL;
  Update the $\hat{f}_h(p), \forall p \in OL$ and reorder OL;
until ($|SL| \geq \frac{k}{\sum_{i=1}^{c} am_c} * am_c$)
if ($|SL| < \frac{k}{\sum_{i=1}^{c} am_c} * am_c$) **then**
  Select $\frac{k}{\sum_{i=1}^{c} am_c} * am_c - |SL|$ points $\notin SL$;

Proposed k-Sampling technique. We propose a greedy algorithm to solve the maximization problem, based on the kernel density estimator. The goal of our approach is to greedily select the point that is able to represent the highest number of points, until the resource constraint will be fulfilled. The steps of our algorithm are shown in algorithm 2, and need to be executed for each cluster individually. We select a radius x to identify the range within which a point can be consider as representative for another point. In our experiments we set x dynamically as $\epsilon / \Big(2 * \frac{k}{\sum_{i=1}^{c} am_c} * am_c\Big)$. First, we estimate the density for every point p as $\hat{f}_h(p)$. That way we are able to identify the points that represent a lot of other points. Then, the points are inserted in an ordered list OL with respect to their kernel density estimator. While less than $\frac{k}{\sum_{i=1}^{c} am_c} * am_c$ points have been selected, we select point p with the highest kernel density estimator value from the ordered list OL and insert it in the list of selected points SL. Then, we remove all the points that belong in the list of `Repr(point p, radius x)` from the ordered list OL, since these points are represented by point p and we note the amount of the representative points rep_p to be used as a weight, as we will discuss later. Moreover, we update the kernel density estimator and the ordered list for the remaining points before continue with the loop, since the kernels might have changed after the removal of the points. For instance a point that was close but outside the radius of p will have a lower density estimator if a lot of neighboring points have been removed. If the algorithm is terminated and the amount of selected points in the list, $|SL|$, are less than $\frac{k}{\sum_{i=1}^{c} am_c} * am_c$ points, then we select $\frac{k}{\sum_{i=1}^{c} am_c} * am_c - |SL|$ points randomly from the points that have not been selected, $p \notin SL$.

After the points have been selected for each cluster we inform the devices that correspond to these points to provide their application-specific ADUs for the sampling. Each of the ADUs will be assigned with a weight, based on the

amount of the representative points for point p, denoted as rep_p and thus the processing of each ADU is considered as processing rep_p identical ADUs. Consider for example an application that aims to export the sound pollution within a cluster. Each ADU contains the estimated decibels collected by the mobile device. The average sound pollution in this application will be computed as: $pollution = (sound\_level_p * rep_p)/\sum rep_p$.

4. EXPERIMENTAL EVALUATION

4.1 Experimental Setup

We have implemented our techniques over the DENSE middleware and tested it on the PlanetLab [30] testbed. Our system was implemented in Java6 with approximately 5.7K lines of code.

Event Identification Application: The experimental evaluation scenario used was an application that is able to identify events that happen in a specific spatiotemporal region, such as concerts, sports events or traffic congestion in the city of Beijing in real-time. We used the T-Drive Trajectory Dataset [38, 37], where we extracted a one-week trajectories (from 2-2-2008 until 8-2-2008) that represent 10,357 taxis in the city of Beijing. The total number of points in this dataset is about 15 million and the total distance of the trajectories reaches 9 million kilometers.

The experimental evaluation focuses on the following parameters: (i) **Clustering and Sampling efficiency**, (ii) **Clustering and Sampling Latency**, (iii) **Comparison of our approach with Optics, D-Stream, DBSCAN, Uniform Sampling, in order to evaluate the benefit of different features of our approach**, (iv) **Benefit from the Online Clustering** and (v) **Energy savings**.

4.2 Operation of DENSE

In this section we demonstrate the operation of DENSE in identifying events in a real dataset and also illustrate its superiority over Uniform Sampling.

First, we illustrate the advantage of our approach, when using our dynamic density clustering to identify the events, before sampling is performed. In figure 3 we present the spatial distribution of all the taxis in the dataset, at 6:20 of 7/2/2008. As can be observed most of the points are located near the city center. Note that the latitude distance (39.6 - 40.6) is approximately 110 km and the longitude distance (115.5-117.5) is approximately 170km.

We have executed our online clustering approach for these points and set the parameters as follows: $MinPts = 30$ and $\epsilon = 0.01 \simeq 1km$, which means that we aim to identify clusters with core-points which have at least 30 neighbors inside a radius that is approximately 1km. This is presented in figure 4, where we extracted 24 clusters. Some of these clusters, like the large red cluster in the city center, can be ignored since they reflect events which are repeated every day in specific time periods, like the traffic in the city center. However, our approach is able to identify whether these clusters should be further explored, in case where the size of these clusters differs from the expected size for the specific time period. An advantage of our clustering technique, that can be shown in figure 4, is that it also considers points that are part of a more densely packed cluster individually. This can be easily observed by the small clusters near the large red cluster which should be treated independently, while other clustering techniques, such as K-Means, would consider all of them as one large cluster.

