Decentralized Processing Strategies for Community Sensing Applications

Heitor Ferreira, Sérgio Duarte, Nuno Preguiça, and David Navalho
CITI-DI-FCT-UNL
Caparica, Portugal
heitorfr@gmail.com, smd@di.fct.unl.pt, nmp@di.fct.unl.pt, david.navalho@fct.unl.pt

ABSTRACT

Participatory Sensing is a new computing paradigm that aims to turn personal mobile devices into advanced mobile sensing networks. In this paper, we describe and evaluate two distributed strategies for processing participatory sensing data. These strategies are built on top of a decentralized architecture composed on user-contributed machines. The evaluation shows that these strategies spread the load among system nodes, thus providing a scalable solution for supporting participatory sensing applications that requires no central node.

Keywords

participatory sensing, distributed processing, mobile computing

1. INTRODUCTION

Participatory Sensing [3, 6] is an emergent pervasive computing discipline that aims to leverage the growing ubiquity of sensor capable mobile phones as the basis for performing wide-area sensing tasks. Taking advantage of people's movements in their daily routines, Participatory Sensing promises an enormous potential for gathering valuable data at a fraction of the cost associated with the deployment of dedicated sensor infrastructures. Examples of this potential are well illustrated in [9, 15, 21], which concern road conservation and traffic monitoring, based on accelerometer and GPS readings collected by mobile devices.

Realizing the potential of Participatory Sensing applications poses several challenges, in particular, that of large-scale support, since the appeal of the paradigm could make its applications popular, but also because high data redundancy may be required to address accuracy concerns. Many of the application examples found in the literature [9, 15, 21] use centralized architectures. These are appropriate for small-scale, proof-of-concept experiments, or if the backing of a large institution can be secured. Decentralized processing, however, offers an economically viable alternative to build a community sensing platform supported mainly by its users. At the same time, it can also address privacy concerns by foregoing the control of user data to a single entity.

In this paper, we present two distribution strategies for fully decentralized processing of participatory data. We describe them as an integral part of the 4Sensing platform - a cooperative system for supporting participatory sensing applications, and evaluate both in a simulator environment. A case study traffic monitoring application is used to compare and discuss their expected performance in a large-scale setting.

The rest of the paper is structured as follows. Section 2 provides an overview of the 4Sensing system. Section 3 details the two distribution strategies proposed in this paper. Section 4 describes two case-study applications developed with 4Sensing, followed by the results of their experimental evaluation in Section 5. The paper concludes with an overview of related work and some conclusions, respectively, in Sections 6 and 7.

2. SYSTEM OVERVIEW

2.1 System Architecture

The 4Sensing system architecture, depicted in figure 1, comprises two kinds of nodes. A large number of mobile nodes, equipped with sensors, are the main source of data for the system, whereas a fixed node infrastructure provides computing and storage resources. Mobile nodes, typically smartphones, are expected to have limited computational power, battery life and communications capabilities. As such, mobile nodes will mainly support user interaction and run data acquisition services on behalf of the applications that run mostly on the fixed infrastructure. The fixed infrastructure can be composed only of personal computers contributed by the users themselves, allowing a self-sufficient system to be assembled and maintained by its own user community. The fixed infrastructure can also include virtual machines and servers hosted by independent entities.

Fixed nodes are organized as a two-tier overlay network of loosely-coupled geographic partitions or clusters. Each fixed node is given a geographic position. Each geographic partition, individually, forms an one-

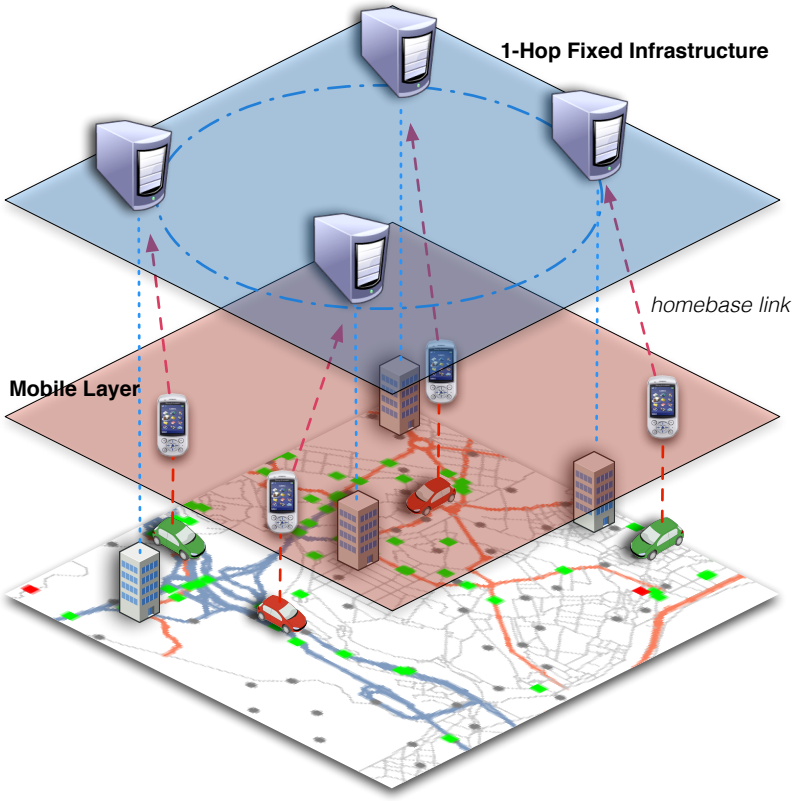

Figure 1: 4Sensing architecture, showing the fixed-infrastructure interconnected by an 1-hop overlay. Each mobile node uploads sensor information via a fixed-insfrastructure node - its homebase

hop DHT [14], as described in [7], meaning that every node strives to maintain complete knowledge about its cluster's node composition (including their coordinates). These partitions are organized according to geography and are centered on major points of interest, such as a large metropolitan area. This arrangement leverages geography to optimize communications latency and throughput, exploiting the strong correlation expected regarding where sensory data is produced and consumed. Inter-cluster connectivity is provided by a top-tier. This tier is also organized as an one-hop DHT and is populated by a few selected nodes (of each partition) with the added task of maintaining routing information to other clusters.

