# Scalable Data Processing for Community Sensing Applications*

Heitor Ferreira, Sérgio Duarte, Nuno Preguiça, and David Navalho

CITI-DI-FCT-UNL, Caparica, Portugal

**Abstract.** Participatory Sensing is a new computing paradigm that aims to turn personal mobile devices into advanced mobile sensing networks. For popular applications, we can expect a huge number of users to both contribute with sensor data and request information from the system. In such scenario, scalability of data processing becomes a major issue. In this paper, we present a system for supporting participatory sensing applications that leverages cluster or cloud infrastructures to provide a scalable data processing infrastructure. We propose and evaluate three strategies for data processing in this architecture.

**Keywords:** participatory sensing, distributed processing, mobile computing.

## 1 Introduction

Participatory Sensing [2,5] 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 [8,11,16], which concern road conservation and traffic monitoring, based on accelerometer and GPS readings collected by mobile devices.

Realizing the potential of Participatory Sensing poses several challenges, in particular, that of large-scale support, since the appeal of the paradigm can make its applications popular, with a very large number of users contributing with sensor data and obtaining information from the system. Many of the application examples found in the literature [8,11,16] use centralized architectures with a single server to process sensor data received from mobile users. These approaches are appropriate only for small-scale, proof-of-concept experiments.

For supporting large communities, a single server will be insufficient to process all data and reply to all queries. Thus, there is a need for partitioning data processing among multiple node. One possible approach is to use a decentralized solution composed by nodes scattered across the Internet. For example, in [9], we

* This work was supported partially by project #PTDC/EIA/76114/2006 and PEst-OE/EEI/UI0527/2011 - CITI/FCT/UNL/2011-12.

A. Puiatti et al. (Eds.): MobiQuitous 2011, LNICST 104, pp. 75–87, 2012.

have proposed such a system, with data processing being performed in a peer-to-peer network with nodes contributed by participants in the system. Such completely decentralized solutions involve a lot of complexity for achieving good performance and fault tolerance.

An alternative approach is to leverage current cluster and cloud computing infrastructures and perform data processing in such infrastructures. Several solutions for data processing in these environments have been proposed recently [6,13,4], but few are appropriate for processing streams of continuous data, as required by real-time participatory sensing applications. The solutions fit for processing continuous data (e.g. [4]) only focus on server execution, while a Participatory Sensing system can start data processing in the mobile nodes.

In this paper we present the Cloud 4 Sensing (C4S) system for supporting participatory sensing applications, allowing application developers to define the computation performed in the system. This includes the computations performed in the mobile nodes and in the system servers executing either in a cluster or in a cloud computing infrastructure. Computation is partitioned across server nodes according to a distribution strategy that defines how data is acquired by the system, where it is processed, and how it is aggregated. We propose three distribution strategies for processing data and evaluate these strategies in the context of a traffic monitoring application.

If the way programmers express computations is mostly inherited from our previous work [9], C4S runs in a completely different computation infrastructure (cluster/cloud vs. peer-to-peer). This leads to differences in the way information is propagated by mobile nodes and processed in the servers. Additionally, unlike our previous work, we propose several different distribution strategies.

The rest of the paper is structured as follows. Section 2 provides an overview of the C4S system. Section 3 details the three processing strategies proposed in this paper, followed by the results of their experimental evaluation in Section 4. The paper concludes with an overview of related work and some conclusions, respectively, in Sections 5 and 6.

## 2 System Overview

### 2.1 System Architecture

The C4S system architecture (Figure 1) comprises two kinds of nodes. A large number of mobile nodes, equipped with sensors, is the main source of data for the system, whereas a much smaller set of data center nodes forms a fixed infrastructure that provides computing, storage and networking resources.

Mobile nodes, typically smartphones, are expected to have limited computational power, battery life and communications capabilities. As such, they will mainly support user interaction and run data acquisition and simple processing services on behalf of the applications hosted at the fixed infrastructure. Mobile nodes do not interact directly, and connect to the fixed infrastructure to upload sensory data and obtain processed information via a frontend service.

Fixed nodes are deemed to be server-grade machines, well connected by a high-throughput and low-latency data center network. They form a loose overlay network, where the key characteristic is that they all know and can interact with each other. Moreover, each fixed node is assigned with a set of virtual geographic positions, chosen randomly from each of the target sensing areas. These locations are the basis for supporting decentralized processing strategies based on geographic partitioning. And, they also determine how mobile nodes bind to the fixed infrastructure.