We state at this point that the selection of the two parameters $MinPts$ and ϵ plays an important role to the identification of the clusters and should be tuned according to the structure of the events that we want to identify. Thus if our goal is to find more dense clusters in an area, we could set a higher number to the $MinPts$ compared to the examined area. For instance we present the same snapshot with figure 5, when $MinPts$ is set to 60 and ϵ is set to 0.01.

In figure 6 we present a snapshot of uniform sampling that contains the same amount of data points as the ones that we extracted through clustering (approximately 35% of the total points). As can be seen the uniform sampling selects users for sampling that cover the whole area, including a lot of outliers. However, since our goal is to identify events in the specific area, if we use uniform sampling we will end up sampling from several users that do not provide any benefit, in terms of event analysis, and thus wasting a lot of resources, which are available for the k-sampling. On the other hand the use of the density clustering enables us to select data only from the users in the clusters, where events occur in order to analyze the events in more detail.

Event Identification: Our technique is based on our belief that when an interesting event occurs there would be a cluster from users that "gathered" at the event. We have validated this belief by identifying several concerts that actually took place in Beijing, retrieved from CHINA DAILY[1]. All the concerts that we examined formed clusters whose center is approximately the location where the event was hosted. For example we investigated a concert that took place in Beijing Concert Hall in Xicheng District on the Feb 7 2008 at 7:30 pm. This is illustrated in figure 4 by the light blue cluster whose center at 6.20 was: 39.90818171987097, 116.37933930159906. The small distance among the cluster's center and the Beijing Concert Hall (approximately 200m) is due to the Beihai Park, where no taxis can enter (and this can be found by the gap in the center of the big red cluster). However, if we analyzed other data as well, such as the direction of the users towards a point we could further increase our confidence about the location of the event. Nevertheless, as the times goes by the cluster's center moves constantly towards the actual location of the event.

The cluster that we study scaled in size over time and its maximum size was observed at 7.20. The observation of the cluster's size over time enables us to approximate the time interval when the event takes place. In most cases, when an event occurs, the size of the users gathering increases over time, until it reaches a peak and then it decreases, which enable us to estimate the actual time of the event as the peak of the cluster's size. Moreover, the retrieved data at that time point will be more accurate for the event. For example the center of the cluster will be much closer to the event's location than it will be an hour later.

4.3 Clustering

In this section we present the advantage of our dynamic clustering compared to the standard Optics algorithm for every iteration, and we present the variation of the clusters with regard to the used variables.

The benefit of our technique relies to the fact that we only need to re-process a subset of the data in order to identify