Mobile nodes do not interact directly. They connect to the fixed infrastructure nodes to upload sensory data and request processed information via the Internet (e.g., 3G connections). A privileged link to a *homebase* fixed node is used to facilitate this and serve as a personal gateway. The fixed homebase is useful in that it minimizes the routing logic and network involvement on the mobile side. For instance, the homebase node can be used to automatically refresh data request leases on behalf of its mobile node.

2.2 Data Model

Applications access participatory sensing data by issuing queries over *virtual tables* - the main high-level data abstraction provided by 4Sensing for naming a collection of data with a particular schema and semantics. Virtual tables have a global scope within a particular instance of the 4Sensing platform. Issuing a query over a virtual table serves two purposes, it allows applications to request data and, at the same time, restrict its scope to a subset of the whole, in the form of spatial and temporal constraints.

While flexible regarding the actual schema, virtual tables handle geo-referenced and time-stamped data. As such, virtual table data is represented as a set of named attributes and, as a common denominator, each data tuple must include a temporal attribute, i.e., a timestamp, and a spatial component, in the form of physical coordinates or a geographic extent. A query over a virtual table will then result in a sequence of data tuples to be forwarded to the requesting application, confined to the requested geographic area and time period.

Virtual tables are purely logical entities, in the sense they do not necessarily need to refer to data already present in some physical storage medium. Interfacing

applications with persisted (stored) data is just one of the facets of this construct. Virtual tables can also target data that does not yet exist and that may never be stored, including data intended to be produced and consumed with (near) realtime requirements. Moreover, another central concept to virtual tables is that they refer to data produced as a result of applying a set of transformations to raw sensor inputs and other virtual tables. As such, virtual tables embody a generic inference mechanism that creates high-order data from simpler constituents. This design is intended to facilitate and promote data sharing and cooperation among otherwise unrelated applications.

2.2.1 Data transformation

Data transformation is achieved by specifying the execution of a *pipeline* of successive operations. Starting with raw sensor samples, or data from other virtual tables, a derived class of data can be produced, based on the output of a combination of pipeline operators. Data transformations typically involve *mapping*, *aggregation* and *condition detection* operations. Mapping adds new attributes to individual data elements, such as tagging a GPS reading with a street segment identifier. Aggregation combines several samples into one and often involves statistical operators, such as *count*, *average*, etc., over temporal data snapshots. Finally, condition detection generates new data indicative of a particular noteworthy event, such as a congested street.

The *TrafficHotspots* virtual table, c.f. Definition 1, exemplifies the inference logic for an application devoted to realtime traffic monitoring, which is later detailed. In this example, starting with discrete GPS readings collected by mobile nodes, a stream of *Hotspot* tuples is generated, representing real-time detections of congested road segments, when certain thresholds are exceeded. In between, data is progressively transformed into intermediate forms, such as *MappedSpeed* and *AggregateSpeed* representing for each road segment, respectively, an individual speed measurement and the average car speed for the last 10 seconds. In section 4 we detail this application. The domain-specific language we have defined for specifying virtual tables and the associated data transformations is detailed elsewhere [11, 10]. A key aspect about data processing in virtual tables is that it is split between two separate transformation pipeline stages. One, *dataSource*, processes data locally available to the node, i.e., data already stored locally or data that is being acquired and uploaded by mobile nodes. The *globalAggregation* merges the contributions of several nodes. Together, the two pipeline stages abstract the 4Sensing distributed data processing facet. Consequently, their operation is very closely tied to the distribution strategy employed, which among other things determines the shape and properties of the infrastructure aggregation tree used, as discussed further on.

2.2.2 Data Dissemination

When a node issues a query, the system instantiates data processing pipelines in certain fixed infrastructure nodes, depending on the geographic coverage of the standing queries. For returning results to the client, 4Sensing adopts a push-based model that leverages a publish/subscribe, content-based routing substrate. As data becomes available in the distributed processing pipelines, it is published and routed by the fixed infrastructure to the interested nodes. When results are to be returned to mobile nodes, they are propagated via their respective *homebase* nodes.

An in-depth discussion of this content-based routing substrate is out of scope of this paper and can be found in [7]. However, the following key characteristics are important to highlight. The substrate spans all fixed infrastructure nodes. Mobile nodes do not participate directly and interface via their respective homebase nodes. It leverages the one-hop DHTs to assemble, on-the-fly, a random dissemination tree for each message separately. The result is trees that span solely the message's publisher node and the actual recipient nodes (i.e., produces no false positives). Successive messages produce different dissemination trees, which improves load-balancing. Fault-tolerance measures also ensure that no false negatives are introduced in the presence of network churn, so that messages will eventually reach all known recipients at the time of publishing, with high probability.

3. DISTRIBUTION STRATEGIES

The abstract model for distributed processing outlined above can be materialized by different distribution strategies, which determine how the overall infrastructure is organized to support multiple participatory sensing applications. In particular, each strategy defines the role of the homebase, how data is acquired by the system, how queries are distributed to relevant nodes, and how data is aggregated. Different strategies strive for different strengths. We next describe two strategies that we have implemented in the 4Sensing platform.

3.1 RTree

The main rational behind RTree (Random Tree) is to have the homebase as a personal data repository and use it for privacy protection mechanisms - such as the privacy firewall advocated in [12]. Personal stores can be private - an incentive for users to contribute resources to the system infrastructure - or shared by a group of users. All data acquired by a mobile node is uploaded to the user's homebase, which is expected to cover a po-

Definition 1 TrafficHotspots virtual table specification

```
sensorInput( GPSReading )
}
dataSource {
   process{ GPSReading r ->
      r.derive( MappedSpeed, [boundingBox: model.getSegmentExtent(r.segmentId) ])
   }
   timeWindow(mode: periodic, size:10, slide:10)
   groupBy(['segmentId']){
      aggregate( AggregateSpeed ) { MappedSpeed m ->
         sum(m, 'speed', 'sumSpeed')
         count(m, 'count')
      }
   }
}

globalAggregation {
   timeWindow(mode: periodic, size:10, slide:10)
   groupBy(['segmentId']){
      aggregate( AggregateSpeed ) { AggregateSpeed a ->
         avg(a, 'sumSpeed', 'count', 'avgSpeed')
   } }
   classify( AggregateSpeed ) { AggregateSpeed a ->
      if(a.count > COUNT_THRESHOLD && a.avgSpeed <= SPEED_THRESHOLD * model.maxSpeed(a.segmentId))
         a.derive( Hotspot, [confidence: Math.min(1, a.count/COUNT_THRESHOLD*0.5) ])
   }
}
```

tentially large geographic area (e.g., a city-wide area) around the user's home, workplace and in-between.