### 2.2 Data Model

Applications access participatory sensing data by issuing queries over *virtual tables* - the main high-level data abstraction provided by C4S for naming a collection of data with a particular schema and semantics. Virtual tables have a global scope within a particular instance of the C4S 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.

### 2.3 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

**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) ])
} }
```

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 for specifying virtual tables and data transformations is detailed elsewhere [9].

A key aspect about data processing in virtual tables is that it is split between two transformation pipeline stages. One, *dataSource*, processes data locally available to the node, i.e., data already stored locally, or data that is being acquired or uploaded by mobile nodes. The *globalAggregation* merges the contributions of several nodes. Simple pipelines, with no *globalAggregation* stage can run entirely in mobile devices, and produce transformed data to feed another pipeline.

The two pipeline stages abstract the C4S 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 data aggregation infrastructure used, as discussed further on.

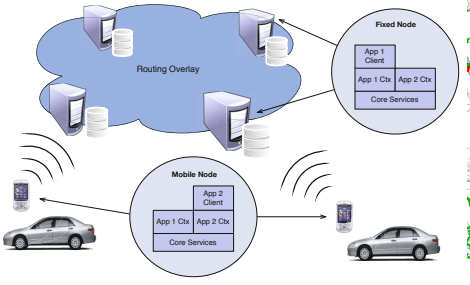


**Fig. 1.** C4S Architecture

**Fig. 2.** Snapshot of the traffic hotspots application for the Lisbon metropolitan area, showing congested spots in dark/red

### 2.4 Data Dissemination

When a node issues a query, the system instantiates data processing pipelines in the necessary fixed infrastructure and mobile nodes, depending on the geographic coverage of the standing queries, and according to the distribution strategy in use, as discussed in the next section. For returning query results to clients, C4S adopts a push-based model that leverages a publish/subscribe, content-based routing substrate. So, issuing a query translates to the client subscribing data conforming to a particular pattern/type, confined to a certain geographic area of interest and time period.

C4S is primarily designed for handling queries covering a large fraction of the target sensing areas. Supporting many "narrow" queries is made possible and efficient by merging them whenever they overlap geographically. For simplicity, the extent of the resulting compound query is used. The goal is to minimize the number of pipeline instantiations and avoid redundant data processing. To that end, the publish/subscribe substrate is used to filter out the query results that are outside of a client's request.

## 3 Distribution Strategies

Distribution strategies specify how the pipeline specification is materialized in the C4S infrastructure. Each strategy determines how data is acquired by the system, where it is processed, and how it is aggregated. Different strategies strive for different strengths. We next describe the three strategies we have implemented in the C4S platform.

### 3.1 RTree

The main rational behind RTree (Random Tree) is to leverage a simple partitioning of the acquired sensor data, backed by a random tree to aggregate and process data towards a root node of the fixed infrastructure.

In RTree, each mobile node maintains a long lasting association to a particular fixed node, established by selecting the fixed node with virtual position closest to a static reference location that defines the usual, general whereabouts of the mobile node. As a result, acquired data is immediately available for processing at each fixed node, but it is also partitioned among the fixed nodes without taking into account the actual spatial attributes of each sample. Therefore, the determining aspect of RTree is that every fixed node potentially hosts data relevant to any particular query.

In a given RTree instance (associated with a particular query), fixed nodes are organized into a random tree of a chosen degree, where they perform 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 along the tree until the root is reached, leading, potentially, to increased processing latency and raising issues of load inbalance. In particular, this can be an issue for virtual tables whose inference logic requires detecting the absence of certain data patterns. Early detections can, however, be published as soon as they are produced in any point of the aggregation tree. A straightforward way to address load balancing issues in RTree is to build a new random aggregation tree periodically to average the load of the nodes over time.

