Automatic Extraction of POIs in Smart Cities: Big Data Processing in ParticipAct

Antonio Corradi, Giovanni Curatola,
Luca Foschini, Raffaele Ianniello
DISI - University of Bologna, Italy
{antonio.corradi, luca.foschini, raffaele.ianniello}
@unibo.it; giovanni.curatola@studio.unibo.it

Carlos Roberto De Rolt
Centro de Ciências da Administração e
Socioeconômicas - CCA / ESAG
Universidade do Estado de Santa Catarina - UDESC
rolt@udesc.br

***Abstract*—Recent advances in sensor-equipped smartphones are opening brand new opportunities, such as automatically extracting Points Of Interest (POIs) and mobility habits of citizens in Smart Cities from the large amount of harvested data hotspots. At the same time, the high dynamicity and unpredictability of Smart Cities crowds, opportunistically collaborating toward these common crowdsensing tasks, introduces challenging issues due to the need for fast and continuous processing of these Big Data Streams in the backend of next generation crowdsensing platforms. This paper presents our practical experiences and lessons learnt in deploying the ParticipAct platform and living lab, an ongoing experiment at University of Bologna that involves 300 students for one year. Among all management issues addressed in ParticipAct, this article shows the integration of MongoDB in the ParticipAct backend, as a powerful NoSQL storage and processing engine to fasten the identification of POIs; the reported performance results confirm the feasibility of the approach by quantifying its advantages for city managers.**

Keywords: Big Data; Crowdsensing; Points Of Interest; Performance Evaluation.

I. INTRODUCTION

In 2013, for the first time, have been shipped more smartphones than feature phones, so marking a trend that brings the former to permanently outnumber the latter. This explosive and widespread diffusion of powerful smartphones equipped with several sensors, such as accelerometers, barometers, cameras, and microphones, enables a wide range of possible data gathering and processing activities. The new scenario makes information so densely available and harvested to pave the way for crowdsensing, in the sense of allowing for a very large-scale fine-grained sensing, by exploiting all personal resources and people's mobile activities/collaborations.

At the same time, Smart Cities themselves can also make available several resources and data to be managed and harnessed safely, sustainably, cost-effectively, and efficiently to achieve positive and measurable economic and societal advantages. Indeed, these new forms of collective intelligence (even if inaccurate) should be able to self-organize spontaneous collaboration of citizen groups involved with other people to get, through their collective actions, a common goal with a tangible effect also on the physical material world, namely e-Participation and e-Inclusion. At the current stage these forms of organization are still largely unexplored [1].

We consider essential to fill that gap by proposing a new class of socio-technical platforms that provide people with the innovative opportunities to participate and act collectively . Our intent is to enable new models for the development of a consciousness/environmental awareness focused on the citizen as a consumer and user of the quality and environmental impact of goods and services that they expect in their Smart Cities.

To study and advance the knowledge in this new field, we focused on crowdsensing management and we developed the ParticipAct crowdsensing platform and living lab testbed, an ongoing experiment at the University of Bologna that involves 300 students for one year in a large-scale crowdsensing campaign. In ParticipAct, we are currently addressing all the manifold social and technical management issues. From the more social point of view, there are many duties involved in people participation, such as the identification of people willing to participate in crowdsensing campaigns [2, 3, 4], how to keep them involved (e.g., by providing attractive crowdsensing and participatory services, entertainment, and rewards) [1, 5], and how to foster their participation with active collaboration actions in some data collection campaigns, for instance that requires people to operate at a specific location (e.g., taking a picture of a monument, tagging a place, etc.) [3, 6]. From the technical point of view, it is important to balance sensing accuracy and user resource utilization to avoid making the crowdsensing process cumbersome to users [7, 8], and to process raw sensing data aboard the smartphone in order to enable high-level inferences (e.g., to infer from raw accelerometer samples the user activity, such as running, standing, and walking) [9].

Among all above challenging management issues, this paper addresses the crucial technical problem of enabling a powerful and fast processing engine to analyze large datasets and to extract from them proper geo-social/preferences profiles integrating it in the backend of the ParticipAct platform. In fact, although information gathered from people, systems, and things, including both social and technical information, is one of the most valuable resources available to city stakeholders, its potentially enormous volume makes difficult its integration and processing, especially in a real-time and scalable manner.