3.1.1 Query Dissemination and Processing

The determining aspect of RTree is that every homebase potentially hosts data relevant to any particular query. Targeting every homebase is too inefficient, as a tradeoff, RTree restricts query distribution by having each homebase define the geographical area it is willing to serve - its *query filter*. In general, this area will be restricted to the regions its owner visits more often, thus guaranteeing that all (or most of) collected data is shared with the community.

For the purpose of distributed processing, RTree treats overlapping queries as a single one. Therefore, for each set of overlapping queries, RTree constructs a random tree spanning all homebases whose query filters intercept the union of the query areas. The distributed algorithm that assembles these trees leverages the same publish/subscribe substrate used to disseminate query results. When a mobile node issues a new query, its homebase publishes the query (characterized by a geographic extent). All homebases with intercepting query filters (and only those) will receive the query and will try to merge it with existing queries. If a node determines that the new query overlaps the boundary of already existing queries, the resulting compound (wider) query is published, repeating the process. Eventually, no more overlaps are produced and the process stops. Since the publish/subscribe substrate already produces a random tree for each published message, each node only has to record the path each query has taken to determine its place in the random aggregation tree. When individual queries are not renewed and expire, it can cause larger compound queries to shrink or split into disjoint pieces. This is also handled in a distributed fashion. When a node determines a individual query has expired, it eliminates it and re-merges the remaining ones. Again, if there is a change in the resulting set, it publishes the new merged queries, mimicking

the initial process. Logical timestamps and other optimizations are used to limit the number of messages exchanged and handle concurrent re-publish operations from different nodes. To improve on load-balancing, merged queries can be re-published periodically, producing a new tree in the process. RTree can use aggregation trees with different fanout degrees, by instrumenting the publish/subscribe substrate to produce the desired value, for instance to control tree depth.

In a given RTree instance, every node (homebase) has the same role and will instantiate both the data source and global aggregation pipeline stages. Since a node cannot determine if it has complete information for a particular spatial extent, data aggregation and processing may have to proceed until the root is reached. This is particularly problematic for virtual tables whose inference logic requires detecting the absence of certain data patterns. Early detections can be published as soon as they are produced in any point of the aggregation tree. As discussed earlier, data dissemination uses dedicated trees and is delivered exclusively to nodes with matching (individual) queries.

3.2 QTree

In QTree (Quad Tree), the idea is to do an *a priori* partition of the data by performing a regular subdivision of the space into quadrants, while the number of infrastructure nodes located in each area exceeds a *minimum occupancy* threshold. Each infrastructure node belongs simultaneously to all the quadrants that contain it, down to the smallest - called its *maximum division quadrant*. Under this scheme, acquired sensor data is committed and bound to any of the fixed nodes that lie in the smallest quadrant that fully encloses the data's coordinates or geographic extent. Therefore, in QTree, dataSource pipeline stages will process data acquired by any mobile node. In the deepest levels of the QTree aggregation tree, all the processed data pertains to the immediate neighborhood of the processing node. This enables QTree nodes to potentially detect highly localized phenomena more efficiently and sooner in the aggregation tree. A more detailed explanation of QTree is found in [11, 10].

3.2.1 Query Dissemination and Processing

QTree and RTree share a large number of features: overlapping queries are merged and split using the same re-publish strategy; a random tree containing all relevant infrastructure nodes is created by disseminating the query using a distributed algorithm; periodically re-publishing the query can be used to improve load-balancing; data dissemination uses dedicated trees spanning only nodes with matching queries.

The algorithms differ mainly on the query filter used and how queries are routed to build the aggregation tree. In QTree, the *query filter* corresponds to the *minimum subdivision quadrant* it belongs to[1]. As for query (content-based) routing, it is performed by recursively subdividing space along quadrant boundaries. At each point of the routing tree, the current quadrant is divided into sub-quadrants, selecting for each of them, a child node, at random, from those enclosed and whose query filters intercept the query area. Along the way, the query path is updated and appended to the query as it travels along the dissemination tree, which will also serve as the data processing aggregation tree.

The role of homebase node in QTree is mainly that of a gateway to the fixed infrastructure. They do not necessarily store or process the data collected by their respective mobile nodes, but commit and route it to a suitable infrastructure node. By binding data to nodes close by, QTree allows for early detection of some data patterns (e.g., a certain number of samples in a given area has been reached), which can be published and notified to applications sooner, typically deep in the aggregation tree.

4. APPLICATIONS

In this section we will detail how to use 4Sensing for creating participatory sensing applications. As 4Sensing is focused on data processing, we will focus mainly on how to define virtual tables in such applications.

4.1 Hot-city areas

Our first application, *Hot-city areas* or simply *Hot-areas*, provides information on which areas are currently hot in some given city (or other geographical area). This application could be used, for example, by tourists in a city to find areas of interest if, for example, users would feed the system with their location whenever they take a picture or make a twitter/facebook post. Even more interesting would be to use a similar application in a natural park, allowing users to find where animals are being spotted at some given moment.

In our example, the information generated by users consists in geographically-tagged Tweeter posts (or simply, tweets). The geographic area is divided in a grid, and each tweet is mapped to a grid square based on its geographic position. To create a map with the hot areas in the city, it is only necessary to count, for each grid square, the number of recent tweets in the area.

Figure 2 presents the definition of the *HotAreas* virtual table that allows to compute this information in 4Sensing. As explained before, a virtual table definition has two main parts: the *dataSource* part, specifying how samples produced by users are initially processed; and the *globalAggregation* part, specifying how

[1]QTree assumes space is sufficiently populated that it can be subdivided to achieve a uniform *minimum subdivision quadrant*.