[1]http://www.chinadaily.com.cn

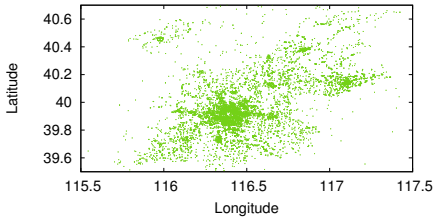

Figure 3: Snapshot of all data points at 6:20 of 7/2/2008

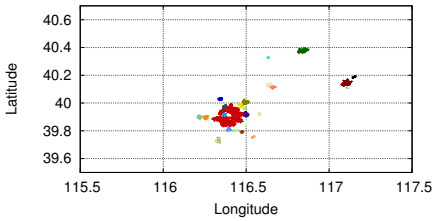

Figure 4: Detection of Clusters using DENSE

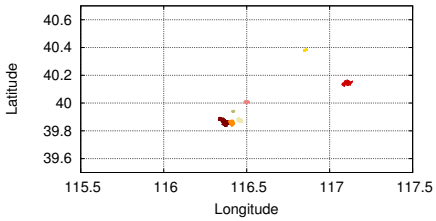

Figure 5: Detection of Clusters using DENSE with $MinPts = 60$

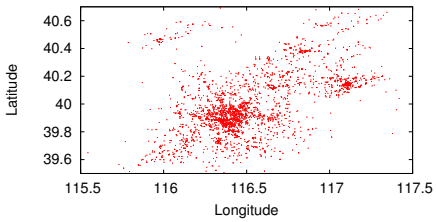

Figure 6: Output with Uniform Sampling

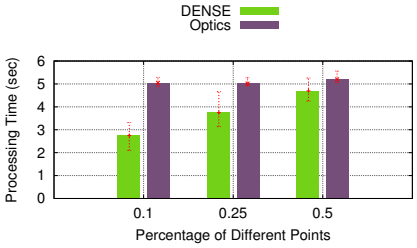

Figure 7: Overhead Comparison

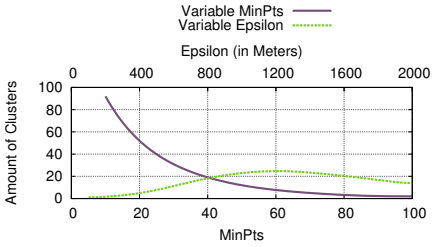

Figure 8: Clusters Variation under different MinPts, Epsilon

the clusters, but the result is identical to the original Optics algorithm. In figure 7 we compare the processing time needed for the clustering for both techniques with confidence intervals, to present the minimum and maximum processing times. The clustering involves the data received in the 7th of February 2008, and the time period to trigger the clustering algorithm is 20 minutes. This experiment was executed in a Intel Core i5 laptop with 4GB of RAM, which is a more controlled environment for the measurements. In our experiments, we use a large time interval as a period (20 minutes), and so the percentage of the taxis that change their geographical location among two sequential time periods is on average, approximately 50%. Note however, that when 50% of the points are updated the percentage of the points, affected due to their relationship is a lot higher, and our technique needs to update all the affected points as well. We compare the processing times using different percentages of points variation. More specifically we compare the techniques when the points that change their location during a time period are 10%, 25% and 50%. As can be seen from the figure DENSE has a great advantage over the standard Optics algorithm when the percentage of changing points is low. This happens since DENSE would not need to reprocess most of the points that remain the same (except from those which are affected from the changes). Thus, in a system with numerous points where only a small fraction changes over time (i.e. when a system utilizes static sensors along with the moving ones or when some cars remain static due to congestion or because they have parked) DENSE has a significant performance improvement. However, as the fraction of points that change over time increases the gap is decreased. Note that when all points change, DENSE will reflect an identical processing time with Optics, since all the points will be processed, as in the original implementation.

In figure 8 we present the variation of the amount of clusters under different values for the $MinPts$ and ϵ variables, for the data points at 6:20 of 7/2/2008, and we show why we set the variables as $MinPts$= 30points and $\epsilon \simeq$ 1km. Note that the x axis is different for each of the variables. We variate the $MinPts$ variable from 10 to 100, while the ϵ variable is set to 1km. It is obvious that when the $MinPts$ value is low a lot of clusters can be identified, however they will not be important events, if the $MinPts$ to form a cluster inside a radium of 1km is small. As the $MinPts$ variable increases the amount of identified clusters is reduced, since we will only identify denser clusters. When the $MinPts$ is set to a high value the clustering will identify only extremely dense clusters, which should be important, but other more sparse but important events will not be identified. Thus, we set $MinPts$ as 30, so that all important events will be identified, without needing to sample from unimportant clusters. Moreover, we variate ϵ from 100m to 2km when the $MinPts$ is set to 30. As can be observed from the figure, the amount of clusters increases when the ϵ value increases, until it reaches 1km. This happens since there are more clusters that can be identified with a specific $MinPts$ value when the radius is increased. However after ϵ has reached 1km it starts decreasing, since the range will be so large that some of the nearby clusters will be merged. However, this might prevent us from identifying events since we will consider two separate events as one. Hence, we have set ϵ as 1km. Nevertheless, the selection should be based on the specific setting and the events that should be identified (i.e. only extremely dense clusters should be considered when we need to identify a major concert).

4.4 Comparison

In this section, we compare DENSE with D-Stream [9], in terms of clustering quality. We choose D-Stream for the comparison since it is a well-known approach that has been proposed in the literature for dynamically clustering streams of data. Like our approach, D-Stream aims to cluster only the subset that changes over time in order to be able to cope with data streams. While our technique manages to achieve that through the ordered list, D-Stream uses grids to divide the spatial area and to avoid re-clustering in grids that have not been changed. The processing time in D-Stream depends on the size of the grids. Dividing the spatial area to smaller grids results on a highest processing time since more grids would typically change over time. However, the values that we set on our experiments for D-Stream produces a slightly faster -less than 5%- processing time than