In particular, this paper proposes a novel flexible and highly interoperable solution to address the whole crowdsensing Big Data creation, management, and presentation process that presents several core original

elements. First, we present ParticipAct, a complete crowdsensing platform including an app running on participant smartphones, and a web application on the back-end taking care of the whole crowdsensing process. Second, we describe the novel Big Data processing engine to complete all needed crowdsensing data processing steps, from data storage in the highly scalable NoSQL MongoDB [10], to Map-Reduce-powered interrogations, and to the integration with Geographic Information System (GIS) for intuitive presentation of the obtained results. Third, we describe a real and interesting use case that we developed to automatically gather from the mobility and user activity data traces the Points of Interest (POIs) of students taking part in ParticipAct. Fourth, we present a thorough performance analysis to quantitatively compare our implementation based on MongoDB with another one based on the MySQL cluster distributed SQL database: all collected results are based on the Big Data ParticipaAct dataset populated by students taking part to the ParticipAct living lab experiment that involves 300 student for one year here at the University of Bologna in the cities of Bologna and Cesena.

The reminder of this paper is structured as follows. Section II gives the necessary background and definitions about Big Data management in ParticipaAct and MongoDB. Section III presents MongoDB integration and management into the ParticipAct backend, and gives some implementation details about POI extraction. Section IV shows extensive experimental results. Section V analyzes related works, and Section VI reports conclusive remarks and concludes the paper.

II. PARTICIPACT BIG DATA MANAGEMENT

Big Data processing is a core facility of new emerging crowdsensing platforms. In fact, in its simplest formulation, crowdsensing is a two-step process to i) assign sensing tasks to users and to ii) wait for results after assignment completions. Effective and precise assignments can be enabled only by exploiting the large amount of information about potential participants and their mobile execution context (people geographical location, frequency of visits in an area, etc.) to better and more effectively tailor the task assignment process, such as to automatically extract POIs where they spend their working and resting hours. That requires to process fast and, possibly, efficiently the large set of data collected about all participants.

This section presents the needed background about our crowdsensing platform ParticipAct, and introduces the main characteristics of MongoDB, the NoSQL database used to realize the ParticipAct Big Data processing engine.

A. *ParticipAct platform*

A crowdsensing platform typically adopts a client-server architecture including a client, running on user devices to manage tasks and to run all required sensing activities interacting with participants via their smartphones, and a server to store, analyze, and present collected results [11].

The ParticipAct client (see Fig. 1) is the component that takes care of receiving tasks, asking users whether they want to run them, managing data collection, and uploading results. Functionally, ParticipAct client comprises two main components: the *task management* component and the *sensing management* component. These components are responsible for both interacting with users and accessing smartphone sensors.

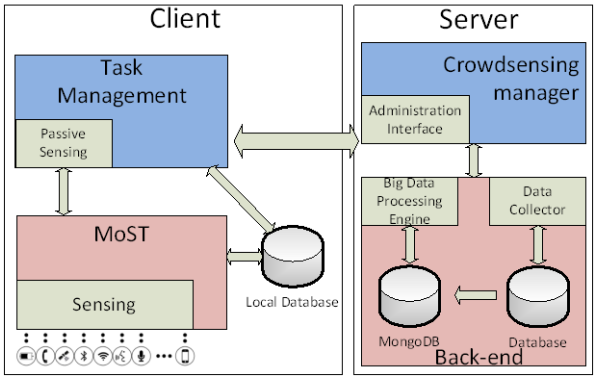


Fig. 1 ParticipAct architecture

In particular, the *task management* component takes care of overseeing the whole task lifecycle on smartphones, from managing tasks to provide users with an interface to control task execution, including the possibility to stop any sensing activity to preserve user privacy, to the final upload of sensed data. The sensing management component, instead, plays a pivotal role because it is in charge of accessing all the sensors available on smartphones and collecting and processing their output through Mobile Sensing Technology (MoST), an open-source and lightweight Android sensing library that includes several physical and virtual sensors providing both raw data and high-level inferences[1]. For additional details about client side components of ParticipAct, out of the scope of the present paper, we refer interested readers to [2, 6, 9].