### 3.2 QTree

In QTree (Quad Tree), the idea is to do an *a priori* geographic partition of the data by performing a regular recursive subdivision of the sensing space into quadrants. 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 samples acquired by any mobile node. As for the global aggregation pipeline stages, QTree organizes them into a random quad tree, which is built, recursively, by picking a random node located in each subquadrant.

The key characteristic of QTree lies in that all the data processed in the deepest levels of the aggregation tree pertains to the immediate neighborhood of the processing node. This enables QTree to potentially detect highly localized phenomena early in the aggregation tree. As in RTree, load balancing issues may be mitigated by rebuilding the aggregation tree periodically.

### 3.3 NMap

NMap (Node Map) partitions data based on a geographic hashing approach. Each incoming data tuple is hashed to a geographic position and assigned to the fixed node closest to that point - its *neareast neighbour*. The hashing function can be tailored to the type and attributes of the sensor data, provided it preserves data locality. For instance, data that refers to a geographic extent can use the extent's centroid as the input for the hashing function. Besides spatial proximity,

other attributes can be used to cluster data and enhance the notion of proximity. The idea is to process geographically related data tuples on the same node to reduce the overall computation latency and, consequently, maximize the chances for earlier pattern detection.

NMap uses a bottom up approach to build the aggregation structure. Initially, in each node, the dataSource pipeline will process data samples acquired by any mobile node. The resulting data tuples are then hashed and forwarded to the corresponding nearest neighbour nodes, where they are processed by their respective global aggregation pipelines. Since data can be re-maped at the global aggregation stage with new coordinates, the process can be repeated to create a multi-level aggregation path that converges at some virtual rendezvous point. Provided the nodes share a consistent membership view of the fixed node infrastructure, closely related data tuples (according to the hashing functions used) will flow towards the same final processing node. As such, NMap provides a distributed processing model that has similarities to the Map/Reduce model [6].

In NMap, load-balancing issues may arise if too much data is allowed to flow towards a small subset of the available processing nodes. This could happen, for instance, as a consequence of particular patterns in the sensory data or the hashing functions used. To address this issue, besides using different hashing functions to different virtual tables, one possible solution is to change periodically the virtual geographical positions assigned to the nodes.

## 4 Case-Study and Experimental Evaluation

Performance metrics of the three distribution strategies were evaluated using a case-study application that concerns realtime road traffic monitoring. The application, *Traffic hotspots*, provides information about the congested areas based on GPS data, sampled at periodic intervals by in-transit vehicles. It relies on the TrafficHotspots virtual table, presented earlier as Definition 1, which supports querying for congestion detections computed from average speeds.

In the TrafficHotspots's *dataSource* pipeline stage, GPS samples are augmented to include the identifier of the road segment the vehicle is in. In this case, as this step requires accessing a database of road segments, it runs in the fixed nodes. For each road segment, an *AggregateSpeed* sample is generated with aggregate information of the cars in the segment. These *AggregateSpeed* samples are propagated every 10 seconds for processing by the *globalAggregation* stage. In that stage, a sliding window of the *AggregateSpeed* samples received in the last 10 seconds is maintained. For each road segment, the total number of cars 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 crosses some thresholds (that depend on the type of road). A client application can use these detections, for instance, to render maps of the current state of the target sensing area, as shown in Figure 2.