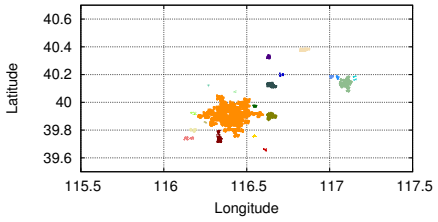

Figure 9: Detection of Clusters using D-Stream

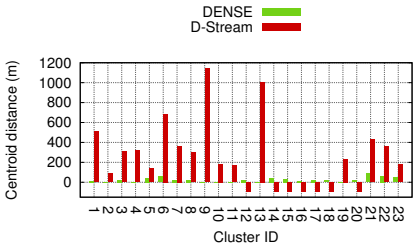

Figure 10: Comparison with DBSCAN

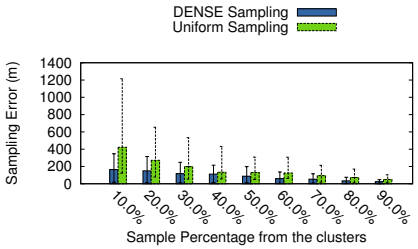

Figure 11: Sampling Error- Distance

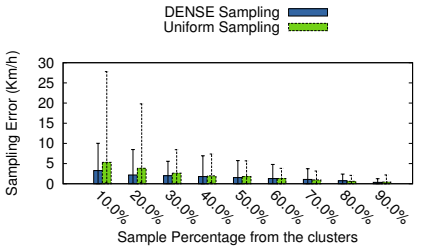

Figure 12: Sampling Error - Speed

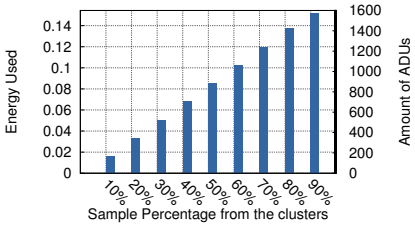

Figure 13: Energy Usage

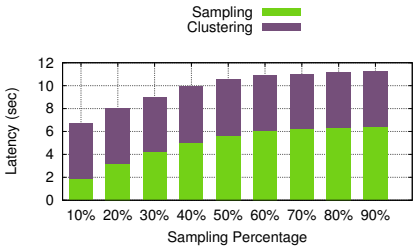

Figure 14: Latency

DENSE. D-Stream identifies dense and transient grids (transient grids have more points than the sparse grids but less than the dense ones), creates a cluster for each dense grid and merges iteratively the clusters of neighboring grids that are either dense or transient. Afterwards, it updates the clusters based on the evolution of the grids. In figure 9 we present the output of D-Stream for the same scenario (data/time) as we illustrated for our dynamic clustering approach. We have set the length of the grids to 0.02, similar to our ϵ radius variable. The disadvantage of D-Stream is that the grid's density represents the whole area of the grid, while in DENSE each point identifies the local density in the defined radius. Hence, a cluster that has its points in the borders of four grids might not be found. This squared structure inevitably lead us to use a smaller $MinPts$ value than in our approach, to be able to instantiate similar clusters. When we set large values for the $MinPts$ (close to 30) to extract the clusters with D-Stream, especially for the transient grids, it produced a lot of small clusters, since there were not enough neighbor grids to be merged with that amount of points. On the other hand, the use of small values led to a few enormously large clusters since several clusters found transient grids and merged together. Thus, we have tuned the minimum points for a transient grid to 12 and for a dense grid to 15 that provides almost the same amount of clusters with DENSE.

As can be observed from figure 9 their approach is able to identify most of the clusters. However, several clusters were merged together, since they were adjacent grids. Consider for example a left grid with a lot of points in its upper-left corner and a neighboring right grid with a lot of points in its lower-right corner. Although these points are far from each other the grids are going to be merged in D-Stream. Hence, this drawback of D-Stream caused the disappearance of the light blue cluster, that we had identified in our use case, and it proves that this technique cannot be used for event detection since a lot of events will be lost. Moreover, we can observe that the use of a smaller $MinPts$ value led to the instantiation of small clusters. However, we could easily prune these clusters when they contain less than 30 points.

In figure 10 we illustrate quality of the clustering for DENSE and D-Stream compared to DBSCAN which is a state of the art density algorithm. DBSCAN was selected for the comparison rather than Optics, so that the comparison would be fair, because DENSE shares the same logic with Optics. Thus, we provide the distance among the centroids of the DBSCAN's clusters compared to respective DENSE and D-Stream clusters. As can be seen from the figure DENSE manages to identify the same clusters with DBSCAN, with a small distance that ranges from 0 to 86m depending on the cluster. That proves that all of the clusters had almost the same points in the result set. On the other hand the D-Stream clustering failed to identify several clusters that DBSCAN found (presented as distance of -100m) and the identified clusters had a distance that ranged from 87 up to 1142 meters, compared to DBSCAN. Thus, we conclude that D-Stream fails to provide similar results to the well-known density clustering techniques.