The server side of ParticipAct (see Fig. 1) provides management, storage, and analysis of crowdsensed data. At the highest level comprises two main parts: the *back-end* and the *crowdsensing manager*. The *back-end* takes care of receiving, storing, and processing sensed data, while the *crowdsensing manager* provides the administration interface to design, assign, and deploy sensing tasks.

In a more detailed view, the *back-end* realizes needed communication functions to exchange tasks and receive results, ensuring authenticity, integrity, and confidentiality via a Representational State Transfer (REST) API [12]. Most notably, the back-end manages the whole data lifecycle supported by two main components, namely, the *data collector* that cleans up incoming data and prepares them for long-term storage, and the Big Data *processing engine* that enables the execution of more complex queries over the whole ParticipAct database and represents the core contribution of this paper. The main data collector functions are data *interpolation* to improve data collection by filling in missing data points that can be inferred with sufficient accuracy, and data *integration* to process data after the interpolation step to aggregate them in time and space, such as collapsing all data of any type collected in the same 5 minute window in a single row to enable fast time-based data querying. As for the *processing engine*, it represents the core contribution of this paper and allows to create and to execute general-purpose

[1] An updated list of sensors wrapped by MoST and the list of high-level inferences on user activities is available at http://participact.unibo.it.

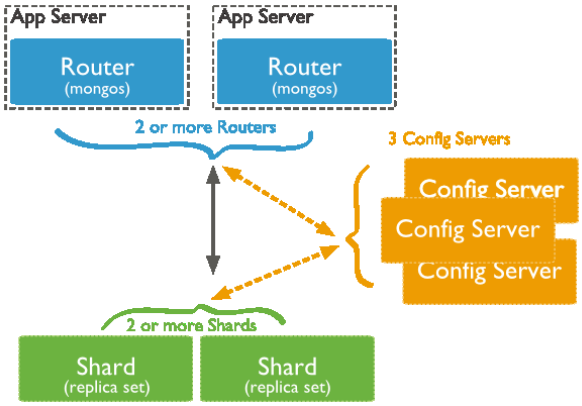


Fig. 2 MongoDB sharded cluster architecture

queries over the whole database. Fine-grained data received from clients are stored on a relational database based on PostgreSQL/PostGIS; then, they are transferred to MongoDB database. Indeed, NoSQL databases such as MongoDB are less suited for storing fine-grained data, but can boost data analysis and fasten query execution.

Finally, the *crowdsensing manager* is the administrator-facing part of ParticipAct that exploits back-end exported functions to provide administration features. The *Web administration interface* allows full administration of the whole crowdsensing, including management of user profiles, design and assignment of tasks, and data review.

After this first overview of the whole ParticipAct architecture, in the following we will focus only on the processing engine (interested readers can find more details on the ParticipAct backend in [2]); in particular, in next subsection we introduce MongoDB, the NoSQL database used to realize this core back-end facility.

B. *Mongodb-enabled big data management in ParticipAct*

MongoDB is a widely diffused and accepted open/source NoSQL database; without any pretense of being exhaustive, in the following we briefly introduce main MongoDB concepts and components.

NoSQL databases can store data in several different ways, such as key-values store, wide column store, and document stores. MongoDB is document-oriented and a document is a data structure that contains fields and values where values can be of different types, including documents and array of documents. Collections aggregate multiple documents and play a similar role of tables in relational databases. The main difference is that NoSQL databases have no schema so it is possible to insert into a collection non-uniform records, records with missing values, and to add fields to some record without having to resettle the whole database.

MongoDB has been designed to be horizontally scalable and capable of managing very large datasets. To reach this objective it is possible to distribute datasets on different nodes through the sharding technique. Every shard is an independent database and the union of all shards realizes a unique logical database. Documents are saved in files called chunks. Every chunk has a limited maximum dimension and, based on configuration files, MongoDB actuates automatic shard balancing by moving chunks through nodes in order to distribute computational load among them. Finally, query routers, or simply routers, interface MongoDB with applications by redirecting queries to shards and providing results to clients. Fig. 2 shows the MongoDB sharded cluster architecture.