Definition 2 HotAreas virtual table specification

```
sensorInput( TwitterReading )

dataSource {
   process{ TwitterReading t ->
      r.derive( MappedTweeter, [cellId: model.getCellId(r, boundingBox: model.getCellBounds(r)])
   }
   timeWindow(mode: periodic, size:30*60, slide:30)
   groupBy(['cellId']){
      aggregate( AggregateTweeters ) { MappedTweeter m ->
         count(m, 'totalTweets')
      }
   }
}

globalAggregation {
   groupBy(['cellId']){
      set(['peerId'], mode: triggered, ttl: 30, period: 10)
      aggregate( AggregateTweeters ) { AggregateTweeter a ->
         sum(a, 'totalTweets', 'totalTweets')
      }
   }
}
```

data from multiple users is combined to compute the needed information.

In this case, the *dataSource* maps each tweet into a *cellId*, which corresponds to a square in the grid of the geographic area. Tweets are stored in a sliding window for 30 minutes. For each *cellId*, every 30 seconds, the *dataSource* counts the number of tweets in the sliding window and passes this value to the *globalAggregation* phase as an *AggregateTweeter*.

The *globalAggregation* phase aggregates reports for each *cellId*. In the dissemination tree that is created to aggregate information, a report received from a child is valid for 30 seconds. Merging of child reports is triggered to occur 10 seconds after the first report is received, to account for the lag between child reports.

An application that uses 4Sensing will receive the information produced by the *HotAreas* virtual table, restricted to the area of interest specified in the submitted query. To test this definition, we have created an application that uses the information from *HotAreas* virtual table to build an heat map based on the realtime stream provided by Twitter[2]. The application runs in a simulated environment consisting of 500 fixed nodes. Each tweet in the realtime stream will generate a *TwitterReading* produced by a mobile node, and propagated to a fixed node, in our simulated environment. Figure 2 presents the heat map generated for the San Francisco city.

[2] `http://dev.twitter.com/pages/streaming\_api`

4.2 Traffic hotspots

Our second application, *Traffic hotspots*, provides information about the congested areas in city traffic. The application detects congestions in a given area, based on GPS data sampled by in-transit vehicles at periodic intervals. For this purpose, one virtual table is used: TrafficHotspots, which supports querying for congestion detections, computed from average speeds. From this information, it is possible to generate a map that shows the traffic conditions at some given moment - e.g. figure 3 shows traffic with different colors depending on whether it is in a congested road or not.

The code for the TrafficHotspots virtual table is presented in Definition 1. In the *dataSource* part, GPS samples received directly from mobile nodes are augmented to include the identifier of the road segment the vehicle is in. For each road segment, an *AggregateSpeed* sample is generated with aggregate information of the cars in the segment. These *AggregateSpeed* samples are

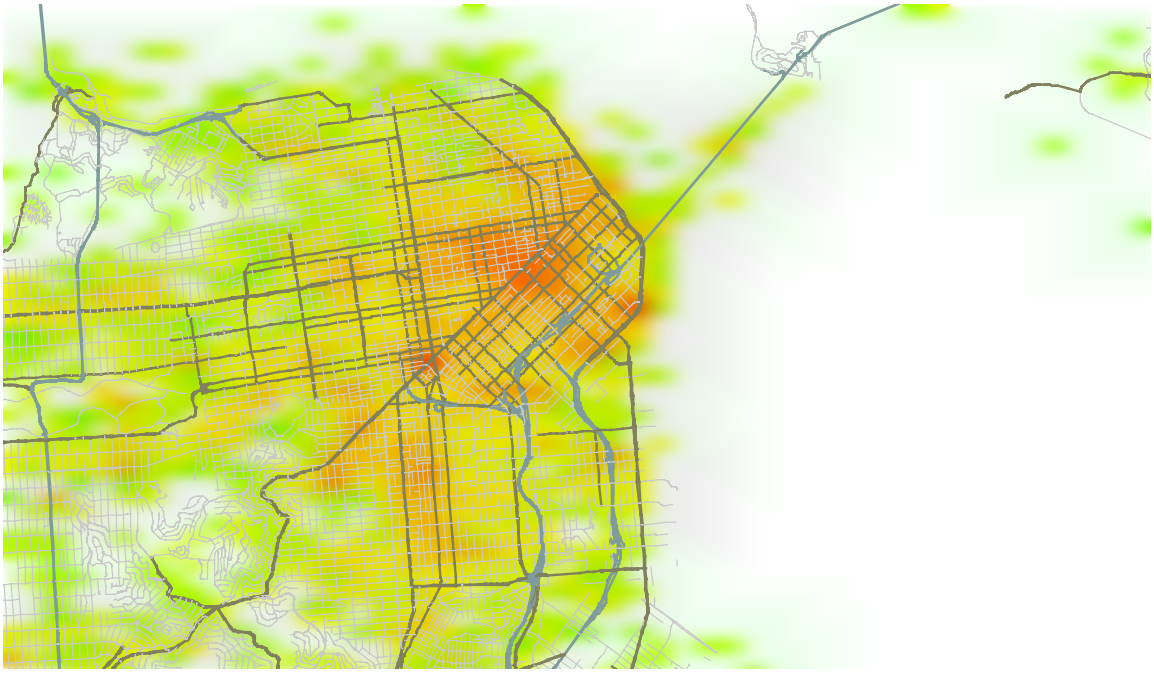

Figure 2: Snapshot of the HotAreas application based on realtime tweets for San Francisco

propagated every 10 seconds for the global aggregation part.

In the *globalAggregation* part, a sliding window of the *AggregateSpeed* samples received in the last 10 seconds is maintained. For each road segment, the total number of car and the average car speed is computed. With this information, an hotspot is signaled for some segment if the number of cars in the segment and their average speed match some given thresholds (that depend on the type of road).

5. EVALUATION

When comparing the two applications presented in the previous section, the traffic hotspots application is much more demanding for the processing system, as much more messages are generated per time unit - while in the traffic hotspots applications all mobile nodes periodically generate GPS samples, in the hot-city areas mobile nodes only produce GPS samples from time to time. As the goal of the evaluation was to compare the performance of the different distribution strategies, we have decided to use the traffic hotspots application in the evaluation.