### 4.1 Evaluation

For the system evaluation, we use the Open Street Map [18] vectorial representation of the road network of Lisbon city. This 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 simulation experiments, 50000 mobile nodes populate the sensing area. They simulate in-transit vehicles, according to a traffic model, and report GPS readings every 5 seconds, as they follow the assigned paths. They interact with the infrastructure frontend, resulting in the delivery of raw GPS data with no latency. For the fixed insfrastructure, besides a single server solution, we evaluated performance using 50 and 500 fixed nodes. In this case, fixed nodes are assigned virtual positions distributed randomly across the sensing area.

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 2 shows a rendering of the traffic simulation.

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

### 4.2 Workload Distribution

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

The workload is measured as the number of data tuples processed in both *dataSource* and *globalAggregation* stages at each node. Figure 3 shows the workload distribution using the different distribution strategies, when considering 50 and 500 fixed nodes, for the top 25 most loaded nodes. The workload is presented as a percentage of the workload experienced by a single, centralized node.

The results show that when using the proposed strategies, both QTree's and NMap's most loaded node processes only a fraction of the tuples of a single

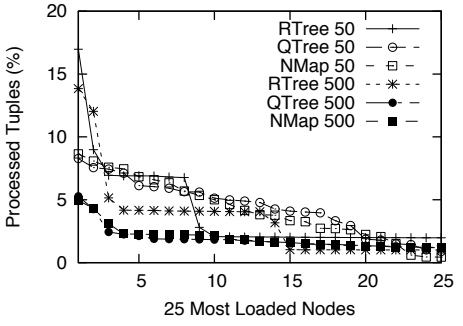


**Fig. 3.** Workload distribution

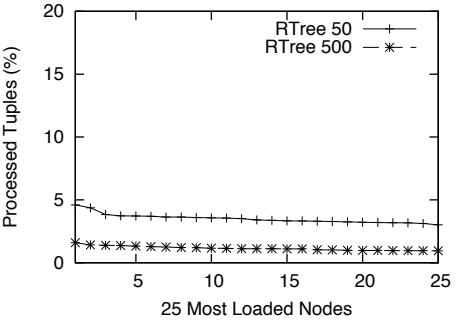


**Fig. 4.** Workload - RTree Rebalance

node solution (below 9% when considering 50 fixed nodes, and below 5% when considering 500). These values show the scalability of our solution - for example, C4S could still work properly even if the workload of the system exceeds by 10× the workload that a single node can process.

RTree presents a clearly skewed load distribution, with some nodes (those at the top of the aggregation) doing significantly more work. To address this load-balancing issue, RTree can rebuild the aggregation tree from time to time. Figure 4 summarizes the results when the tree is rebuilt every 10 minutes (for a total of 2 hours). It shows clearly that just a few tree updates are very effective at better spreading the load among the infrastructure nodes.

### 4.3 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 5 plots the detection latency for the accumulated level of successful detections. Results show clearly that QTree and NMap detect congestions faster than RTree, regardless of the number of considered fixed nodes. Their performance is comparable to the centralized solution. In RTree aggregation progress is slower because it is maily driven by the degree of the tree, rather than by strong spatial coherence, as in QTree or NMap.

**Failures.** We implemented a simple failure recovery mechanism consisting in rebuilding the aggregation tree periodically or when failures exceed a certain threshold in RTree and QTree. For NMap, sensory data is distributed disregarding the failed nodes upon their detection.

Our experiments allowed us to conclude that different distribution strategies present different behaviors that are hard to reflect in average results. Figure 6

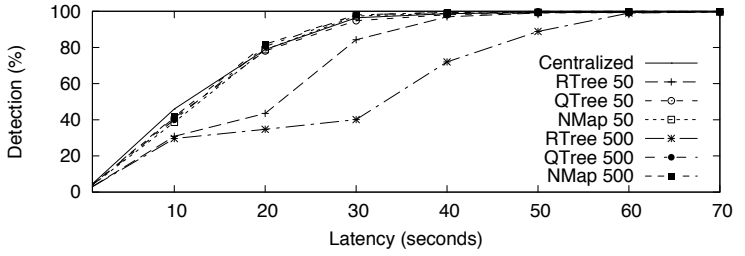


**Fig. 5.** Detection latency

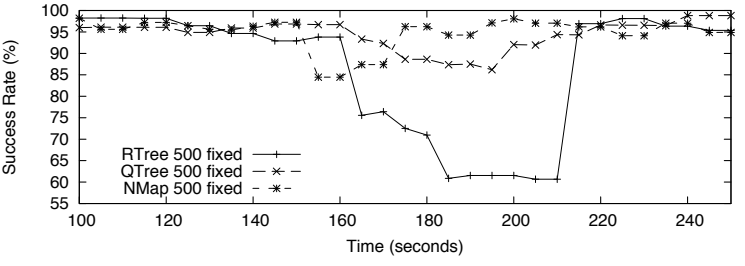


**Fig. 6.** Success rate in the presence of node failures

shows a selected run for each distribution strategy that exemplifies the impact of failures, when observed.

In RTree, the influence in the success rate observed depends hugely on the position of nodes failed in the aggregation tree. When nodes are mostly leaf nodes, little to no influence is observed, as data redundancy still allows detections to be observed. On the other hand, when nodes higher in the tree hierarchy are affected, the impact in the success rate is high.

QTree exhibits a similar behavior with stronger consequences for failures at lower levels and weaker consequences for failures at higher levels. The former can be explained by the fact that sensory data for a given region is processed by fewer nodes - the failure of a few of these nodes has an important impact in the final result. The latter can be explained by the fact that by aggregating the sensory data closer to the leaves, detections are often performed in lower levels of the tree, which makes inconsequent a failure in an higher level node.