Moreover, MongoDB provides also a framework based on Map-Reduce to aggregate and manage query results obtained from stored collections. Map-Reduce is a programming model designed to process very large amounts of data in parallel, originally designed and promoted by Google. Two main concepts at the basis of this model are the *map* and *reduce* functions: users can define a *map* function that, given an input, generates an intermediate set of key-values couples, and a *reduce* function that merges together all the intermediate values associated with a same key. The framework automatically collects all the intermediate outputs and sorts them by key before executing the *reduce* function.

MongoDB applies the *map* phase to each input document, namely, the documents in the collection that match the query condition. The map function emits key-value pairs, and then the *reduce* phase collects and aggregates these intermediate results saving final results. Optionally, the output of the reduce function may pass through a *finalize* function to further condense or process the results of the aggregation.

All MongoDB Map-Reduce functions take the documents of a single collection as the *input* and can perform any arbitrary sorting and selection before beginning the map stage. Map-Reduce can return the results of a map-reduce operation as a document, or may write the results to collections. When using sharded collections as input of a Map-Reduce operation, the routing service that manages sharded collections will automatically dispatch the Map-Reduce job to each shard in parallel and wait for jobs on all shards to finish. After that it is possible to merge results on each node through the finalize phase.

Let us stress that the availability of this Map-Reduce framework completely integrated and ready-to-use with the NoSQL functions made of MongoDB the best candidate for the implementation of the ParticipAct Big Data processing engine.

III. BIG DATA-ENABLED CROWDSENSING DATA ANALISYS

This section discusses the possibility of combining different crowdsensing data in order to achieve an automatic identification of POIs for a specific user, and focuses on the ParticipAct implementation based on MongoDB and MapReduce as computation back-end.

A. *POI identification process*

GPS location is a kind of information that is very important in Smart Cities scenarios. It can be used to connect any other sensing task to a specific place, and then connecting it to other users and events happening in the same place. Moreover, it can be used to identify POIs for a specific user; this information is very valuable because it allows to identify user behavior and to predict user positions and main roaming habits over the day.

Automatic identification of user POIs is a non-trivial operation; the ParticipAct implementation consists of three main phases. The first phase selects, for each user, all the GPS locations that correspond to a user either still or walking; to