In our simulation, we are using the Open Street Map (OSM) [23] vectorial representation of the road network of Lisbon city. This map data is used to map geographic coordinates to road segments, to determine the spatial extent of segments and their associated road type. The model used in the evaluation divides roads, as needed, into segments with a maximum of 1 km and uses separate segments for each driving direction. Each road is assigned an expected (uncongested) driving speed according to its type: highway, primary to tertiary and residential.

In the experiments, we simulated 5000 fixed nodes and 50000 mobile nodes. Fixed nodes are distributed randomly across the urban space with a minimum internode distance of 80 meters. Mobile nodes simulate intransit vehicles, according to a traffic model, and report GPS readings every 5 seconds as they follow the assigned paths. They interact with the respective homebase counterpart directly, resulting in the delivery of raw GPS data with no latency. Communication among fixed nodes experiences latency and jitter. A common clock is used to timestamp readings; any effects of clock desynchronization are not considered.

Traffic is modeled by emulating a fleet of vehicles driving through random routes. The maximum speed for a given segment is the same value used for congestion detection and depends on the road type. An average speed, for each road segment at a given time, is determined by its current car density and used to generate the random speed individually for each vehicle, according to a normal distribution. In the experiments performed, congestion occurs in segments with a density of at least 50 vehicles. Vehicle paths are determined by choosing a random start position and sequence of road intersections; a new path is assigned whenever a vehicle reaches its destination. Figure 3 shows a rendering of the traffic simulation.

In the following results, we have evaluated the effects of a single query covering 25% of the overall simulation

Figure 3: Snapshot of the traffic hotspots application for Lisbon metropolitan area

area in an area with high density of mobile nodes. Note that having a query over a large area is what happens when multiple queries over nearby small areas (e.g., several points in the city downtown) are merged in our algorithms. The set of metrics captured was averaged over 5 runs, corresponding to different fixed node placements.

5.1 Workload Distribution

We have started by evaluating how the effort required to evaluate a query is spread among the fixed nodes in the different distribution strategies. For comparison, we have also computed the effort that a centralized solution, with a single central node, would experience.

The workload is measured as the number of data tuples processed at each fixed node in both acquisition and aggregation steps. Specifically, the acquisition pertains to the number of GPS sensor readings received and processed by the acquiring node, while the aggregation refers to the number of inputs handled by the global aggregation stage and corresponds to the updates received for each segment aggregated by that node.

Table 1 presents the workload using a centralized approach, while Figure 4 shows the workload distribution using the different distribution strategies, which for RTree involved aggregation trees with fanout degree of 4 and 8, besides the default value of 6.

The results show that using a decentralized approach, the most loaded node only processes a small fraction of the tuples processed in a centralized approach - 0.8% for QTree and 11.5% for the best RTree configuration. Comparing the two distribution strategies, QTree balances load very effectively, while in RTree, irrespective of the configuration, there are some nodes with a much higher load. The reason for this is that the detection of congestion in QTree is done faster, in the lower levels of the tree, because tuples of the same area will be aggregated in some node sooner. In RTree, there is often the need to propagate the information up to the top of the tree to check if a congestion exists or not, as data for the same areas may be propagated in any path. This explains why the most loaded nodes, corresponding to the nodes near the root of the tree, have so much load.

Acquisition	278,969	96.3%
Aggregation	10,614	3.7%
Total	289,583	100%

Table 1: Workload using centralized processing (tuples/minute)

5.2 Communication Load

We have also measured the communication cost, as the number of messages exchanged during the execution of a query. The communication load includes one-hop DHT overhead (4.4 messages/min per node) [3], acquisi-

[3]Corresponding to a fixed cost of 7.5 join/leave DHT

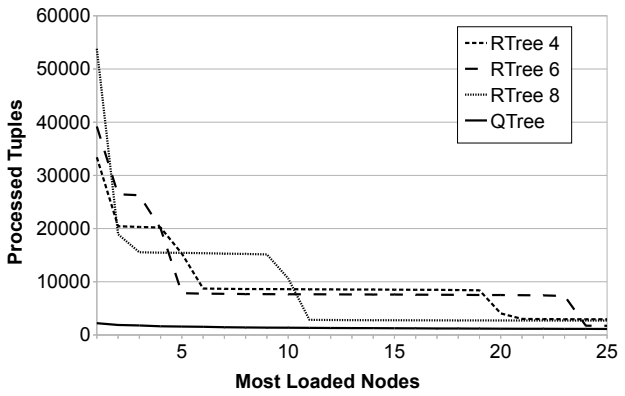

Figure 4: Workload distribution - for the 50 most loaded nodes

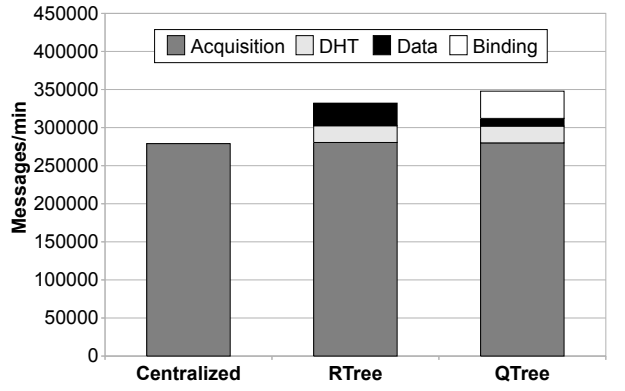

Figure 5: Total communication cost

tion messages, data messages and binding events. Acquisition messages are those sent by the mobile nodes to a fixed node. Data messages carry the tuples exchanged between peers, relative to query data (incomplete aggregations that are forwarded up the tree). Binding events capture the additional overhead associated with uploading the data from mobiles to a fixed node in QTree, where the node the data is sent varies over time. In all cases, the number of messages sent by individual nodes is limited at 1 message per pipeline stage every 10 seconds by the use of the *timeWindow* operator.