NMap is the extreme case of impact in the failure of nodes - the failure of a node will always lead to a drop in the success rate, as results computed by that node are lost. However, as the recovery solution implemented is simpler, the success rate returns to the initial values quickly.

## 5 Related Work

Our work has been influenced by works in different areas.

Participatory sensing has become popular recently [2,5], and a large number of supporting systems have been designed. Some systems (e.g., BikeNet [7], PEIR [17], CarTel [11,8]) rely on a centralized approach, where mobile nodes

propagate sensing data to a central node responsible for processing information. Our system is also based on a centralized approach, but unlike previous works focus on the scalability of data processing in the servers.

In other systems, computation is performed by a combination of fixed nodes scattered across the Internet (e.g., COSMOS [15], SenseWeb [10], 4Sensing [9]). Some of these solutions address the scalability issue by spreading the load across multiple nodes in a peer-to-peer fashion. Although these solutions are interesting, the churn usually observed in nodes of a peer-to-peer network make solution complex for providing adequate quality of service and fault tolerance, and incur in a large communication overhead. Our solutions are inspired in some of the ideas proposed in these works, adapted to the target environment.

Sensor networks [1], while operating under a different context and communication assumptions, provided us with inspiration on how to gather and share data among nodes. TinyDB [14] extends the standard SQL syntax with a continuous query semantics, providing highly flexible queries as well as summarization and event detection. Directed Diffusion [12] 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 our system, 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 algorithms have been proposed for processing information [21]. However, these algorithms tend to focus on minimizing energy usage and nodes can only communicate with nearby nodes. Thus, the topologies that are formed to aggregate results have very different constraints when compared to our approach.

A large number of solutions have been proposed for stream and data processing - e.g. [3,13,6,4]. While these systems usually only focus on the processing in the servers, our solution spans both the servers and mobile nodes, which feed geo-referenced data into the servers periodically. Nevertheless, our work is inspired by some of these works. In NMap, data of a given region is processed by a single node in a way similar to a map-reduce computation [4], where the reduce step processes all data that has been mapped to the same key.

Quad trees have been previously used for a large number of goals, including for addressing geographical-related problems concerning the monitoring and tracking of moving objects [20] and for querying data in peer-to-peer networks [19]. QTree is inspired by these works, building a query specific tree during query dissemination.

## 6 Conclusions

This paper presents a system for supporting Participatory Sensing applications, focusing on data processing. In our system, data processing spans both mobile nodes and a set of server nodes, deployed in a cluster or cloud infrastructure. We present three algorithms for performing data processing in the servers. RTree is the simplest solution, leveraging random aggregation trees. QTree combines

random trees with regular, *a priori*, subdivision of geographic space for improved performance. In NMap, all closely related sensory data is aggregated and processed in a single node.

The experimental evaluation using simulation and a case-study application, allows us to draw the following conclusions. The three distribution strategies distribute the load across multiple nodes, with the most loaded node processing only a fraction of the tuples of a single node solution. NMap and QTree exhibit overall better load balancing when compared with RTree. The results also show that NMap and QTree can compute results faster than RTree, and with a latency similar to a single node solution. Thus, C4S with either NMap or QTree presents a scalable solution for processing data in participatory sensing applications. Compared to QTree, NMap offers the additional flexibility afforded by the use of hashing functions to distribute (and cluster) processing, as they can be tailored to match the application scenario closely.