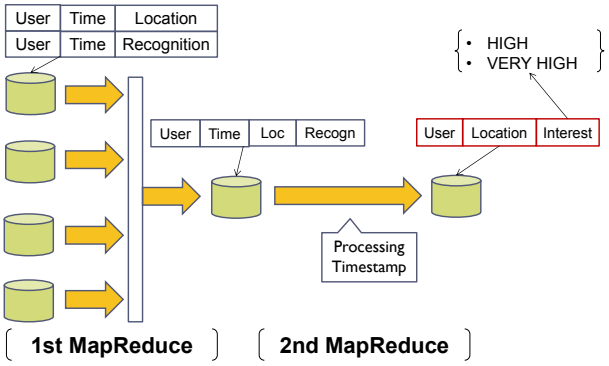


Fig. 3 POI Map-Reduce sequence schema

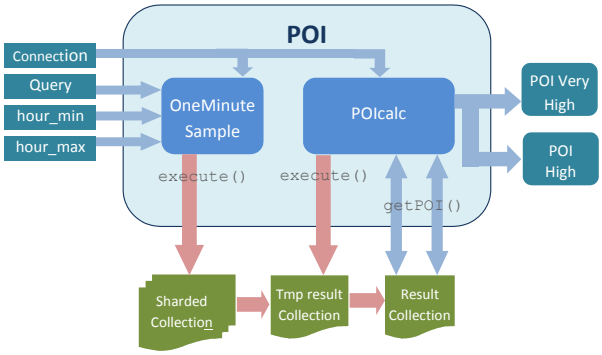


Fig. 4 POI module schema

grant higher accuracy this selection is done by melting together geographical GPS locations with user activity data obtained via accelerometer measurements. In particular, from the accelerometer sensor of a smartphone it is possible to obtain from MoST high-level user activity inferences that report if the user that carries it is still or moving and, depending on the readings of the sensor, even to discriminate how she is moving, such as if the she is walking, running, and moving by car or bicycle [6]. Once obtained these activity recognition samples (recognition in Fig. 3), it is necessary to aggregate them with the GPS samples for the same user. This operation, apparently easy, requires to collapse in a unique record samples with different timestamps. In fact, in order to save precious energy resources every different sensor in ParticipAct adopts a different sampling frequency. Because the GPS sensor is very expensive we found that a sample every 5 minutes is a good tradeoff to get reasonable localization accuracy and to avoid too fast battery drainage, while activity recognition reports a sample once per minute. This time-alignment operation terminates the first phase.

The second phase, instead, approximates all the GPS coordinates in the same vicinity (small rectangles of about 8x11 meters) as many samples of the same location, to discretize and reinforce locations where users stay longer.

Finally, the third phase aims to count the number of occurrences of visits by the user of the same place both within the same period of the day and over different days, so to select POIs with high and very high probability, as better detailed in the next section.

B. Map-Reduce-based POI extraction in MongoDB

We implemented the POI search in MongoDB by using the Map-Reduce support. We split the load among shards by using as identifier the unique ParticipAct user identifier: that grants that all activity recognition and GPS location data from the same user are all available on the same shard. Then, when we run our POI search for all users, MongoDB horizontally splits the load among the different shards for the different users.

The main computation is composed by two steps of Map-Reduce executed in waterfall followed by a finalize step (see Fig. 3). The first step (the first phase in the previous subsection) combines GPS data with activity recognition ones. This operation is executed by defining a key that uses user id and the same minute of timestamp; this means that the results have sample time of one minute. Then, it adds to GPS point the information of activity recognition, if it is classified as still or walking and reports the same key, while dropping the GPS data if activity recognition reports that the user is running or moving by a vehicle. This step is of paramount importance because when activity recognition data are absents, and so it is not possible to connect activity recognition with GPS, from our preliminary experiments we discovered that POI extraction performances was less effective.

The second step (second phase of the process) is another Map-Reduce that refines results from previous step to approximate nearby locations as the same location. In particular, the GPS location is approximated to the fourth decimal number using coordinates as signed decimal degrees. This means that POIs are discretized and pinpointed at the vertexes of a grid with sides of about 11 meters north-south and 8 meters east-west at our Bologna latitude. In this step, the key used by Map-Reduce is defined using user id and approximated coordinates, and the associated value is a list of timestamp corresponding to those occurrences.

The final step (third process phase) is a finalize step for the second Map-Reduce where the occurrences in a specific place are filtered and aggregated to calculate POI and their probability. A new occurrence is counted if a user comes back to the same place after at least an hour; otherwise, all subsequent samples close in time are considered as the same occurrence. The final result considers how many times a place has been visited during a period, and then classified according to this result as a POI with “high” or “very high” probability that correspond to a mean of at least one visit every two days or every day respectively. We decided to set a minimum of 10 days as range between first and last visit to consider a place as potential POI in order to eliminate accidental or circumscribed visiting events.

Figure 4 reports the internal POI extraction component architecture. The input allows to choose the time range to search and to specify the query to select data. The results are stored on the database and can be retrieved using getPOI() method that returns the data classified into the two sets “POI very high” and “POI high”.

IV. EXPERIMENTAL RESULTS

We have thoroughly validated and quantitatively evaluated the processing engine and the POI extraction function based on it via extensive results run over the Big Data ParticipAct dataset. In the following, after a brief introduction about collected data, we present some results that validate the integrated component. Then, we show a thorough performance analysis comparing execution time and resource consumption of the adopted implementation based on MongoDB and another

possible alternative one based on MySQL cluster, chosen as an example of a more traditional distributed SQL database.