Figure 5 presents the total communication cost in the centralized scenario and in both distribution strategies presented. As expected, decentralized approaches involve additional messages, as data with intermediate computations needs to be exchanged among nodes in the system. However, results show that these messages represent only a small fraction of the messages propagated from the mobile nodes to the fixed infrastructure in any case. The reason for this is that messages exchanged by the fixed node usually carry data that is computed from a large number of data items received

events per minute, cf. [7].

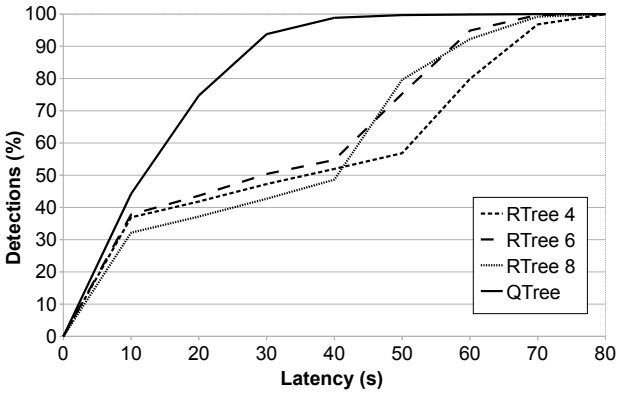

Figure 6: Detection latency

from mobile nodes and data propagation can stop whenever a detection is performed at some tree node.

Comparing the two distribution strategies, RTree exchanges more data messages than QTree, as expected, because congestion computation tends to be computed in higher levels of the tree, requiring more messages to be propagated. However, QTree has an additional overhead related with binding events, as mobile nodes need to know to which fixed node to propagate a message when they move to a different area. This overhead is important, leading to an overall larger communication cost in QTree when compared with RTree. In the future, we intend to explore strategies to reduce the need for these binding events, e.g., by caching in mobile nodes the identity of fixed nodes responsible for a query in adjacent areas.

5.3 Detection Latency

Detection latency measures the lag between the occurrence of a segment congestion and its detection by the system. Transient congestions (lasting less than 20 seconds) were not considered for the evaluation. As the traffic patterns produced by the traffic model are highly dynamic compared to real world conditions, with frequent short lived congestions, these experiments represent a challenging scenario to the system.

Figure 6 plots the detection latency for the accumulated level of successful detections. Results show clearly that QTree detects congestions faster than RTree. The reason for these results lie in the fact that in QTree, related data flow to the same nodes faster, making detections to occur in the lower levels of the tree. Up to 35% of successful detections, latency is similar for QTree and RTree. This is explained by the fact that, when a road is very congested, with much more cars than necessary for a detection, even in RTree enough data is aggregated in the lower levels of the tree.

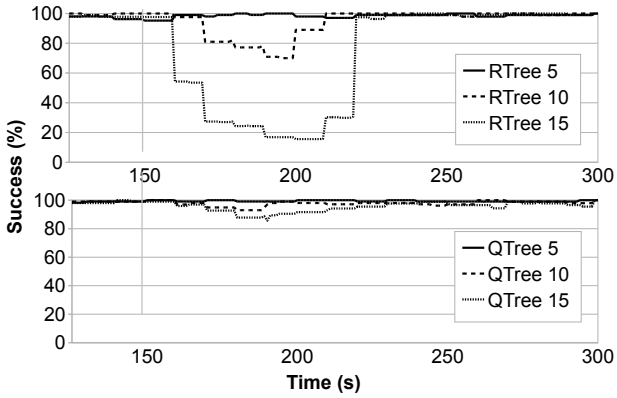

Figure 7: Success rate in the presence of node failures (at time = 100)

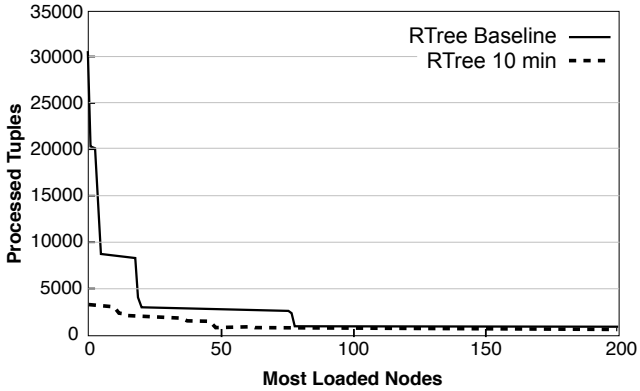

Figure 8: RTree - Workload distribution after periodic aggregation tree updates

5.3.1 *Failures*

We implemented a simple failure recovery mechanism consisting in rebuilding the aggregation tree periodically or when failures exceed a certain threshold. Using this strategy, we observed that the loss of a small percentage of nodes has little impact on the query success rate, depending on the sensor acquisition rate used. Figure 7 summarizes those results, which involved sensor acquisition intervals of 5, 10 and 15 seconds and the sudden failure of 15% of the fixed nodes. As expected, detection success is adversely affected by node failures but afterwards it recovers to its original values. The lower the acquisition rate, the more pronounced the effect is. This behavior is explained by the fact that node failures cause the loss of some of the acquired data and, until a new tree is built, aggregation is impaired due to broken tree paths. If the acquisition rate is high enough, the effect can be negligible due to data redundancy. In the opposite case, the impact is more severe since the amount of lost data is proportionally higher. Comparing both distributed strategies, RTree suffers the most from node failures. In RTree node failures tend to have a global impact because data is spread among all nodes, so it requires enough data redundancy and tree integrity to operate properly. Due to its *a priori* reliance on data locality, QTree does not need to aggregate as much, so the effect of node failures is also much more localized and impacted less by problems in the aggregation tree.

5.3.2 *Improving Load-Balancing*

The evaluation results presented in the previous sections show that RTree can suffer from load-balancing issues, arising from an excessive burden imposed on the nodes comprising the top levels of the aggregation tree. To address this problem, RTree can rebuild the aggregation tree from time to time, taking advantage of the randomness that is the basis for this distribution strategy. Re-publishing existing queries, periodically, causes a new aggregation tree to be assembled each time. This tends to shuffle the nodes' positions in the tree and, over time, workload asymmetries can be expected to attenuate. Figure 8 shows the effect of rebuilding the aggregation tree every 10 minutes for an RTree (fanout degree of 4), compared to the static tree scenario. The results corresponds to running a query for a total of 2 hours. They clearly show that just a few tree updates are very effective at better spreading the load among the infrastructure nodes. Running the query longer will extend those gains. Faster tree updates show the same improvement sooner, however, updating too often can interfere on the normal flow of data along the processing pipelines, depending on the dimension of the windows in use.