As for future work, we intend to continue studying solutions for improving load balancing, processing overhead and query success and latency in all strategies. Additionally, we intend to extend the support for processing information in mobile nodes, thus further reducing the load experienced by server nodes and helping to improve the overall scalability of the system.

## References

1. Akyildiz, I.F., Su, W., Sankarasubramaniam, Y., Cayirci, E.: A survey on sensor networks. IEEE Communications Magazine 40(8), 102–114 (2002)
2. Campbell, A.T., Eisenman, S.B., Lane, N.D., Miluzzo, E., Peterson, R.A., Lu, H., Zheng, X., Musolesi, M., Fodor, K., Ahn, G.-S.: The rise of people-centric sensing. IEEE Internet Computing 12(4), 12–21 (2008)
3. Cherniack, M., Balakrishnan, H., Balazinska, M., Carney, D., Çetintemel, U., Xing, Y., Zdonik, S.B.: Scalable distributed stream processing. In: CIDR (2003)
4. Condie, T., Conway, N., Alvaro, P., Hellerstein, J.M., Elmeleegy, K., Sears, R.: Mapreduce online. In: NSDI 2010: Proceedings of the 6th USENIX Symposium on Networked Systems Design and Implementation (2010)
5. Cuff, D., Hansen, M., Kang, J.: Urban sensing: out of the woods. Commun. ACM 51(3), 24–33 (2008)
6. Dean, J., Ghemawat, S.: Mapreduce: simplified data processing on large clusters. In: Proc. 6th Symp. on Operating Systems Design & Implementation (2004)
7. Eisenman, S.B., Miluzzo, E., Lane, N.D., Peterson, R.A., Ahn, G.-S., Campbell, A.T.: The bikenet mobile sensing system for cyclist experience mapping. In: SenSys 2007: Proc. 5th Int. Conf. on Embedded Networked Sensor Systems (2007)
8. Eriksson, J., Girod, L., Hull, B., Newton, R., Madden, S., Balakrishnan, H.: The Pothole Patrol: Using a Mobile Sensor Network for Road Surface Monitoring. In: Proc. 6th Int. Conf. on Mobile Systems, Applications, and Services (June 2008)
9. Ferreira, H., Duarte, S., Preguiça, N.: 4Sensing - Decentralized Processing for Participatory Sensing Data. In: 16th International Conference on Parallel and Distributed Systems (ICPADS 2010). IEEE (2010)
10. Grosky, W., Kansal, A., Nath, S., Liu, J., Zhao, F.: Senseweb: An infrastructure for shared sensing. IEEE Multimedia 14(4), 8–13 (2007)

11. Hull, B., Bychkovsky, V., Zhang, Y., Chen, K., Goraczko, M., Miu, A.K., Shih, E., Balakrishnan, H., Madden, S.: CarTel: A Distributed Mobile Sensor Computing System. In: 4th ACM SenSys (November 2006)
12. Intanagonwiwat, C., Govindan, R., Estrin, D.: Directed diffusion: a scalable and robust communication paradigm for sensor networks. In: MobiCom 2000: Proc. 6th Int. Conf. on Mobile Computing and Networking, pp. 56–67 (2000)
13. Isard, M., Budiu, M., Yu, Y., Birrell, A., Fetterly, D.: Dryad: distributed data-parallel programs from sequential building blocks. In: Proc. 2nd EuroSys European Conference on Computer Systems, EuroSys 2007, pp. 59–72 (2007)
14. Madden, S.R., Franklin, M.J., Hellerstein, J.M., Hong, W.: Tinydb: an acquisitional query processing system for sensor networks. ACM Trans. Database Syst. 30, 122–173 (2005)
15. Marie Kim, Y.J.L., Lee, J.W., Ryou, J.-C.: Cosmos: A middleware for integrated data processing over heterogeneous sensor networks. ETRI Journal 30(5) (October 2008)
16. Mohan, P., Padmanabhan, V., Ramjee, R.: Nericell: Rich Monitoring of Road and Traffic Conditions using Mobile Smartphones. In: Proceedings of ACM SenSys 2008 (November 2008)
17. Mun, M., Reddy, S., Shilton, K., Yau, N., Burke, J., Estrin, D., Hansen, M., Howard, E., West, R., Boda, P.: Peir, the personal environmental impact report, as a platform for participatory sensing systems research. In: MobiSys 2009: Proc. of the 7th Int. Conf. on Mobile Systems, Applications, and Services, pp. 55–68. ACM (2009)
18. OpenStreeMap (April 2010), `http://www.openstreetmap.org`
19. Tanin, E., Harwood, A., Samet, H.: Using a distributed quadtree index in peer-to-peer networks. VLDB Journal 16, 165–178 (2007)
20. Tayeb, J., Ulusoy, Ö., Wolfson, O.: A quadtree-based dynamic attribute indexing method. Comput. J. 41(3), 185–200 (1998)
21. Yick, J., Mukherjee, B., Ghosal, D.: Wireless sensor network survey. Comput. Netw. 52(12), 2292–2330 (2008)