A. *POI extraction validation*

We tested our implementation of POI search with MongoDB with a large dataset of GPS points gathered using ParticipAct app. This app is provided to about 300 students of University of Bologna for one year. During this time, the app sent to our server a GPS sample per user every 5 minutes resulting in 7.5 millions of GPS samples. This means a very high quantity of data to process, and a valuable test for the solution presented in this paper. For more limited periods of time, depending on the crowdsensing tasks required to our students, smartphones also reported activity recognition results. As explained before the proposed POI extraction process uses these data to refine POI search and drop a priori GPS points labeled as moving/transit. We developed also an intuitive Web GIS interrogation interface integrated with Google Maps that city managers can use to query POIs. For every user, we executed two different POI extractions for two different time slots, namely, [9a.m., 7 p.m.] and [10p.m., 7a.m.] with the purpose of distinguishing between work POIs and residential POIs.

In order to validate obtained results, after running our POI extraction, we interviewed volunteer users asking for confirmation of extracted POIs. We interviewed about 10% of all users and we got a confirmation rate around 90% on residence position, while confirmations for the workplace were lower, around 65%. Indeed, analysis of the nighttime led to a more accurate result due to the unicity of the house. Analysis of daytime, instead, is more complex because students tend to have a more active and irregular behaviour with respect to a normal worker/employee, potentially stopping by at different schools and classrooms to attend their lessons, and typically being engaged in different activities during the day thus producing more than one POI.

B. *Performance tests*

Focusing on performance results, we start presenting the response time. We compared our solution with another solution implemented using MySQL Cluster and a version of the algorithm implemented in Java. MySQL Cluster has the possibility of replication and partitioning of datasets. A MySQL cluster is characterized by the presence of three types of nodes: i) management node that focuses on startup, shutdown, configuration, and backup of nodes in the cluster; ii) data node that contains data partition; iii) SQL node that manages the access to data stored when requested by external applications.

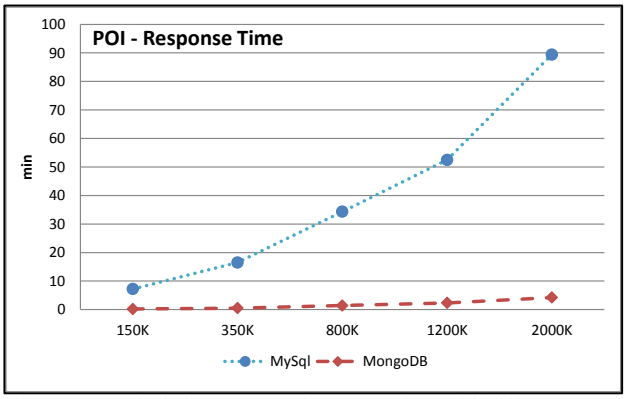
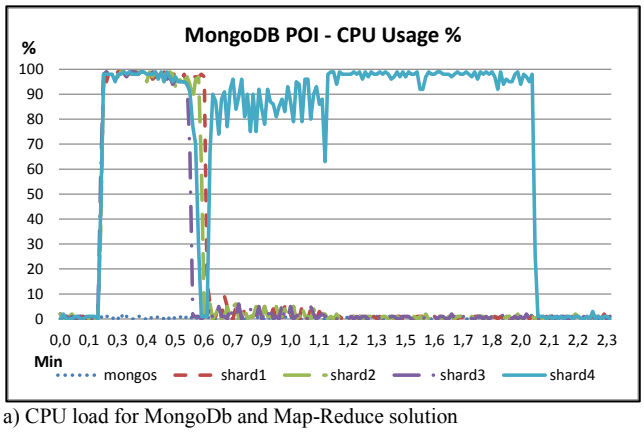


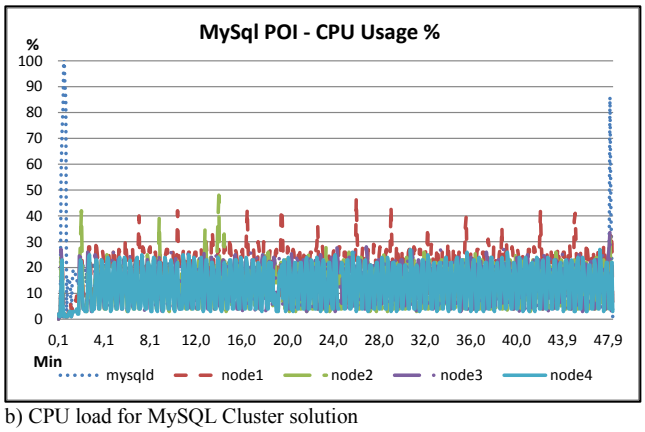
Fig. 5 POI calculation response time comparison with different dataset dimensions