4.5 Sampling

Finally we examine the effectiveness of our sampling approach. We present the accuracy of DENSE compared to the actual result when all the points from the clusters are used in the processing. Additionally, we compare our approach with performing random uniform sampling on the clusters. We choose uniform sampling for the comparison since it is a common technique for sampling.

We state that the sampling reflects the percentage of points that is used from the clusters. In our database, and for the specific settings we have defined ($MinPts$=30, $\epsilon \simeq$ 1km) the clusters (along with the ones that we filter due to repetition) involve approximately the 30% of the whole database. Thus, when we sample for example 10% of the clusters this is 3% of all the points in the specific time period.

Our goal in the specific experiment is (i) to identify the center of the cluster, when the event actually occurs, as we proved in the use case and (ii) to identify the average speed of the users within the cluster. Although we are not able to extract more interesting events due to the limited data in T-drive, in a real application where we can retrieve multiple types of ADUs from the selected users we could provide more interesting application specific events.

Error Metrics. Since our mechanism is based on sampling, it is expected that the results would have a deviation,

compared to the results we would get by processing all the ADUs in the given geographical area. In order to examine the effectiveness of DENSE we rely on two error metrics. First we use the Euclidean distance among the actual center of the cluster and the estimated one through sampling. Hence, we define the sampling error metric Δ_i as the expected absolute difference between the estimate location (lat'_i, lng'_i) and the exact location (lat_i, lng_i) of the cluster's center: $\Delta_i = \sqrt{(lat'_i - lat_i)^2 + (lng'_i - lng_i)^2}$. Moreover, when we compare the average speed among the actual and the one estimated from sampling we use the following error metric: $\Delta_i = \sqrt{(speed' - speed)^2}$.

In figures 11,12 we present the average Sampling Error of all the clusters, produced at 6:20 of 7/2/2008, to identify the center of the cluster and its average speed, along with the minimum and maximum error, for several sampling percentages. We also compare our k-sampling technique by sampling k points, that correspond to the sampling percentage, by performing random uniform sampling on the set of the clusters. Note that, the DENSE system has filtered the large red cluster due to its repetition.

As expected, the output is dependent to the amount of points that we sample, and thus when we use a higher sampling percentage the results are more accurate. However, we can observe that when we use 10% of the sample we get an average error of approximately 165m, and an error of 347m on the worst case. On the contrary the uniform sampling provides an average error of 422m for all the clusters. When the sample size increases to 40%, we can even identify the center of the cluster in less than 111 meters on average and within less than 215 meters in the worst case, presented by the confidence intervals, while the uniform sampling is able to identify the center of the cluster with an error of 133m on average and 431m in the worst case. Moreover, with a sampling percentage of 90% from the clusters, we can identify the events with an error which is less than 22 meters on average. As can be seen our technique outperforms random uniform sampling in all cases and it is able to provide accurate results even with a minimum sampling size.

Figure 12 presents a similar experiment with figure 11, but it illustrates the error of the average speed of the devices within the cluster. As can be observed DENSE can provide accurate results even with a minimum sampling size, since the average error is 3.25km/h when the sample size is 10% and it decreases as the sample size increases. Additionally, we illustrate that our technique outperforms random uniform sampling especially for a small sample, where the selection of the optimal points is very important. The difference among the two sampling techniques compared to the previous experiment is smaller, because the ADUs in the previous experiment depend on the location while in this experiment this is not always the case. For example typically all the taxis in the same geographical location will have a similar speed. However, some of them might have slowed down, stopped or parked.

Figure 13 illustrates the energy savings from the sampling for the same setting with the previous experiment. This derives from the amount of ADUs that were transmitted to the system for processing, since each ADU requires energy from the mobile device to send the ADU and the respective energy consumption for processing the ADU in our system. Thus we present the amount of transmitted ADUs for different sampling percentages and the percentage of the ADUs transmitted out of the total ADUs, since the sampling percentages refer to the amount of ADUs produced from nodes, within the clusters. Thus, for a sample of 10%, where we illustrated that the average error is approximately 165m from the center of the clusters and 3.25km/h from the actual speed in the clusters, we used only 168 points, which is only the 1.6% of all the points in the dataset for the specific timeperiod. Similarly the sample size of 40% is the 6.8% of the whole database with 706 points and the sample size of 90%, whose average error was less than 22 meters and 0.34km/h is approximately 12.7% of all the points. Thus, we conclude that using the clustering to determine the events can filter a lot of outliers and reduce the energy consumption.