We also examined the effect of updating the aggregation periodically on QTree, since it also relies on a random aggregation tree, albeit a more constrained one. Those results are summarized in Figure 9. It can be observed that QTree load-balancing performance also shows improvement compared to the reference static scenario. However, the effect is much less pronounced than that of RTree. This is within expectations, since the baseline QTree configuration already achieved good load-balancing performance, by performing a significant amount of processing work on the lower levels of the tree.

6. RELATED WORK

Participatory sensing has become popular recently [3, 6], but a large number of supporting systems have already been designed.

In BikeNet [8], users can share information about their bike rides. PEIR [22] computes estimates of environmental impact and exposure from location samples. CarTel [15, 9] focuses on vehicle based sensing applications and served as inspiration for our evaluation scenario. Although these systems are different, and some

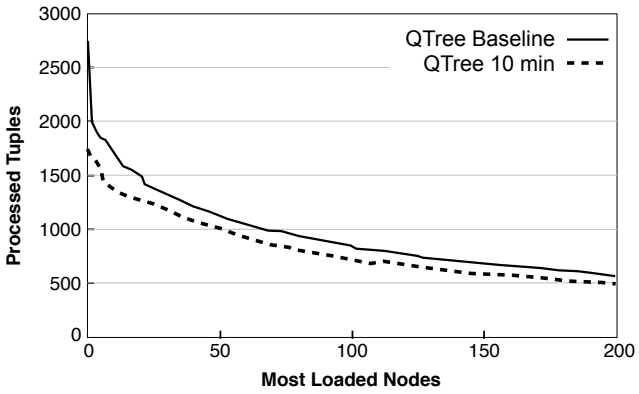

Figure 9: QTree - Workload distribution after periodic aggregation tree updates

even rely on a delay-tolerant network substrate, they all are based on a centralized approach. Our work studies decentralized solutions that improve the scalability of the system, by spreading the processing load requirements by a large number of nodes, as shown in the results of our evaluation. Lacking the need for having a powerful server infrastructure also makes this approach more suitable for community-based sensing, with the needed resources being contributed by the community.

Some sensing systems present an architecture built entirely in the mobile nodes and ad-hoc coordination to support applications (e.g. [24] and [18]). A limitation of this approach is that mobile nodes have to spend computation, communication and energy resources in coordination efforts, regarding node and service discovery and context dissemination. This is specially relevant given that communication can be expensive and energy is scarce.

Other systems (e.g., [20, 13]) have an architecture more similar to ours, based on the combination of fixed and mobile nodes. For example, in SensorWeb [13], a set of fixed nodes act as gateways for sensor network and proxies for mobile devices. In this system, a coordinator node mediates and coordinates the access of applications to the different gateways and proxies. Unlike SensorWeb, in our system, nodes are also used to process information, thus allowing to more evenly distribute the load.

Sensor networks, while operating under a different context and communication assumptions, provide inspiration on how to gather and share data among nodes [1]. TinyDB [19] extends the standard SQL syntax with a continuous query semantics, providing highly flexible queries as well as summarization and event detection. Directed Diffusion [16] presents a data-centric solution, where a node requests data by sending *interests* for named data. Data matching this query is then gathered towards the node. In 4Sensing, the sensor definition, virtual table and pipeline models separate querying from acquisition and data operations, embodying characteristics of topic-based publish-subscribe systems and SQL based queries.

In wireless sensor networks, a large number of decentralized algorithms have been proposed [27]. However, these algorithms tend to focus on minimizing energy usage and nodes can only communicate with nearby nodes. Thus, the overlay topologies that are formed to aggregate results have very different constraints when compared to our approach. Our algorithms are closer to distributed processing of data streams - e.g. [4, 5]. However, the application scenario makes our solution different, by having mobile nodes feed geo-referenced data into the system periodically. In QTree, we explore this property for aggregating related data at lower levels of the dissemination tree, allowing to make detections faster.

Quad trees have been previously used for geographical-related problems concerning the monitoring and tracking of moving objects [26, 2]. In [17], moving nodes, identified by object identifiers, constantly or periodically update their positions, with the tree being updated with their new positions. Queries are made periodically to determine the set of objects that are contained within each query. In [25], a quadtree is adapted to a peer-to-peer network, using a recursive subdivision of space to assign spatial objects to the index nodes. QTree mixes some aspects of the described spatial indexes. It uses a regular spatial decomposition characteristic of quadtrees indexes. Similar to [25], each level of the decomposition is assigned to a particular node. However, QTree builds a query specific tree during query dissemination instead of relying on a fixed index tree, made possible because of the shared responsibility over each partition among the enclosed nodes.

7. CONCLUSIONS

In this paper, we presented and evaluated two algorithms for decentralized processing of Participatory Sensing data. Namely, RTree, which leverages random aggregation trees and personal data repositories, and QTree, which combines random trees with regular, *a priori*, subdivision of geographic space.

Their experimental evaluation using simulation and a case-study application, allows us to draw the following conclusions. Both decentralized solutions, in regards to data processing, show a small total message overhead compared to a centralized solution, and are clearly competitive. The most loaded node processes a small fraction of the data in both algorithms. QTree, however, performs better in this respect, spreading the processing load more evenly. QTree also shows that it can produce results earlier, exhibiting a smaller processing latency.

Furthermore, experiments have shown that updating

periodically the random aggregation trees that qtree and RTree utilize is effective at improving load-balancing among the processing nodes. The gain is especially significant for RTree, whose main motivation lies in addressing privacy issues and allowing participatory data to be managed through personal data repositories and privacy firewalls.

As for future work, we intend to continue scrutinizing how periodic update of the aggregation trees impacts on load-balancing, processing overhead and query success and latency. Moreover, we want to evaluate inter-partition interactions and measure the impact on the overall system performance of inter-partition queries.