In order to complete these tests, we used 3 physical hosts with 2 x 16 cores AMD OPTERON 6376 @ 1400 MHz, 32 GB RAM with Ubuntu 12.04 64 bit. Every node has 2 Virtual Machines (VM) with 2 GB of RAM running MongoDB 2.6.3 and MySql 5.6.19 MySql Cluster NDB-7.3.6. One node hosts the client that initiates the requests and the frontend of the application, together with Configuration Server. Other two nodes execute 4 instances of MongoDB with a shard of the dataset each. A similar configuration was used for MySQL Cluster scenario where 2 nodes host 4 VMs running the 4 data and management nodes; another node hosts the SQL node that receives incoming requests.

As we can see in Figure 5, we compared the completion time between the two systems using datasets with different dimensions. Results show that MongoDB solution is much faster and this difference grows with larger datasets. The main reason of this difference is in the ability of MongoDB, through to distribute execution of data nodes computation on Map-Reduce. Figure 6 shows that by comparing of CPU usage between the two solutions: it is clear that there is a much more intense load distribution between shards in MongoDB than MySQL Cluster. In fact, in MySQL Cluster the elaboration is made by the application that requests data while in MongoDB, using Map-Reduce, distributes the elaboration between shards and then the result is prompted to the application requesting



a) CPU load for MongoDb and Map-Reduce solution



b) CPU load for MySQL Cluster solution

Fig. 6 CPU load for 4 nodes configurations during POI search using 1200K documents of GPS data

services. Delving into finer details Figure 6.a also shows that in the second part of the execution of the algorithm only one node is working. This is a known problem of MongoDB when using an external application such as over Java-based Map-Reduce as source of interrogation instead of the MongoDB JavaScript console. Datasets are correctly distributed before starting elaboration, but at the end of the first Map-Reduce results are prompted together to the node that initiates the process, so they are located on one node only. Then, the second Map-Reduce is executed only in that node. This problem causes computation to take longer time. Finally let us stress that tests presented in figure 5 and 6 showed a very limited variance, always below 5% over all runs.

V. Related Work

Without any pretense of being exhaustive, given the large amount of literature in the field, in the following we present a selection of crowdsensing works that, similarly to ours, analyze location data adopting a profiling-based approach and considering user activities and spatial-temporal analysis on users' data history. We will start with systems that are farther from ParticipAct to terminate with more similar ones.

Matador focuses on context awareness to optimize task assignment [3]; each task is characterized by its context, which specifies when such a task should be triggered to the user or by the user's device. The resulting system aims at delivering the right tasks to the right people in the right circumstances. USense [8] is a middleware for community sensing that collects data and builds spatio-temporal data models and profiles of participating clients. Vita focuses on providing crowdsensing as a service integrated with usual software services, and on supporting sensing task assignment also based on user profiles and assigning tasks to users by using a "social vector" [4]; that concise representation of user resources and knowledge shares some similarities with ParticipAct POIs. Finally, PEIR exploits mobile phones to evaluate whether users have been exposed to airborne pollution, enables data sharing to encourage community participation, and estimates the impact of individual user/community behaviors on the surrounding environment [13]; PEIR allows users to share their location traces with people they trust.

All above systems system provide some backend processing facilities to extract meaningful and synthetic location information to help the assignment of future crowdsensing campaigns. However, they tend to adopt ad-hoc solutions and have not examined in depth impact on utilized resources for data analytics applied to large Big Datasets. In addition, to the best of our knowledge, none of them is considering real data traces for a large amount of 300 people for a long time span, such as in the ParticipAct living lab.