Moreover, in figure 14 we present the latency for the clustering and the sampling, for the same experiment. We executed this experiment in a Intel Core i5 laptop with 4GB of RAM, that provides a controlled environment. As can be observed the clustering takes almost the same time, which is approximately 4.8 seconds, for all cases. The sampling time though depends on the sampling size since when it is increased, the algorithm needs to select more points, thus it traverses through the while-clause more times and the selection and the reassignment of the new kernel density estimators increases the processing time. However even for a sample size of 90% our approach takes almost 11.2 seconds, to select the optimal points of 10,357 points, that provides a high accuracy as we presented before. Nevertheless, the latency is low compared to the time interval we consider as period (20 minutes) and as we explained, a smaller time interval would provide lower processing times for the clustering and thus lower latency.

5. RELATED WORK

Participatory Sensing systems have recently become extremely popular for processing high-throughput, low-latency data streams and a number of systems have emerged in the literature [13], [35]. Hull *et al* in CarTel [17] propose a mobile sensor computing system for traffic monitoring. They use a query-oriented programming interface, to handle the data from the sensors, opposed to our stream processing architecture. Additionally, they suggest a "carry-and-forward" delay-tolerant network, which is opposed to our mechanisms that suggest a realtime processing logic.

Distributed stream processing systems have recently become extremely popular for processing high-throughput, low-latency data streams. A number of stream processing systems have emerged in the literature (including our own work on the Synergy middleware [31]). The research in this area is very rich and many papers have been published on detailed aspects of the technology such as data models, operators and query languages, resource management, scheduling, admission control policies, composition and placement algorithms, etc. Although, these research efforts have focused on high performance stream processing engines, our work focuses on the problem of sampling out of the available data streams to identify events of interest when the requested system capacity is incapable to handle all the data streams.

Clustering has been widely studied and many algorithms have been proposed. Several works aim to cluster moving objects that move along paths close to each other for a certain time [23, 6, 20, 18]. However, in our approach the users move towards an event from different places, so they have different directions and speeds. Moreover, we do not

need heavy-weight algorithms that constantly track the trajectories of the objects, but we only need to know the location of the active users for each time period. DBSCAN [32] is a well-known algorithm for density clustering, that identifies and clusters dense regions, separated by low density regions. OPTICS [4], that we extend in our approach, is able detect meaningful clusters in data of varying density, while DBSCAN cannot. Several approaches exist that deal with the clustering of data streams [1]. They aim to provide a good clustering using a small amount of memory and time but they do not consider density clustering. D-Stream [9] and DenStream[8], are techniques for density clustering over data streams. As we proved in the experimental section, our technique outperforms D-Stream. DenStream provides an approximation of the actual clusters, based on the DBSCAN, while our approach provides identical results with the OPTICS algorithm. Authors in [25] present SCUBA, that develops moving spatio-temporal clusters and perform intelligent load shedding for the data. As we discussed, moving clusters cannot be used in our approach to cluster the users of an event. Moreover, opposed to DENSE, they consider that every point belongs to a cluster, and they merge moving clusters with similar speed and direction even though their overlap can be only a few points. Finally, they use a grid structure for the clustering and they select the data tuples based on their distance to the centroid while DENSE selects the most representative ones. Authors in [26] propose EDACluster that is also based on grids to perform density clustering.

The research in the area involving the problem of sampling is very rich and several approaches have been proposed. In [24], the authors perform region sampling in sensor networks, to reduce the energy cost rate and use statistics to predict the optimal sampling plan. However DENSE is able to sample only specific regions where events occur. Al-Kateb *et al* in [2] propose an algorithm to extend the reservoir sampling, that selects a uniform random sample of a given size from an input stream of an unknown size, with an adaptive-size reservoir. However, our technique driven by the application logic, outperforms uniform random samples. Halkidi *et al* in [15] study the problem of online clustering, however the focus is on high dimensional sensor data. In [5] they propose a k-sampling technique that aims to select the most recent data, based on the timestamp. Stratified Sampling [10] is another well-known method for efficient sampling from a population, where the members of the population constitute homogeneous subgroups(stratums) and random sampling is performed for each stratum. However, we proved that DENSE outperforms random sampling.

A similar paper to our approach is [22] where the authors propose a technique for biased sampling, where the probability that a given point will be included in the sample depends on the local density of the data set. Our technique differs since we sample only within the clusters. Their approach might abandon clusters of lower density completely, which can lead to unidentified events. Authors in [21] propose Watchdog, an event detection framework that aims to cluster the right sensors to meet user specified detection accuracy during runtime. In order they determine the accuracy detection they generate clusters of each possible size, and examine them using Hidden Markov Models. This approach cannot be used in stream data and when the data size is too large, since the clustering would need a lot of time. Palmer and Faloutsos in [28] propose an algorithm to sample for clusters, using density information. Their approach works under the assumption that clusters have a zipfian distribution and is designed to identify clusters when they differ a lot in size and density, and there is no noise. Although their approach is an one-pass algorithm it provides an approximation and it is not adaptive, thus, it has to be executed even if no points have changed.