8. REFERENCES

[1] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. A survey on sensor networks. *Communications Magazine, IEEE*, 40(8):102–114, November 2002.

[2] B. R. Badrinath and T. Imielinski. Replication and mobility. In *Workshop on the Management of Replicated Data*, pages 9–12, 1992.

[3] A. T. Campbell, S. B. Eisenman, N. D. Lane, E. Miluzzo, R. A. Peterson, H. Lu, X. Zheng, M. Musolesi, K. Fodor, and G.-S. Ahn. The rise of people-centric sensing. *Internet Computing, IEEE*, 12(4):12–21, 2008.

[4] M. Cherniack, H. Balakrishnan, M. Balazinska, D. Carney, U. Çetintemel, Y. Xing, and S. B. Zdonik. Scalable distributed stream processing. In *CIDR*, 2003.

[5] T. Condie, N. Conway, P. Alvaro, J. M. Hellerstein, K. Elmeleegy, and R. Sears. Mapreduce online. In *NSDI2010: Proceedings of the 6th USENIX Symposium on Networked Systems Design and Implementation*, 2010.

[6] D. Cuff, M. Hansen, and J. Kang. Urban sensing: out of the woods. *Commun. ACM*, 51(3):24–33, 2008.

[7] S. Duarte, J. L. Martins, M. Mamede, and N. Preguiça. SIFT - Decentralized Load-balanced Content-based Routing. Technical Report UNL-DI-4-2010, `http://goo.gl/ACQw2`, Universidade Nova de Lisboa, December 2010.

[8] S. B. Eisenman, E. Miluzzo, N. D. Lane, R. A. Peterson, G.-S. Ahn, and A. T. Campbell. The bikenet mobile sensing system for cyclist experience mapping. In *SenSys '07: Proceedings of the 5th international conference on Embedded networked sensor systems*, pages 87–101. ACM, 2007.

[9] J. Eriksson, L. Girod, B. Hull, R. Newton, S. Madden, and H. Balakrishnan. The Pothole Patrol: Using a Mobile Sensor Network for Road Surface Monitoring. In *MobiSys '08: Proceeding of the 6th Int. Conf. on Mobile systems, applications, and services*, June 2008.

[10] H. Ferreira. 4Sensing - Distributed Processing for Participatory Sensing Data. Master's thesis, DI - FCT - Universidade Nova de Lisboa, June 2010.

[11] H. Ferreira, S. Duarte, and N. Preguiça. 4Sensing - Decentralized Processing for Participatory Sensing Data. In *16th International Conference on Parallel and Distributed Systems (ICPADS 2010)*. IEEE, 2010.

[12] R. K. Ganti, N. Pham, Y.-E. Tsai, and T. F. Abdelzaher. Poolview: stream privacy for grassroots participatory sensing. In *SenSys '08: Proceedings of the 6th ACM conference on Embedded network sensor systems*, pages 281–294. ACM, 2008.

[13] W. Grosky, A. Kansal, S. Nath, J. Liu, and F. Zhao. Senseweb: An infrastructure for shared sensing. *Multimedia, IEEE*, 14(4):8–13, Oct.-Dec. 2007.

[14] A. Gupta, B. Liskov, and R. Rodrigues. One hop lookups for peer-to-peer overlays. In *Ninth Workshop on Hot Topics in Operating Systems*, pages 7–12, May 2003.

[15] B. Hull, V. Bychkovsky, Y. Zhang, K. Chen, M. Goraczko, A. K. Miu, E. Shih, H. Balakrishnan, and S. Madden. CarTel: A Distributed Mobile Sensor Computing System. In *4th ACM SenSys*, November 2006.

[16] C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed diffusion: a scalable and robust communication paradigm for sensor networks. In *MobiCom '00: Proceedings of the 6th annual international conference on Mobile computing and networking*, pages 56–67, New York, NY, USA, 2000. ACM.

[17] D. V. Kalashnikov, S. Prabhakar, and S. E. Hambrusch. Efficient evaluation of continuous range queries on moving objects. In *DEXA 2002, Proc. of the 13th International Conference and Workshop on Database and Expert Systems Applications, Aix en Provence*, pages 731–740, 2002.

[18] N. Kotilainen, M. Weber, M. Vapa, and J. Vuori. Mobile chedar: A peer-to-peer middleware for mobile devices. In *PERCOMW '05: Proc. of the 3rd IEEE Int. Conf. on Pervasive Computing and Communications Workshops*, pages 86–90. IEEE Computer Society, 2005.

[19] S. R. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. Tinydb: an acquisitional query processing system for sensor networks. *ACM Trans. Database Syst.*, 30(1):122–173, 2005.

[20] Y. J. L. Marie Kim, Jun Wook Lee and J.-C. Ryou. Cosmos: A middleware for integrated data processing over heterogeneous sensor networks. *ETRI Journal*, 30(5), October 2008.

[21] Mohan, Prashanth and Padmanabhan, Venkata and Ramjee, Ramachandran . Nericell: Rich Monitoring of Road and Traffic Conditions using Mobile Smartphones. In *Proceedings of ACM SenSys 2008*, November 2008.

[22] M. Mun, S. Reddy, K. Shilton, N. Yau, J. Burke, D. Estrin, M. Hansen, E. Howard, R. West, and P. Boda. Peir, the personal environmental impact report, as a platform for participatory sensing systems research. In *MobiSys '09: Proc. of the 7th Int. Conf. on Mobile systems, applications, and services*, pages 55–68. ACM, 2009.

[23] OpenStreeMap. http://www.openstreetmap.org, April 2010.

[24] O. Riva and C. Borcea. The urbanet revolution: Sensor power to the people! *Pervasive Computing, IEEE*, 6(2):41–49, April-June 2007.

[25] E. Tanin, A. Harwood, and H. Samet. Using a distributed quadtree index in peer-to-peer networks. *VLDB Journal*, 16:165–178, 2007.

[26] J. Tayeb, Ö. Ulusoy, and O. Wolfson. A quadtree-based dynamic attribute indexing method. *Comput. J.*, 41(3):185–200, 1998.

[27] J. Yick, B. Mukherjee, and D. Ghosal. Wireless sensor network survey. *Comput. Netw.*, 52(12):2292–2330, 2008.