VI. Conclusions

Crowdsensing enables the collection of Big Data urban monitoring information without the need for costly infrastructure investments, as demonstrated by the ParticipAct project. The proposed MongoDB-based processing engine boosts the possibility to flexibly and efficiently drill and extract from this data meaningful and synthetic geo-spatial profiles, such as in the case of POIs.

On the basis of achieved results, we currently work along several new research lines. On the one hand, we are extending the MongoDB support to overcome some limitations, especially focusing on geo-spatial queries that are natively limited to a result set of 16 MB. On the other hand, we are realizing other Big Data applications for fast computation of geo-clustering algorithms aimed at selecting and assigning crowdsensing task to our participants, such as in the case of the DBSCAN clustering algorithm [14].

Acknowledgements

This research was supported in part by CIRI, Center for ICT technology transfer of the University of Bologna, funded by POR FESR Emilia-Romagna 2007-2013, and by CAPES - processo n. 8745-13-7 for Carlos Roberto De Rolt.

References

[1] B.J. Fogg, "A behavior model for persuasive design," Proc. of the 4th Int. Conf. on Persuasive Technology (Persuasive), pp. 1-7, 2009.

[2] G. Cardone, A. Cirri, A. Corradi, and L. Foschini, "The ParticipAct Mobile Crowd Sensing Living Lab: The Testbed for Smart Cities," IEEE Communications Magazine, vol.52 , n. 10, October 2014.

[3] I. Carreras, D, Miorandi, A. Tamilin, E. R. Ssebaggala, and N. Conci, "Matador: Mobile task detector for context-aware crowd-sensing campaigns," PERCOM Workshops '13: Proceedings of the IEEE International Conference on Pervasive Computing and Communications Workshops, pp. 212-217, 2013.

[4] H. Xiping, T. H. S. Chu, H. C. B. Chan, and V.C.M. Leung, "Vita: A Crowdsensing-Oriented Mobile Cyber-Physical System," IEEE Transactions on Emerging Topics in Computing, vol. 1, no. 1, pp. 148-165, 2013.

[5] G. Cardone, A. Corradi, L. Foschini, and R. Montanari, "Socio-Technical Awareness to Support Recommendation and Efficient Delivery of IMS-Enabled Mobile Services," IEEE Communications Magazine, vol. 50, no. 6, pages 82-90, June 2012.

[6] G. Cardone, A. Cirri, A. Corradi. L. Foschini, R. Ianniello, and R. Montanari, "Crowdsensing in Urban Areas for City-scale Mass Gathering Management: Geofencing and Activity Recognition," IEEE Sensors Journal, 2014.

[7] G. Cardone, L. Foschini, C. Borcea, P. Bellavista, A. Corradi, M. Talasila, and R. Curtmola, "Fostering ParticipAction in Smart Cities: A Geo-Social CrowdSensing Platform," IEEE Communications Magazine, vol. 51, no. 6, pp 112-119, June 2013.

[8] V. Agarwal, N. Banerjee, D.Chakraborty, and S. Mittal, "USense - A Smartphone Middleware for Community Sensing," Proc. of the IEEE Int. Conf. on Mobile Data Management (MDM), pp. 56-65, 2014.

[9] G. Cardone, A. Cirri, A. Corradi, L. Foschini, and D. Maio, "MSF: An Efficient Mobile Phone Sensing Framework," International Journal of Distributed Sensor Networks, vol. 2013, Article ID 538937, 9 pages, 2013

[10] "MongoDB," [Online]. Available: http://www.mongodb.org/.

[11] N. D. Lane, E. Miluzzo, H. Lu, D. Peebles, T. Choudhury, and A. T. Campbell, "A survey of mobile phone sensing," IEEE Communications Magazine, vol. 48, no. 9, pp. 140-150, 2010.

[12] R. T. Fielding and R. N. Taylor, "Principled design of the modern Web architecture," ACM Transactions on Internet Technology, vol. 2, no. 2, pp. 115-150, 2002.

[13] 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," MobiSys '09: Proceedings of the 7th International Conference on Mobile Systems, Applications, and Services, pp. 55-68, 2009.

[14] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise," KDD '96: Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining, pp. 226-231, 1996.