Several event detection approaches have been proposed in the literature to address in-situ event identification, especially for wireless sensor networks that emphasize in the energy savings. Regions of similar sensor data are detected in [34]. Although they use Kernel Density Estimators to identify similar sensor readings this cannot be extended for Participatory Sensing Systems where the mobile devices might produce different readings for the same event (*e.g.*, speed).

Authors in [3] focus on finding events described by a query. This is a complementary approach to ours; we take a more exploratory approach since: (i) they do not focus on the detection of the phenomena (events), but they assume that the description of the phenomena is given, (ii) the selection of the tuples is based on the distance from the phenomenon and thus all the tuples in the specific region will be returned, while we choose good representative users to provide data, (iii) they use different approaches to solve the problem thus they do not provide any clustering or sampling techniques.

6. CONCLUSIONS

In this paper, we have presented DENSE, a system that aims to improve user participation in community-based participatory sensing systems. DENSE makes it easy for users to sense, collect and share data units which are used to identify real life events when they occur. We propose online techniques to cluster the user data and select only a subset of users in these clusters for sampling to identify and track the events over time. The advantage of our technique is that it filters the noisy points through clustering, based on the application logic, and that it considers the data streams that depict the highest interest when selecting the data sample. Detailed experimental results illustrate that our approach is practical, efficient, depicts good performance and is able to provide accurate results with a relatively small sample size.

For our future work we plan to extend our approach to examine the capabilities of the devices in real-time, in terms of availability, battery levels etc., to include the resource capabilities of the devices when we decide which ones should be selected for the sampling.

7. REFERENCES

[1] C. C. Aggarwal, J. Han, J. Wang, and P. S. Yu. A framework for clustering evolving data streams. In *VLDB*, Berlin, Germany, Sep 2003.

[2] M. Al-Kateb, B. S. Lee, and X. S. Wang. Adaptive-size reservoir sampling over data streams. In *SSDBM*, Banff, Canada, July 2007.

[3] M. H. Ali, M. F. Mokbel, and W. G. Aref. Phenomenon-aware stream query processing. In *MDM*, pages 8–15, Mannheim, Germany, May 2007.

[4] M. Ankerst, M. M. Breunig, H.-P. Kriegel, and J. Sander. Optics: ordering points to identify the clustering structure. In *SIGMOD*, Philadelphia, PA, June 1999.

[5] B. Babcock, M. Datar, and R. Motwani. Sampling from a moving window over streaming data. In *SODA*, San Francisco, CA, January 2002.

[6] M. Benkert, J. Gudmundsson, F. Hübner, and T. Wolle. Reporting flock patterns. *Computational Geometry*, 41(3):111–125, 2008.

[7] I. Boutsis and V. Kalogeraki. Radar: Adaptive rate allocation in distributed stream processing systems under bursty workloads. In *SRDS*, Irvine, CA, October 2012.

[8] F. Cao, M. Ester, W. Qian, and A. Zhou. Density-based clustering over an evolving data stream with noise. In *SIAM*, Bethesda, MD, April 2006.

[9] Y. Chen and L. Tu. Density-based clustering for real-time stream data. In *KDD*, San Jose, CA, Aug 2007.

[10] W. G. Cochran. *Sampling Techniques, 3rd Edition*. John Wiley, 1977.

[11] N. Cressie. *Statistics for spatial data*. Wiley & Sons, 1993.

[12] A. Dou, V. Kalogeraki, D. Gunopulos, T. Mielikinen, V. Tuulos, S. Foley, and C. Yu. Data clustering on a network of mobile smartphones. In *SAINT*, Munich, Germany, July 2011.

[13] R. K. Ganti, N. Pham, H. Ahmadi, S. Nangia, and T. F. Abdelzaher. Greengps: a participatory sensing fuel-efficient maps application. In *MobiSys*, San Francisco, California, USA, June 2010.

[14] S. Guha, R. Rastogi, and K. Shim. Cure: an efficient clustering algorithm for large databases. In *SIGMOD*, pages 73–84, Seattle, Washington, USA, June 1998.

[15] M. Halkidi, V. Kalogeraki, D. Gunopulos, D. Papadopoulos, D. Zeinalipour-Yazti, and M. Vlachos. Efficient online state tracking using sensor networks. In *MDM*, Nara, Japan, May 2006.

[16] J. A. Hartigan and M. A. Wong. Algorithm AS 136: A K-Means Clustering Algorithm. *Applied Statistics*, 28(1):100–108, 1979.

[17] B. Hull, V. Bychkovsky, Y. Zhang, K. Chen, M. Goraczko, A. Miu, E. Shih, H. Balakrishnan, and S. Madden. Cartel: a distributed mobile sensor computing system. In *SenSys*, pages 125–138, Boulder, Colorado, USA, Oct-Nov 2006.

[18] C. S. Jensen, D. Lin, and B. C. Ooi. Continuous clustering of moving objects. *IEEE Trans. on Knowl. and Data Eng.*, 19(9):1161–1174, September 2007.

[19] S. Johnson. Hierarchical clustering schemes. *Psychometrika*, 32:241–254, 1967.

[20] P. Kalnis, N. Mamoulis, and S. Bakiras. On discovering moving clusters in spatio-temporal data. In *Advances in spatial and temporal databases*, pages 364–381. Springer, 2005.

[21] M. Keally, G. Zhou, and G. Xing. Watchdog: Confident event detection in heterogeneous sensor networks. In *RTAS*, pages 279–288, Stockholm, Sweden, April 2010.

[22] G. Kollios, D. Gunopulos, N. Koudas, and S. Berchtold. Efficient biased sampling for approximate clustering and outlier detection in large datasets. *IEEE Trans. on Knowl. and Data Eng.*, 2003.

[23] Z. Li, B. Ding, J. Han, and R. Kays. Swarm: mining relaxed temporal moving object clusters. *Proc. VLDB Endow.*, 3(1-2):723–734, September 2010.

[24] S. Lin, B. Arai, D. Gunopulos, and G. Das. Region sampling: Continuous adaptive sampling on sensor networks. In *ICDE*, Cancún, México, April 2008.

[25] R. V. Nehme and E. A. Rundensteiner. Scuba: scalable cluster-based algorithm for evaluating continuous spatio-temporal queries on moving objects. In *EDBT*, Munich, Germany, March 2006.

[26] C. S. d. Oliveira, P. I. Godinho, A. S. G. Meiguins, B. S. Meiguins, and A. A. Freitas. Edacluster: an evolutionary density and grid-based clustering algorithm. In *ISDA*, Rio de Janeiro, Brazil, Oct 2007.

[27] M. Olson, A. H. Liu, M. Faulkner, and K. M. Chandy. Rapid detection of rare geospatial events: earthquake warning applications. In *DEBS*, New York, NY, July 2011.

[28] C. R. Palmer and C. Faloutsos. Density biased sampling: an improved method for data mining and clustering. In *SIGMOD*, Dallas, TX, May 2000.

[29] H.-S. Park and C.-H. Jun. A simple and fast algorithm for k-medoids clustering. *Expert Syst. Appl.*, 36(2):3336–3341, Mar. 2009.

[30] PlanetLab Consortium. http://www.planet-lab.org, 2004.

[31] T. Repantis, X. Gu, and V. Kalogeraki. Synergy: Sharing-aware component composition for distributed stream processing systems. In *Middleware*, Melbourne, Australia, Nov. 2006.

[32] J. Sander, M. Ester, H.-P. Kriegel, and X. Xu. Density-based clustering in spatial databases: The algorithm gdbscan and its applications. *Data Min. Knowl. Discov.*, 2(2):169–194, June 1998.

[33] B. Silverman. *Density Estimation for Statistics and Data Analysis*. Monographs on Statistics and Applied Probability. Chapman & Hall, 1986.

[34] S. Subramaniam, V. Kalogeraki, and T. Palpanas. Distributed real-time detection and tracking of homogeneous regions in sensor networks. In *RTSS*, Rio de Janeiro, Brazil, Dec 2006.

[35] A. Thiagarajan, L. Ravindranath, K. LaCurts, S. Madden, H. Balakrishnan, S. Toledo, and J. Eriksson. Vtrack: accurate, energy-aware road traffic delay estimation using mobile phones. In *SenSys*, Berkeley, CA, Nov 2009.

[36] M. Wand and C. Jones. *Kernel Smoothing*. Monographs on Statistics & Applied Probability. Chapman & Hall, 1995.

[37] J. Yuan, Y. Zheng, X. Xie, and G. Sun. Driving with knowledge from the physical world. In *KDD*, San Diego, California, USA, August 2011.

[38] J. Yuan, Y. Zheng, C. Zhang, W. Xie, X. Xie, G. Sun, and Y. Huang. T-drive: driving directions based on taxi trajectories. In *SIGSPATIAL GIS*, San Jose, CA, November 2010.